Kairos: Efficient High-Threshold Asynchronous Distributed Key Generation from Lightweight Cryptography

Xinxin Xing
xxxshixiannva@buaa.edu.cn
Beihang University
Beijing, China

Yizhong Liu*
liuyizhong@buaa.edu.cn
Beihang University
Beijing, China

Zifan Li
zifan@buaa.edu.cn
Beihang University
Beijing, China

Jianwei Liu
liujianwei@buaa.edu.cn
Beihang University
Beijing, China

Yudi Yang
yangyudi@buaa.edu.cn
Beihang University
Beijing, China

Yuan Lu
lu_yuan@a-star.edu.sg
A*STAR IAIC
Singapore, Singapore

Song Bian
sbian@buaa.edu.cn
Beihang University
Beijing, China

Zhenyu Guan
guanzhenyu@buaa.edu.cn
Beihang University
Beijing, China

Tianwei Zhang
tianwei.zhang@ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Abstract

Asynchronous distributed key generation (ADKG) bootstraps threshold cryptosystems by robustly generating threshold public and private key pairs. This primitive is a fundamental building block for distributed algorithms such as random beacons, threshold signatures, threshold decryption, and general distributed key management services (DKM). While significant progress has been made, ADKG constructions still exhibit computation inefficiency, especially in high-threshold reconstruction, due to their tight coupling to homomorphic-commitment-based verifiable secret sharing.

We fill in this research gap by presenting Kairos, a concretely and asymptotically efficient ADKG protocol. For efficiency, Kairos avoids homomorphic commitments by deriving dLog-based public-private key sharings from the lightweight hash-based asynchronous complete secret sharing (ACSS), with an asymptotically optimal, quadratic per-key communication. For secrecy, Kairos tolerates up to f malicious nodes among $n = 3f + 1$ parties, and produces unbiased public key shares with a high reconstruction threshold of $t \geq f$. The core techniques in Kairos are a non-interactive threshold lifting for private key shares, and a lightweight high-threshold verification script aggregator for public key shares under a lock-then-verify paradigm, which may be of independent interest and applied in asynchronous dynamic proactive secret sharing (ADPSS) or multi-party computation. Evaluations conducted on distributed AWS EC2 servers demonstrate that Kairos reduces 63.7-99.4% runtime compared to the baselines DXKR23 (Usenix Security'23) and HARTS (ASIACRYPT'24) under $t = 2f$, achieving a throughput of 3k keys/min at $n = 16$. When applied in ADPSS, our solution reduces 91.0-98.7% runtime compared to LongLive (Usenix Security'23) at $t = 2f$, resharing 2.3k secrets/min across 16-node committees.

CCS Concepts

• **Security and privacy** → **Distributed systems security**.

Keywords

Distributed key generation; Byzantine robustness; Asynchrony

ACM Reference Format:
Xinxin Xing, Yizhong Liu, Zifan Li, Jianwei Liu, Yudi Yang, Yuan Lu, Song Bian, Zhenyu Guan, and Tianwei Zhang. 2026. Kairos: Efficient High-Threshold Asynchronous Distributed Key Generation from Lightweight Cryptography. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3830454.3832655

1 Introduction

Asynchronous distributed key generation (ADKG) is a foundational primitive for threshold fault-tolerance systems. This primitive enables the robust generation of threshold public and private key pairs among mutually distrustful nodes, in practical asynchronous networks. Owing to its robustness and functionalities, ADKG is widely deployed in distributed key management (DKM) infrastructures to support authentication like Web3Auth [60] in MetaMask or wallet custody like Lit Protocol [6]. The generated keys also underpin various cryptographic primitives, including threshold signatures [38], threshold decryption [26], and random beacons [16].

Current discrete-logarithm-based (dLog-based) ADKG [2–5, 8, 22, 24, 29, 32, 36, 41, 52, 68] has made significant steps toward efficiently bootstrapping robust threshold cryptosystems. As demonstrated in Table 1, recent advances achieve the optimal quadratic or near-quadratic communication [2, 5, 8, 29, 32] with respect to the size of the committee. Nevertheless, production DKM systems and threshold Schnorr signatures [41] impose stricter computational

*Corresponding author. Also with Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beijing, China.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

CCS '26, The Hague, Netherlands.

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832655

Table 1: Comparison of relevant dLog-based asynchronous distributed key generation schemes

Scheme	Security		Computation Efficiency	Communication Complexity†				Setup	Implementation
	full secrecy♮	*threshold*	*support hash-based ACSS?*	*amortized*	*in total*	*round*	*batched*	*no CRS?*	
KMS20 [52]	○	$f \le t < n-f$	○	$O(\kappa n^4)$	$O(\kappa n^4)$	$O(n)$	○	○	○
ZDC+23 [68]	●	$f \le t < n-f$	○	$O(\kappa n^4)$	$O(\kappa n^4)$	$O(1)$	○	●	●
AJM+21 [4]	●	$f \le t < n-f$	○	$O(\kappa n^3 \log n)$	$O(\kappa n^3 \log n)$	$O(1)$	○	●	○
Bingo [3]	●	$f \le t < n-f$	○	$O(\kappa n^3)$	$O(\kappa n^3)$	$O(1)$	○	○	○
DYX+22[1] [24]	●	$t = f$	○	$O(\kappa n^3)$	$O(\kappa n^3)$	$O(\log n)$	○	●	●
DYX+22[2] [24]	●	$f \le t < n-f$	○	$O(\kappa n^3)$	$O(\kappa n^3)$	$O(\log n)$	○	●	●
DXKR23 [22]	●	$f \le t < n-f$	○	$O(\kappa n^3)$	$O(\kappa n^3)$	$O(\log n)$	○	●	●
GLL+22 [36]	●	$f \le t < n-f$	○	$O(\kappa n^3)$	$O(\kappa n^3)$	$O(1)$	○	○	○
[32]+[29]	●	$f \le t < n-f$	○	$O(\kappa c n^2)$	$O(\kappa c n^2)$	$O(1)$	○	○	○
ABLS25 [2]	●	$f \le t < n-f$	○	$O(\kappa^2 n^{2+1/k})$	$O(\kappa^2 n^{2+1/k})$	$O(k)$	○	●	○
Haven++ [5]	●	$f \le t < n-f$	○	$O(\kappa n^2 \log n)$	$O(\kappa n^2 \log n(b+n))$	$O(1)$	●	●	○
HARTS [8]	●	$f \le t < n-f$	○	$O(\kappa n^2 \log n)$	$O(\kappa n^2 \log n(b+n))$	$O(1)$	●	●	● (reproduced)
GS24 [41]	○	$t = f$	●	$O(\kappa n^2)$	$O(\kappa n^2 b + \kappa n^4 \log n)$	$O(1)$	●	●	● (reproduced)
Kairos	●	$f \le t < n-f$	●	$O(\kappa n^2)$	$O(\kappa n^2 b + \kappa n^4 \log n)$	$O(1)$	●	●	●

† **Communication Complexity** - Communication cost for all messages exchanged among honest nodes in the worst-case bound. It is presented as: (i) *total complexity*: overall communication for the whole batch, (ii) *amortized complexity*: amortized communication per key pair. Note that we denote n as the committee size, b the batch size, and κ the security parameter. For schemes [2, 29, 32], c and k are static parameters, with c denoting the size of the reference committee, and k denoting the depth of the recursion tree. For GS24 [41], the reported $O(\kappa bn^2 + \kappa n^4 \log n)$ corresponds to its worst-case instantiation with SS24 [58]. When no sharing instance enters recovery and only the overall public keys are required, GS24 achieves $O(\kappa bn + \kappa n^3 \log n)$ communication. LT-Kairos can be extended to attain the same asymptotic complexity by adapting the techniques of [61].
♮ **Full Secrecy** - Full secrecy is defined in the simulation-based model as in Gennaro et al. [38], requiring that the output public keys are indistinguishable from the ideal ones sampled from the key space. It enables universal key applicability across threshold cryptosystems, especially for general key management services and threshold decryption. In contrast, KMS20 [52] and GS24 [41] realize weaker notions that admit adversarial bias, which are tailored to specific applications such as threshold Schnorr signature [12].
‡ ● support; ○ not support.

requirements on ADKG [12, 37] beyond the conventional one-time initialization settings. Specifically, DKM platforms continuously provision fresh threshold key pairs for wallet onboarding, while threshold Schnorr signing consumes one pre-generated pair in each signature, consisting of fresh shared randomness and the corresponding dLog-based group element. Such workloads naturally admit batching: these materials can be generated offline during low-utilization periods and later consumed by online requests.

To reduce amortized overhead, the notion of batched ADKG [5, 8, 41] was recently proposed, which enables the distributed generation of multiple key pairs in a single instance. Two batched ADKG algorithms, HARTS [8] and Haven++ [5], are designed with an amortized communication of $O(\kappa n^2 \log n)$, where n is the number of participating nodes and κ is the security parameter. Despite the amortization, almost all existing designs still incur prohibitive computational cost due to their tight coupling to *homomorphic-commitment-based* verifiable secret sharing. In practice, the state-of-the-art implementation DXKR23 [22] takes 30 minutes of computation to produce 0.5k key pairs within a 28-node committee, making it impractical for service-level threshold key pairs delivery.

GS24 [41] pioneered a lightweight route to high-throughput threshold Schnorr preprocessing by combining hash-based asynchronous complete secret sharing (ACSS) with group-oriented verification. GS24 realizes an application-specific biased-key-generation functionality, with a reconstruction threshold of $t = f$, where f denotes the malicious node tolerance and $n = 3f + 1$. General-purpose ADKG and DKM services, however, typically require a different interface: uniformly distributed keys under the standard full-secrecy definition [38], the complete vector of public-key shares, and, in many deployments, a high reconstruction threshold $t > f$ to tolerate additional share exposure beyond the adversarial bound. Despite offering roughly two orders of magnitude lower computational cost, hash-based ACSS lacks the homomorphic structure traditionally used to derive and certify high-threshold shares. This challenge becomes particularly acute when simultaneously requiring unbiased key generation and output-optimal quadratic amortized communication, leaving efficient high-threshold ADKG from lightweight cryptography unresolved. This research gap brings a burning question:

> *Can we design an efficient high-threshold ADKG that derives unbiased threshold public-private key pairs from hash-based ACSS with the optimal amortized communication of $O(\kappa n^2)$?*

1.1 Our Results

We answer the above question affirmatively by introducing a new dLog-based ADKG construction with both asymptotic and concrete efficiency. Our solution is developed in two deliberate stages. First, in LT-Kairos, we derive low-threshold sharings of unbiased public and private key pairs from hash-based ACSS. Second, in Kairos, we design a high-threshold ADKG that builds atop LT-Kairos by proposing a new threshold lifting approach.

- **LT-Kairos: an unbiased ADKG compiler for hash-based ACSS.** Our design introduces two core mechanisms to avoid homomorphic commitments while providing unbiased key generation and optimal quadratic communication: (1) A *verifiable sharing locking* paradigm that first strategically fixes private key sharings from the hash-based ACSS [58, 61], then locks the broadcast public key shares from $n - f$ nodes; and (2) a *lightweight verification script aggregator*, which decouples the homomorphic commitments by producing a lightweight correctness proof for verifying the locked public key shares. In these mechanisms, we bound the broadcast message size to the batch size for optimal communication, and fix unbiased private keys before giving public key shares for full secrecy [38]. Consequently, LT-Kairos provides a new ADKG construction from the hash-based ACSS with high performance and general downstream applicability.

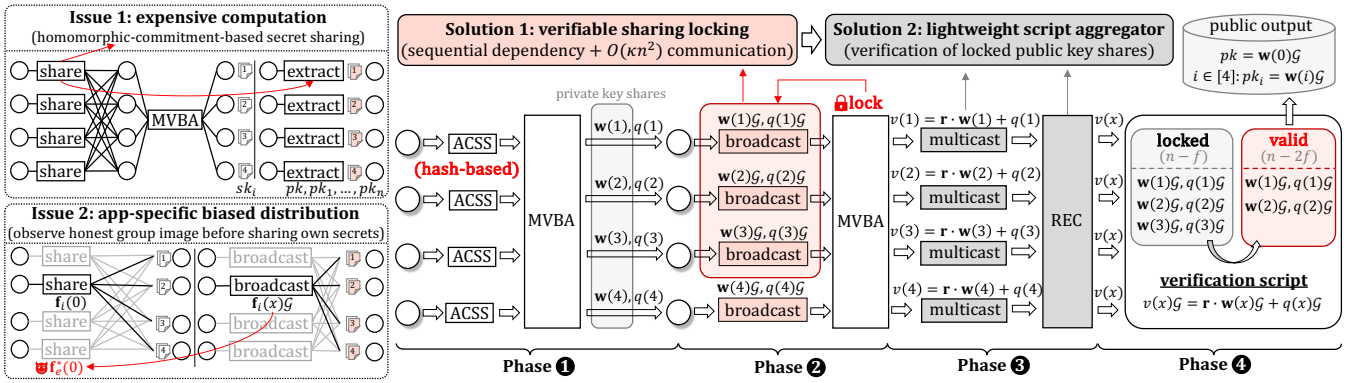


Figure 1: Technical overview of LT-Kairos with committee size $n = 4$, threshold $t = 1$, and batch size b for simplicity.

• **Kairos: an efficient high-threshold ADKG protocol with amortized quadratic communication.** Extending from LT-Kairos, our high-threshold ADKG introduces two new techniques: (1) A *non-interactive threshold lifting technique* (*sk*-NTL) for private key shares generation, which leverages the multiplication homomorphic property of Shamir's sharings, and locally transforms (n, f)-sharings to an (n, t)-sharing with threshold $t \geq f$. (2) A *lightweight high-threshold verification script aggregator* (*pk*-HVS) for high-threshold public key shares verification. This aggregator, for the first time, eliminates high-threshold ADKG's structural reliance on the homomorphic-commitment-based ACSS. Specifically, it propagates the linear relation in *sk*-NTL to aggregate low-degree polynomials into a lightweight verification script for high-threshold public key shares, addressing the challenge of high-degree polynomial evaluation verification from lightweight cryptography. Consequently, Kairos provides the first high-threshold ADKG from hash-based ACSS, while providing optimal amortized communication, high concrete efficiency, and full secrecy for general applicability.

• **Diverse applications for Kairos.** We demonstrate various implications of our efficient ADKG design. First, by extending our *sk*-NTL and *pk*-HVS techniques to the dual-committee setting, we design the first high-threshold asynchronous dynamic proactive secret sharing (ADPSS) from hash-based ACSS, with an amortized communication of $O(\kappa n^2)$. Evaluation results demonstrate that it reduces 91.0-98.7% runtime and more than 74.7% bandwidth usage compared to the state-of-the-art high-threshold ADPSS, LongLive [67]. Second, distributed key management infrastructures, such as Web3Auth [60] in MetaMask, generate a large amount of threshold key pairs through high-threshold, generally secure ADKG. Kairos breaks the prior performance bottlenecks and delivers sustained batch-processing efficiency with low per-key costs, making it a practical foundation for production deployments.

• **Implementation and distributed deployment.** We implement LT-Kairos, Kairos, GS24 [41], and HARTS [8]; reproduce DYX+22 [24], DXKR23 [22], and LongLive [67]; and conduct evaluations on Amazon EC2 instances evenly distributed across 4 regions, with one node per instance. The results demonstrate that Kairos consistently outperforms all baselines in both runtime and bandwidth usage. In the low-threshold setting, LT-Kairos takes 235.8s of computation to produce 30k sharings of unbiased key pairs at $n = 16$, compared with 2267.1s in DYX+22 [24]. In the high-threshold setting, Kairos produces 3k key pairs in 59.4s within a 16-node committee, compared with 479.2s in DXKR23 [22] and 4586.9s in HARTS [8]. It reduces up to 99.4% of the runtime and 89.4% of the bandwidth across all the evaluated configurations.

2 Technical Overview

As a warm-up, we outline the standard ADKG framework. In the *private key construction* phase, almost every node i secret-shares random values through b polynomials $\mathbf{f}_i(x)$ of t-degree, distributing $\mathbf{f}_i(j)$ to every node j. A consensus protocol then selects a qualified set T of valid sharing instances, and the (n, t)-threshold private key share $\mathbf{w}(i)$ at each node i is then extracted from $\{\mathbf{f}_j(i)\}_{j\in\mathsf{T}}$ through standard randomness extraction [22]. In the *public key derivation* phase, the nodes agree on the public keys $\mathbf{w}(0)\mathcal{G}$ and all public key shares $\{\mathbf{w}(i)\mathcal{G}\}_{i\in[n]}$ as output, where $\mathcal{G}$ is a group generator.

2.1 LT-Kairos: Low-Threshold ADKG with $t = f$

2.1.1 Motivation. State-of-the-art ADKG schemes [2, 5, 8, 29, 32] derive the public outputs directly from homomorphic commitments embedded in the underlying verifiable secret sharing (Figure 1, Issue 1). Although this approach achieves near-quadratic communication, homomorphic-commitment-based sharing is computationally expensive, and the cost becomes prohibitive when scaling to large batches as required by production key management [6, 60] and threshold Schnorr signatures [12]. An alternative approach [41], avoids homomorphic commitments by having each dealer i broadcast group elements $\mathbf{f}_i(x)\mathcal{G}$ (or simply $\mathbf{f}_i(0)\mathcal{G}$ as in its native protocol when public key shares are not produced) alongside the lightweight hash-based verifiable secret sharing. However, because the adversary may observe the group images contributed by honest dealers before fixing its own randomness, the resulting keys may be adversarially biased. This biased key-generation functionality is sufficient for the threshold Schnorr preprocessing considered in [41], but does not satisfy the standard full-secrecy requirement [38] of general-purpose ADKG (Figure 1, Issue 2).

2.1.2 Our Approach. LT-Kairos resolves this tension by enabling *lightweight* public key verification *after* the private key shares are irrevocably fixed. We realize this through a new verifiable sharing locking paradigm and a lightweight verification script aggregator.

Our verification is inspired by a classical technique. Given a batch of f-degree polynomials $\mathbf{w}(x)$, an f-degree blinding polynomial $q(x)$, and a challenge vector $\mathbf{r}$, consider the linear combination $v(x) = \mathbf{r} \cdot \mathbf{w}(x) + q(x)$. Once a candidate tuple is fixed, any deviation in its public-key evaluation induces a nontrivial linear equation in the uniformly sampled challenge vector $\mathbf{r}$, which holds with probability at most $1/p$. This lightweight tool reduces batched evaluation verification into a simple linear check.

Verifiable sharing locking paradigm. Random linear combination enables us to verify public key shares $\mathbf{w}(i)\mathcal{G}$ via the lightweight verification script $v(x)$, instead of the expensive homomorphic commitments. However, two conditions must be met: (1) the challenge $\mathbf{r}$ remains unpredictable until the public key shares are irrevocably committed; and (2) $v(x)$ is consistently available to all nodes. Given our goals and these constraints, the paradigm proceeds as follows.

- *Phase 1: lock private key sharings.* The committee executes the private key construction phase following the standard ADKG framework, but uses hash-based ACSS instead. Upon completion, each node i holds its (n, t)-threshold private key share $\mathbf{w}(i)$ and a blinding share $q(i)$ with $t = f$. Guaranteed by the secrecy of ACSS, no additional information about the private keys is revealed. Thus, if the public keys are deterministically mapped from private keys, the adversary cannot bias the key distribution, ensuring full secrecy.
- *Phase 2: lock public key sharings.* Each node i reliably broadcasts its public key shares $\mathbf{w}(i)\mathcal{G}$ along with a blinding term $q(i)\mathcal{G}$. To ensure the unpredictability of the challenge $\mathbf{r}$, one can simply wait for all broadcasts to complete before sampling $\mathbf{r}$ in synchronous networks. In asynchronous networks with unbounded delays, however, this creates a dilemma: waiting for all broadcasts stalls progress, while revealing $\mathbf{r}$ after a partial set allows the adversary to adaptively craft shares that pass verification. We resolve this through the following steps. The nodes invoke consensus to agree on a *locked set* T of public key shares from $n - f$ delivered broadcasts. Once T is irrevocably fixed, the nodes open a hidden random seed from phase 1 and derive the challenge $\mathbf{r}$ by hashing the seed with T, where T binds all public key shares to be verified, while the seed keeps $\mathbf{r}$ unpredictable. Further verification is restricted to the locked set, preserving soundness. Crucially, each node broadcasts only $O(\kappa b)$ bits for batch size b, yielding $O(\kappa n^2)$ amortized communication.

Lightweight verification script aggregator. With $\mathbf{r}$ determined, the nodes construct the verification script $v(x)$ through this aggregator, and filter valid public key shares from the locked set T.

- *Phase 3 and Phase 4.* Each node i, holding $\mathbf{w}(i)$ and $q(i)$, computes and multicasts its evaluation $v(i) = \mathbf{r} \cdot \mathbf{w}(i) + q(i)$ on $v(x)$. The receivers apply online error-correction (detailed in Section 3.1), a robust interpolation algorithm that tolerates up to f Byzantine senders, to recover the unique f-degree polynomial $v(x)$ consistent with $n - f$ correct evaluations. Each locked tuple $\mathbf{w}(\tau)\mathcal{G}$ and $q(\tau)\mathcal{G}$ is then validated by checking whether $\mathbf{r} \cdot \mathbf{w}(\tau)\mathcal{G} + q(\tau)\mathcal{G}$ equals $v(\tau)\mathcal{G}$. Since at most f-out-of-$(n - f)$ locked public key shares are adversarial, the survived $n - 2f \geq f + 1$ valid shares are sufficient to interpolate the public outputs, i.e., $\mathbf{w}(0)\mathcal{G}$ and $\{\mathbf{w}(j)\mathcal{G}\}_{j\in[n]}$.

2.2 Kairos: High-Threshold ADKG with $t \geq f$

2.2.1 Motivation. LT-Kairos produces key shares with the reconstruction threshold $t = f$. Many applications, including consensus protocols [42, 64] and distributed key management systems [6, 60], require a *higher* threshold $t \geq f$ to tolerate additional leakage of shares beyond f corrupted nodes. Existing high-threshold ADKG protocols [2–5, 8, 22, 24, 29, 32, 36, 52, 68] address this by directly employing high-threshold verifiable secret sharing built on homomorphic commitments, which is typically more expensive than its low-threshold alternatives. DXKR23 [22] takes a more efficient approach: it runs low-threshold ACSS and *transforms* the resulting low-threshold shares to high-threshold *private* key shares via an *interactive* protocol called distributed polynomial sampling. The high-threshold *public* key shares are then derived from the homomorphic commitments embedded in ACSS through zero-knowledge proofs. While faster than prior work, the reliance on homomorphic-commitment-based ACSS remains a performance bottleneck.

2.2.2 Our Approach. To break this limitation, we propose the first high-threshold ADKG that accommodates lightweight hash-based ACSS. First, we develop a new *non-interactive* threshold lifting method, *sk*-NTL, which transforms low-threshold Shamir shares into high-threshold *private* key shares through local computation. Second, we introduce a *lightweight* high-threshold verification script aggregator, *pk*-HVS, that derives dLog-based high-threshold *public* key shares from general low-threshold Shamir's sharings, entirely avoiding the reliance on homomorphic commitments.

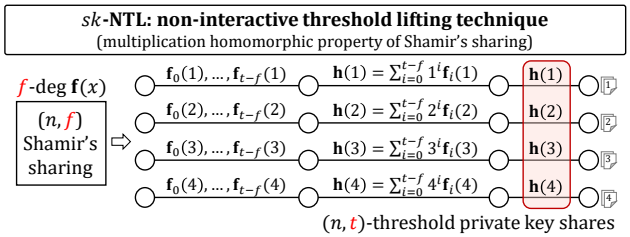


Figure 2: Illustration of *sk*-NTL with $n = 4, f = 1, t = 2$

***sk*-NTL: non-interactive threshold lifting.** Let f denote the malicious node tolerance and $t \in [f, n-f)$ the desired reconstruction threshold. As illustrated in Figure 2, our *sk*-NTL technique transforms (n, f)-Shamir shares into (n, t)-Shamir shares *locally*, which serves as the high-threshold private key shares.

An attempt. A natural idea exploits the multiplicative homomorphism of Shamir sharing: suppose that there exists a batch of f-degree polynomials $\mathbf{f}(x)$, define $\mathbf{h}(x) = x^{t-f} \cdot \mathbf{f}(x)$, then $\mathbf{h}(x)$ is of t-degree as desired. Thus, given the low-threshold shares $\mathbf{f}(i)$, each node can locally compute the high-threshold share $\mathbf{h}(i) = i^{t-f} \cdot \mathbf{f}(i)$. However, this construction *violates secrecy* in high-threshold setting: since each t-degree polynomial in $\mathbf{h}(x)$ has only $f+1$ unknown coefficients from those of $\mathbf{f}(x)$, any $f+1$ shares suffice to recover the secret, which is fewer than the intended $t+1$.

Our construction. The fix is to introduce independent randomness at each degree level. Suppose that the nodes hold shares of $t-f+1$ independent vectors of f-degree polynomials $\mathbf{f}_0(x), \mathbf{f}_1(x), \ldots, \mathbf{f}_{t-f}(x)$. We define the t-degree polynomials $\mathbf{h}(x)$ as follows:

$$\mathbf{h}(x) = \mathbf{f}_0(x) + x \cdot \mathbf{f}_1(x) + \cdots + x^{t-f} \cdot \mathbf{f}_{t-f}(x) \tag{1}$$

Consequently, each node i, holding the (n, f)-threshold shares $\mathbf{f}_0(i), \ldots, \mathbf{f}_{t-f}(i)$, calculates the (n, t)-threshold shares $\mathbf{h}(i)$ under the same linear relation. An adversary corrupting f nodes learns

$\mathbf{f}_k(i)$ for all $k \in [0, t-f]$ and all corrupted i, retaining one degree of freedom on each $\mathbf{f}_k(x)$. In high-threshold reconstruction, the adversary may additionally obtain up to $t-f$ high-threshold shares $\mathbf{h}(j)$ from honest nodes before threshold t is breached. Secrecy follows the high-level idea that each $\mathbf{h}(j)$ provides only *one* linear constraint on polynomials $\{\mathbf{f}_k(x)\}_{k\in[0,t-f]}$. With only $t-f$ such constraints, the adversary cannot solve for all $t-f+1$ degrees of freedom, and the secrets $\mathbf{h}(0) = \mathbf{f}_0(0)$ remain hidden.

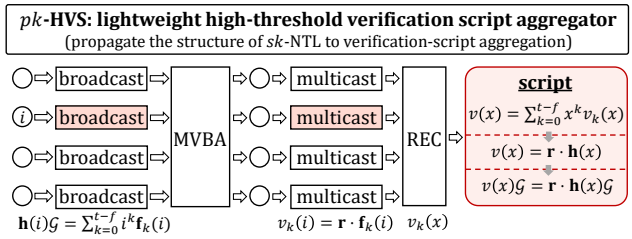


Figure 3: Illustration of pk-HVS with $n = 4, f = 1, t = 2$

***pk*-HVS: high-threshold lightweight verification script aggregator.** With private key shares lifted to the threshold t, we further produce public outputs through pk-HVS, as illustrated in Figure 3.

Attempt 1: transform low-threshold public key shares into high-threshold. One might have each node broadcast all low-threshold group elements $\{\mathbf{f}_k(i)\mathcal{G}\}_{k\in[0,t-f]}$ with each $|\mathbf{f}_k(i)| = b$. These elements can be verified using the same technique in LT-Kairos, and then transformed into high-threshold public key shares $\mathbf{h}(i)\mathcal{G}$ via Equation (1). While correct, each broadcast carries $O(bn)$ group elements, leading to $O(\kappa n^3)$ amortized communication per key.

Attempt 2: broadcast high-threshold public key shares directly. For $O(\kappa n^2)$ amortized communication, each node broadcasts its high-threshold public key shares $\mathbf{h}(i)\mathcal{G}$ of size $O(\kappa b)$. Naturally the next step is to reconstruct a verification script $v(x) = \mathbf{r}\cdot\mathbf{h}(x)$ through online error-correction and check each locked $\mathbf{h}(i)\mathcal{G}$ against it, where the last entry in $\mathbf{h}(x)$ is the blinding polynomial for randomizing $v(x)$ and is excluded from the key construction. The obstacle is that $v(x)$ has degree $t>f$, and online error-correction that tolerates f errors uniquely recovers a t-degree polynomial only if $t+2f+1$ evaluations are available. Since $t+2f+1 > 3f+1 = n$, one *cannot* collect enough evaluations to reconstruct the t-degree script $v(x)$.

Our solution. The key insight is that the structure of Equation (1) propagates to the aggregation of the verification script. Specifically:

$$v(x) = \mathbf{r}\cdot\mathbf{h}(x) = \underbrace{\mathbf{r}\cdot\mathbf{f}_0(x)}_{v_0(x)} + x\cdot\underbrace{\mathbf{r}\cdot\mathbf{f}_1(x)}_{v_1(x)} + \cdots + x^{t-f}\cdot\underbrace{\mathbf{r}\cdot\mathbf{f}_{t-f}(x)}_{v_{t-f}(x)} \quad (2)$$

Each f-degree component $v_k(x) = \mathbf{r}\cdot\mathbf{f}_k(x)$ *can* be reconstructed via online error-correction. Thus, we let each node i calculate and multicast $v_0(i), v_1(i), \ldots, v_{t-f}(i)$, such that the receivers reconstruct each f-degree polynomial $v_k(x)$ and compute the t-degree script as $v(x) = v_0(x) + x\cdot v_1(x) + \cdots + x^{t-f}\cdot v_{t-f}(x)$. With $v(x)$ in hand, the nodes verify each locked public key share $\mathbf{h}(\tau)\mathcal{G}$ by checking whether $\mathbf{r}\cdot\mathbf{h}(\tau)\mathcal{G}$ equals $v(\tau)\mathcal{G}$. After collecting enough valid shares, the public outputs of $\mathbf{h}(0)\mathcal{G}$ and $\{\mathbf{h}(j)\mathcal{G}\}_{j\in[n]}$ are interpolated, which completes the protocol.

3 Preliminaries and Problem Formulation

Notations. Let $\mathbb{Z}_p$ denote a finite field of large prime order p, $\mathbb{G}$ denote a cyclic group of order p with hard discrete logarithm, and $\mathcal{G}$ denote a generator of $\mathbb{G}$. We use additive notation for the group operation of $\mathbb{G}$, and use κ as the bit length of the cryptographic security parameter. Let b denote the batch size, n denote the number of participating nodes, f denote the malicious node tolerance, and t denote the reconstruction threshold. We use $:=$ for deterministic assignment, $\leftarrow$ for probabilistic assignment, and $\|$ for vector concatenation. Bold symbols such as $\mathbf{a}$ denote vectors or matrices. The element-wise threshold secret sharing of $\mathbf{a}$ is written as $[\mathbf{a}]$, with $[\mathbf{a}]_i$ being the share for node $\mathcal{P}_i$. For two m-dimension vectors $\mathbf{a}$ and $\mathbf{b}$, we define their inner-product as $\mathbf{a}\cdot\mathbf{b} = \sum_{i=1}^{m} a_i b_i$, and their entry-wise addition as $\mathbf{a}+\mathbf{b} = (a_1+b_1, \ldots, a_m+b_m)$. Let $\mathbf{f}(x)$ denote a vector of polynomials over $\mathbb{Z}_p[x]$. $\mathbf{f}(i)$ consists of the evaluations at $x = i$ for all polynomials in $\mathbf{f}(x)$. We say that $(i, \mathbf{a})$ lies on $\mathbf{f}(x)$ only if $\mathbf{a} = \mathbf{f}(i)$. For integers $a \le b$, let $[a, b] := \{a, a+1, \ldots, b\}$ and $[a, b) := \{a, a+1, \ldots, b-1\}$. We further abbreviate $[n] := [1, n]$ for the number of nodes n.

Adversary model. We assume the adversary $\mathcal{A}$ is a probabilistic polynomial time (P.P.T) Byzantine adversary who *statically* corrupts nodes within a committee of n nodes with an upper limit f, where $n = 3f + 1$. If a node is corrupted, $\mathcal{A}$ gains full control over it, reads all its inputs, and sets the messages to be sent.

Network model. Communication occurs over an asynchronous network of authenticated pairwise point-to-point channels. The adversary can arbitrarily delay messages or alter their order. Messages exchanged between honest nodes will eventually be delivered.

3.1 Building Blocks

Asynchronous complete secret sharing (ACSS). An (n, f, b)-ACSS protocol enables a dealer to share a batch of b secrets $\mathbf{s} \in \mathbb{Z}_p^b$ among n nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where up to f nodes are corrupted by an adversary. In a successful execution, every $\mathcal{P}_i$ receives a Shamir's share $[\mathbf{s}]_i$. The secret vector $\mathbf{s}$ can be interpolated from $f+1$ valid shares, and any subset of at most f shares reveals no information about $\mathbf{s}$. An (n, f, b)-ACSS protocol satisfies:

- **Correctness**: If the dealer is honest and inputs the secret vector $\mathbf{s}$, there exists b polynomials $\mathbf{f}(x) \in \mathbb{Z}_p[x]$ of f-degree, such that $\mathbf{f}(0) = \mathbf{s}$ and every honest $\mathcal{P}_i$ outputs the share $[\mathbf{s}]_i = \mathbf{f}(i)$.
- **Completeness**: If any honest node produces an output, there must exist a batch of b polynomials $\tilde{\mathbf{f}}(x) \in \mathbb{Z}_p[x]$ of degree up to f, such that every honest node $\mathcal{P}_i$ outputs $\tilde{\mathbf{f}}(i)$.
- **Secrecy**: If the dealer is honest, the protocol reveals no information about $\mathbf{s}$ to the adversary.

Homomorphic-commitment-based ACSS [8, 66] certifies shares via homomorphic polynomial commitments (e.g., Pedersen [54]), whereas hash-based ACSS relies on lightweight tools (e.g., collision-resistant hashing, erasure coding, and symmetric encryptions), which cannot directly produce homomorphic validity proofs. Representative hash-based ACSS schemes include SS24 [58] and GoSSamer [61]. Both achieve linear per-secret communication in the optimistic case. When recovery is triggered by provable dealer misbehavior, SS24 may incur quadratic per-secret communication, whereas GoSSamer preserves linear per-secret communication. However, GoSSamer by itself is only a hash-based ACSS and does

not provide the dLog-based algebraic structure needed by ADKG. After the ACSS, consensus, and extraction framework, the parties obtain only low-threshold private key sharings $[s]$. The main ADKG challenge is then how to derive the corresponding public keys and (high-threshold) public key sharings $[s]\mathcal{G}$ efficiently and without bias, despite the lack of homomorphic commitments. (LT-)Kairos resolves this gap and enables an unbiased ADKG with output-optimal amortized communication.

Online error-correction. Consider $n = 3f+1$ evaluations of an unknown univariate f-degree polynomial, of which up to f may be adversarially withheld or corrupted. In asynchronous settings where evaluations arrive sequentially, online error-correction [11] uniquely reconstructs the polynomial under this error bound, and is widely used in distributed protocols [22, 41, 58].

Overview. The routine proceeds in up to $f+1$ iterations. Given an accumulating set Q that contains code-words of the Reed-Solomon code, in the r-th iteration, the algorithm waits for $|Q| = 2f + r$ and decodes the fragments in set Q to obtain $\mathcal{L}$. Once obtaining the message $\mathcal{L}$, we encode it using Reed-Solomon code for the fragment set $\hat{Q}$. If there are at least $2f + 1$ same fragments in Q and $\hat{Q}$, online error-correction successfully outputs the decoded message $\mathcal{L}$. Since eventual delivery in asynchronous networks ensures that $n-f$ valid fragments from honest nodes will eventually deliver, the algorithm is guaranteed to terminate in $f+1$ iterations. In Reed-Solomon codes, the message vector is in one-to-one correspondence with an f-degree polynomial via the fixed encoding, so decoding $\mathcal{L}$ is exactly the same as reconstructing the designated polynomial.

Reliable broadcast. A reliable broadcast protocol ensures that a message m distributed by a dealer $\mathcal{P}$ is consistently received by all the n participants. We adopt the construction from DXR21 [23] with a communication complexity of $O(n|m|+\kappa n^2)$. Reliable broadcast is widely used in asynchronous protocols [24, 41, 58] and satisfies the following properties if at most f participating nodes are corrupted:

- **Correctness**: All honest nodes that output a message output the same one. If the dealer $\mathcal{P}$ is honest, the output message is the original input of the dealer $\mathcal{P}$.
- **Completeness**: If any honest participating node outputs a message, or if an honest dealer inputs a message, all honest participating nodes will output a message eventually.

Multi-valued validated asynchronous byzantine agreement (MVBA). MVBA is an asynchronous Byzantine fault-tolerant consensus protocol, initially proposed in CKPS01 [15]. In MVBA, each participating node proposes an input value; the protocol ensures agreement on one of these values that satisfies a predefined common predicate. MVBA is widely used in ADKG constructions [8, 22] and satisfies the following properties:

- **Agreement**: All honest nodes output the same value.
- **Termination**: If all honest nodes propose inputs that satisfy the predicate, then all honest nodes eventually output.
- **External Validity**: If an honest node outputs a value, then the output satisfies the predicate.

3.2 ADKG Definition

Asynchronous distributed key generation. An (n, t, f, b)-ADKG protocol enables n participating nodes $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to distributively generate a batch of b dLog-based threshold key pairs, where up to f nodes are corrupted. Upon terminating the protocol, each $\mathcal{P}_i$ outputs private key shares $[\mathbf{e}]_i$, public keys $\mathbf{e}\mathcal{G}$ and public key shares $([\mathbf{e}]_1\mathcal{G}, \ldots, [\mathbf{e}]_n\mathcal{G})$, where $[\mathbf{e}]$ represents an (n, t)-Shamir sharing [56] of $\mathbf{e}$ with $|\mathbf{e}| = b$ and $\mathcal{G}$ represents a generator in group $\mathbb{G}$. The vector of private keys $\mathbf{e}$ remains secret until $t + 1$ valid shares $[\mathbf{e}]$ are exposed. The protocol is *high-threshold* if it supports any reconstruction threshold $t \in [f, n-f)$. In batched settings, we assume that $b = \text{poly}(n)$, as considered in prior work [41, 58, 61]. An (n, t, f, b)-ADKG protocol satisfies the following properties:

- **Termination**: Every honest node outputs the private key shares $[\mathbf{e}]_i$, public keys $\mathbf{e}\mathcal{G}$, and public key shares $([\mathbf{e}]_1\mathcal{G}, \ldots, [\mathbf{e}]_n\mathcal{G})$.
- **Correctness**: (i) All honest nodes output the same public keys $\mathbf{e}\mathcal{G}$ and public key shares $([\mathbf{e}]_1\mathcal{G}, \ldots, [\mathbf{e}]_n\mathcal{G})$; (ii) the private key shares $[\mathbf{e}]_i$ for node $\mathcal{P}_i$ are consistent with corresponding public key shares $[\mathbf{e}]_i\mathcal{G}$; (iii) there exist b polynomials $\mathbf{f}(x) \in \mathbb{Z}_p[x]$ of t-degree, such that $\mathbf{e} = \mathbf{f}(0)$ and $[\mathbf{e}]_i = \mathbf{f}(i)$ for all $i \in [n]$.
- **Full Secrecy**: No computationally-bounded adversary can (i) learn information about $\mathbf{e}$ beyond what is given by the public keys $\mathbf{e}\mathcal{G}$; and (ii) prevent the public keys $\mathbf{e}\mathcal{G}$ from being uniformly sampled over $\mathbb{G}$. Formally, for any P.P.T adversary $\mathcal{A}$ that corrupts up to f nodes, there exists a P.P.T simulator that on input the uniformly sampled $\mathbf{e}\mathcal{G}$, can produce a simulated view view_{sim}, such that $(\text{view}_{\text{sim}}, \mathbf{e}\mathcal{G})$ is computationally indistinguishable from $(\text{view}_{\text{real}}, \mathbf{pk})$, where $\text{view}_{\text{real}}$ is the view of $\mathcal{A}$ in the real execution with $\mathbf{pk}$ as the output public keys.

4 LT-Kairos: Low-Threshold ADKG Protocol

LT-Kairos is a low-threshold ADKG protocol that simultaneously generates a batch of threshold public-private key pairs within a committee. Its public input includes the finite field $\mathbb{Z}_p$ of prime order p, the number of participating nodes n, the resilience of malicious nodes f, the reconstruction threshold t with $t = f$, the output batch size b, and a generator $\mathcal{G}$ of a group $\mathbb{G}$ of order p. Algorithm 1 illustrates the procedure of LT-Kairos.

Initially, the committee nodes collectively produce the private key shares through lightweight randomness sharing (Algorithm 2), which is built atop hash-based ACSS (Phase ❶). The nodes then reliably broadcast their public key shares and lock $n - f$ valid broadcast instances through consensus (Phase ❷). After deriving a challenge vector after the locked set is fixed, each node calculates a share through random linear combination, and multicasts it to produce the verification script (Phase ❸). Through the script, valid public key shares from $f + 1$ nodes are filtered from the locked ones. Finally, all the public outputs are interpolated, including the public keys and all public key shares (Phase ❹).

4.1 Concrete Scheme

Phase ❶: lightweight private key generation. The objective of this phase is to produce private key shares for all nodes.

As illustrated in Algorithm 1, each node invokes the lightweight randomness sharing component (Algorithm 2), to prepare the sharing $[\mathbf{u}]$ of $b+2$ random values $\mathbf{u}$ among the committee, such that every honest node $\mathcal{P}_i$ outputs its share $[\mathbf{u}]_i$. We denote the first b entries in $[\mathbf{u}]_i$ as $[\mathbf{w}]_i$, and the $(b+1)$-th entry as $[q]_i$. Each $\mathcal{P}_i$ sets its private key shares as $\mathbf{sk}_i := [\mathbf{w}]_i$ and public key shares as $\mathbf{pk}_i := [\mathbf{w}]_i\mathcal{G}$. $[q]_i$ serves as a blinding share, and its mapping

Algorithm 1 LT-Kairos: low-threshold batched ADKG

input: $\mathbb{Z}_p$, n, f (malicious node tolerance), $t = f$ (reconstruction threshold), b (output batch size), and $\mathcal{G}$ (a generator of a group $\mathbb{G}$ of order p)
private output: $\mathbf{sk}_i$ (private key shares for each node $\mathcal{P}_i$)
public output: $\mathbf{pk}$ (public keys), and $\{\mathbf{pk}_j\}_{j\in[n]}$ (public key shares)

Phase ❶: lightweight private key generation (as node $\mathcal{P}_i$)
• invoke Algorithm 2, randomness sharing, with batch size $b+2$
upon outputting $[\mathbf{u}]_i$ from Algorithm 2 with $|\mathbf{u}| = b+2$ **do**
  ∘ let the first b entries in $[\mathbf{u}]_i$ be $[\mathbf{w}]_i$, the $(b+1)$-th entry be $[q]_i$
  ∘ private key share $\mathbf{sk}_i := [\mathbf{w}]_i$ and public key share $\mathbf{pk}_i := [\mathbf{w}]_i\mathcal{G}$
  ∘ $[q]_i$ is the blinding share and $e_i := [q]_i\mathcal{G}$ is the public blinding share

Phase ❷: public key shares locking (as node $\mathcal{P}_i$)
• reliable broadcast $(\mathbf{pk}_i, e_i)$ to all nodes
upon receiving $(\mathbf{pk}_j, e_j)$ from node $\mathcal{P}_j$ **do**
  ∘ add the index j into the local accumulating set T_i
  upon $|\mathsf{T}_i| = n - f$ **do** invoke MVBA with input T_i
upon outputting the agreed set T from MVBA **do** // lock for verification
  ∘ gather the locked public key shares $\mathsf{T}_{\mathsf{lock}} := \{(\mathbf{pk}_\tau, e_\tau) : \tau \in \mathsf{T}\}$

Phase ❸: lightweight script generation (as node $\mathcal{P}_i$)
upon constructing the locked set $\mathsf{T}_{\mathsf{lock}}$ **do**
  ∘ multicast the $(b+2)$-th entry in $[\mathbf{u}]_i$, denoted as $[d]_i$, among all nodes
upon receiving $[d]_j$ from node $\mathcal{P}_j$ **do** // robustly reconstruct d
  ∘ add $(j, [d]_j)$ into a local accumulating set $\mathsf{T}_{\mathsf{oec}}$
  upon $|\mathsf{T}_{\mathsf{oec}}| = n-f$ **do** continuously run online error-correction on $\mathsf{T}_{\mathsf{oec}}$
upon obtaining d from online error-correction and constructing $\mathsf{T}_{\mathsf{lock}}$ **do**
  ∘ calculate b-dimension challenge vector $\mathbf{r}$ by hashing $(\mathsf{sid}, d, \mathsf{T}_{\mathsf{lock}})$
  ∘ calculate $v(i) := \mathbf{r} \cdot [\mathbf{w}]_i + [q]_i$ and multicast $v(i)$ among all nodes
  ∘ repeat the online error-correction process and reconstruct $v(x)$

Phase ❹: public keys derivation (as node $\mathcal{P}_i$)
upon outputting $v(x)$ from online error-correction **do**
  for each $\tau \in \mathsf{T}$, **if** $v(\tau)\mathcal{G} = \mathbf{r} \cdot \mathbf{pk}_\tau + e_\tau$ **do** // verify each entry in $\mathsf{T}_{\mathsf{lock}}$
    • add $(\tau, \mathbf{pk}_\tau)$ into set T_{pk} of valid public key shares // $\mathbf{pk}_\tau = [\mathbf{w}]_\tau\mathcal{G}$
  if $|\mathsf{T}_{\mathsf{pk}}| \geq t+1$ **do** // filter $t+1$ valid ones from $n-f$ locked
    • interpolate $t+1$ elements in T_{pk} into $\mathbf{w}\mathcal{G}$ and $\{[\mathbf{w}]_j\mathcal{G}\}_{j\in[n]}$
    • output $\mathbf{pk} := \mathbf{w}\mathcal{G}$, along with $\mathbf{pk}_j := [\mathbf{w}]_j\mathcal{G}$ for all $j \in [n]$

$e_i := [q]_i\mathcal{G}$ in the group serves as the public blinding share. The $(b+2)$-th entry in $[\mathbf{u}]_i$ is later used in challenge vector derivation.

Lightweight randomness sharing. Randomness sharing is a standard subroutine in asynchronous threshold protocols [22, 61, 65, 67]. It robustly generates Shamir's sharings $[\mathbf{u}]$ of m random values $\mathbf{u}$, ensuring that every honest node terminates with its share $[\mathbf{u}]_i$ and no information of $\mathbf{u}$ is leaked, against any P.P.T adversary that corrupts up to 1/3 nodes. This component invokes the ACSS and MVBA protocols in a black-box way. When instantiated with lightweight ACSS [58, 61] and lightweight MVBA [21, 27], we refer to the construction as lightweight randomness sharing. Each node $\mathcal{P}_i$ begins by sampling a small batch of $\hat{m} = \lceil m/(f+1) \rceil$ random values $\mathbf{a}_i$, and initiates an ACSS instance to share the secrets $\mathbf{a}_i$ among the n-node committee. Once node $\mathcal{P}_i$ outputs the share $[\mathbf{a}_u]_i$ from the ACSS instance initiated by $\mathcal{P}_u$, it adds the index u into the local set T_i. To reach a consensus of $n-f$ successful ACSS instances in asynchronous networks, each node submits its accumulating set T_i into MVBA if it reaches the size $n-f$, and waits for MVBA to return the common set T. Following a standard extraction technique [22, 67] (further detailed in Appendix C), every $\mathcal{P}_i$ extracts $f+1$ vectors of random sharings from $n-f$ ACSS sharings, where up to f may be corrupted. Specifically, it multiplies $\hat{m}(n-f)$ shares $\{[\mathbf{a}_u]_i\}_{u\in\mathsf{T}}$ by

Algorithm 2 Lightweight randomness sharing

input: $\mathbb{Z}_p$ (finite field of prime order p), n (the number of participating nodes), f (malicious node tolerance), m (batch size of the generated randomness)
private output: $[\mathbf{u}]_i$ (share of randomness $\mathbf{u}$) for each node $\mathcal{P}_i$

upon initialization **do** // as each node $\mathcal{P}_i$ of all the n nodes
  • sample a vector $\mathbf{a}_i$ of $\hat{m} = \lceil m/(f+1) \rceil$ secrets
  • initiate an ACSS instance to share secrets $\mathbf{a}_i$
upon outputting $[\mathbf{a}_u]_i$ from the ACSS initiated by $\mathcal{P}_u$ **do**
  • add the index u into the local accumulating set T_i
  upon $|\mathsf{T}_i| = n - f$ **do** invoke MVBA with input T_i
upon outputting the agreed set T from MVBA **do**
  • extract $[\mathbf{u}]'_i$ from $\{[\mathbf{a}_u]_i\}_{u\in\mathsf{T}}$ through super-invertible matrix
  • output the first m entries of $[\mathbf{u}]'_i$ as $[\mathbf{u}]_i$

$\hat{m}$ super-invertible matrices of size $(f+1) \times (n-f)$, obtains shares $[\mathbf{u}]'_i$ of $\hat{m}(f+1)$ randomness $\mathbf{u}$, and outputs the first m shares.

Phase ❷: public key shares locking. The objective of this phase is to lock a globally consistent set of public key shares from $n-f$ nodes. The locked set is further used in subsequent validity checks.

Each node $\mathcal{P}_i$ reliably broadcasts the tuple $(\mathbf{pk}_i, e_i)$ within the committee, which contains its public key share and its public blinding share. Upon delivering $(\mathbf{pk}_j, e_j)$ from $\mathcal{P}_j$, each node $\mathcal{P}_i$ adds the index j into its local accumulating set T_i. When T_i reaches the size $n-f$, the node submits it to MVBA. MVBA will output a common set T of size $n-f$, which ensures that, for every τ in T, the corresponding broadcast instance will deliver an identical payload at all honest nodes. Therefore, each node can deterministically assemble the locked set $\mathsf{T}_{\mathsf{lock}} := \{(\mathbf{pk}_\tau, e_\tau) : \tau \in \mathsf{T}\}$ from these payloads, and all honest nodes are guaranteed to construct the same locked set.

Phase ❸: lightweight script generation. The goal of this phase is to generate a common verification script for validating the locked public key shares. After constructing the complete locked set $\mathsf{T}_{\mathsf{lock}}$, each node $\mathcal{P}_i$ multicasts its share $[d]_i$ of the independent random seed d, corresponding to the $(b+2)$-th output of the lightweight randomness-sharing phase. The nodes robustly reconstruct d using online error-correction and derive the b-dimensional challenge vector $\mathbf{r}$ by hashing $(\mathsf{sid}, d, \mathsf{T}_{\mathsf{lock}})$. Since $\mathsf{T}_{\mathsf{lock}}$ is irrevocably fixed before any honest seed share is revealed, the challenge remains unpredictable when the public key shares are chosen. Each node $\mathcal{P}_i$ then computes and multicasts $v(i) := \mathbf{r} \cdot [\mathbf{w}]_i + [q]_i$. Upon receiving $v(j)$ from $\mathcal{P}_j$, a node adds $(j, v(j))$ to a new local accumulating set $\mathsf{T}_{\mathsf{oec}}$ and continuously runs online error-correction once $|\mathsf{T}_{\mathsf{oec}}| = n-f$. The decoder tolerates up to f Byzantine evaluations and eventually reconstructs the unique degree-f polynomial $v(x) = \mathbf{r} \cdot \mathbf{w}(x) + q(x)$. Here, $q(x)$ serves to blind the linear combination of $\mathbf{w}(x)$, ensuring that no information is revealed to the adversary beyond the local shares $\mathbf{w}(i)$ held by corrupted nodes.

Phase ❹: public keys derivation. The goal of this phase is to identify valid public key shares from $t+1$ nodes in the locked set, and to reconstruct all the public outputs. Given the decoded verification script $v(x)$, the nodes verify every locked tuple $(\mathbf{pk}_\tau, e_\tau)$ in $\mathsf{T}_{\mathsf{lock}}$ by checking whether $\mathbf{r} \cdot \mathbf{pk}_\tau + e_\tau$ equals $v(\tau)\mathcal{G}$, a group element calculated from the script $v(x)$. Since $|\mathsf{T}_{\mathsf{lock}}| = n-f$ and at most f tuples are adversarial, $n-2f \geq t+1$ valid tuples are guaranteed to be identified. As shown in Appendix D, once the locked tuple is fixed, an incorrect public-key share passes the random linear check with probability at most $1/p$. Hence, every accepted public-key share

satisfies $\mathbf{pk}_\tau = \mathbf{w}(\tau)\mathcal{G}$, except with negligible probability. From the $t+1$ valid public key shares, evaluations on $\mathbf{w}(x)\mathcal{G}$ at $x \in [0, n]$ are interpolated. Thus, the public keys $\mathbf{pk} = \mathbf{w}\mathcal{G}$ equal $\mathbf{w}(0)\mathcal{G}$, and the public key shares $\mathbf{pk}_j = [\mathbf{w}]_j\mathcal{G}$ for $\mathcal{P}_j$ equal $\mathbf{w}(j)\mathcal{G}$.

Challenge vector derivation. Recall that the verification script satisfies $v(x) = \mathbf{r} \cdot \mathbf{w}(x) + q(x)$, and a locked tuple $(\mathbf{pk}_\tau, e_\tau)$ is accepted only if $v(\tau)\mathcal{G} = \mathbf{r} \cdot \mathbf{pk}_\tau + e_\tau$. Soundness therefore requires the tuples to be fixed before $\mathbf{r}$ becomes known, while all honest nodes must derive the same challenge. The nodes first invoke MVBA to agree on an index set T of size $n - f$; the corresponding reliable broadcasts yield the common locked set $\mathsf{T}_{\mathsf{lock}} = \{(\mathbf{pk}_\tau, e_\tau) : \tau \in \mathsf{T}\}$. To derive a post-lock challenge without invoking a standalone random beacon [9, 41, 58], the lightweight randomness-sharing phase additionally generates an independent degree-f sharing $[d]$ of a random seed d. After constructing $\mathsf{T}_{\mathsf{lock}}$, each honest node reveals its seed share, and the nodes robustly reconstruct d through online error-correction and derive $\mathbf{r}$ by hashing a domain-separated encoding of the session identifier, d, and $\mathsf{T}_{\mathsf{lock}}$. Since d remains hidden until the locked set is fixed, $\mathbf{r}$ is independent of the committed payloads except with negligible probability of guessing d in a random-oracle query; agreement on $\mathsf{T}_{\mathsf{lock}}$ and unique reconstruction of d ensure that all honest nodes obtain the same challenge.

5 Kairos: High-Threshold ADKG Construction

Kairos is a high-threshold ADKG scheme that generates the sharings of public-private key pairs, which supports a high reconstruction threshold that exceeds the number of corrupted nodes. Its public input includes the field $\mathbb{Z}_p$, the committee size n, malicious node tolerance f, reconstruction threshold $t \in [f, n-f)$, batch size b, and a generator $\mathcal{G}$. For description convenience, we set $h = t - f + 1$, which also represents the minimal number of shares needed from honest nodes in reconstruction. We use $\boldsymbol{\theta}_k$ to denote the k-th standard basis vector with 1 at coordinate k, 0 elsewhere; $|\boldsymbol{\theta}_k| = f + 1$. The security analysis is given in Appendix D, where the challenge hash is modeled as a random oracle with range $\mathbb{Z}_p^b$.

Algorithm 3 illustrates the procedure of Kairos in the happy path. Initially, the committee nodes produce low-threshold Shamir's sharings and transform them into high-threshold private key shares through our *sk*-NTL technique (Phase ❶). The nodes then reliably broadcast their public key shares and lock $n - f$ valid broadcast instances through consensus (Phase ❷). After deriving a challenge vector from the locked set and a hidden seed, each node calculates multiple intermediate results through random linear combination, and multicasts them to aggregate the high-threshold verification script through our *pk*-HVS technique (Phase ❸). If valid public key shares from $t+1$ nodes are filtered from the locked $n - f$ ones, the public outputs are interpolated (Phase ❹). Under adversarial manipulation, the valid filtered shares may be fewer than $t+1$, triggering the passive path. We first present an intuitive multi-iteration construction of the passive path (Phase ❺) and then strengthen it to a constant round (Phase #❺, Algorithm 4).

Algorithm 3 Kairos: high-threshold batched ADKG

input: $\mathbb{Z}_p$, n, f (malicious node tolerance), $t \in [f, n-f)$ (reconstruction threshold), b (output batch size), and $\mathcal{G}$ (a generator of group $\mathbb{G}$)
private output: $\mathbf{sk}_i$ (private key shares for each node $\mathcal{P}_i$)
public output: $\mathbf{pk}$ (public keys), and $\{\mathbf{pk}_j\}_{j\in[n]}$ (public key shares)
notation: $h := t - f + 1$ (the number of low-degree components used in the threshold-lifting transformation), and $\boldsymbol{\theta}_k$ (the k-th standard basis vector with 1 at coordinate k, 0 elsewhere; $|\boldsymbol{\theta}_k| = f+1$)

Phase ❶: high-threshold private key generation (as node $\mathcal{P}_i$)
• invoke Algorithm 2 with batch size $h(b+f+1)+f+1$
upon outputting $[\mathbf{u}]_i$ from Algorithm 2 with $|\mathbf{u}| = h(b+f+1)+f+1$ **do**
  • let the first $f+1$ entries in $[\mathbf{u}]_i$ be $[d_0]_i, \ldots, [d_f]_i$
  • partition the remaining $h(b+f+1)$ entries into h groups of size $b+f+1$
  • denote these groups as $[\mathbf{h}_0]_i, \ldots, [\mathbf{h}_{h-1}]_i$
  • $\mathbf{F}(i) := [\mathbf{h}_0]_i + i[\mathbf{h}_1]_i + \cdots + i^{h-1}[\mathbf{h}_{h-1}]_i$ // $|\mathbf{F}(i)| = b+f+1$
  • denote the first b entries in $\mathbf{F}(i)$ as $\mathbf{H}(i)$
    ◦ the shares of the key pairs are $\mathbf{sk}_i := \mathbf{H}(i)$ and $\mathbf{pk}_i := \mathbf{H}(i)\mathcal{G}$
  • denote the last $f+1$ entries in $\mathbf{F}(i)$ as $\mathbf{E}(i)$
    ◦ $\mathbf{E}(i)$ is a blinding share and $\mathbf{e}_i := \mathbf{E}(i)\mathcal{G}$ is the public blinding share

Phase ❷: public key shares locking (as node $\mathcal{P}_i$)
• reliable broadcast $(\mathbf{pk}_j, \mathbf{e}_j)$ // similar to Phase 2 in Algorithm 1
upon receiving $(\mathbf{pk}_j, \mathbf{e}_j)$ from node $\mathcal{P}_j$ **do**
  ◦ add the index j into the local accumulating set T_i
  upon $|\mathsf{T}_i| = n - f$ **do** invoke MVBA with input T_i
upon outputting the agreed set T from MVBA **do**
  ◦ gather the locked public key shares $\mathsf{T}_{\mathsf{lock}} := \{(\mathbf{pk}_\tau, \mathbf{e}_\tau) : \tau \in \mathsf{T}\}$

Phase ❸: high-threshold script generation (as node $\mathcal{P}_i$)
upon constructing the locked set $\mathsf{T}_{\mathsf{lock}}$ **do**
  ◦ multicast $[d_0]_i$ among all nodes
upon receiving $[d_0]_j$ from node $\mathcal{P}_j$ **do**
  ◦ add $(j, [d_0]_j)$ into a local accumulating set $\mathsf{T}_{\mathsf{oec}}$
  upon $|\mathsf{T}_{\mathsf{oec}}| = n - f$ **do** continuously run online error-correction on $\mathsf{T}_{\mathsf{oec}}$
upon obtaining d_0 from online error-correction and constructing $\mathsf{T}_{\mathsf{lock}}$ **do**
  ◦ calculate $\mathbf{r}$ by hashing $(\mathsf{sid}, \mathsf{T}_{\mathsf{lock}}, d_0)$
  for each $j \in [0, h)$ **do** calculate $v_j(i) := (\mathbf{r} \parallel \boldsymbol{\theta}_1) \cdot [\mathbf{h}_j]_i$
  ◦ multicast $v_0(i), \ldots, v_{h-1}(i)$ to all nodes
  ◦ robustly reconstruct $v_0(x), \ldots, v_{h-1}(x)$ from online error-correction
  ◦ calculate $v(x) := v_0(x) + xv_1(x) + \cdots + x^{h-1}v_{h-1}(x)$

Phase ❹: public keys derivation (as node $\mathcal{P}_i$)
upon obtaining verification script $v(x)$ **do**
  for each $\tau \in \mathsf{T}$, **if** $v(\tau)\mathcal{G} = \mathbf{r} \cdot \mathbf{pk}_\tau + \boldsymbol{\theta}_1 \cdot \mathbf{e}_\tau$ **do**
    • add $(\tau, \mathbf{pk}_\tau)$ into set T_{pk} of public key shares // $\mathbf{pk}_\tau = \mathbf{H}(\tau)\mathcal{G}$
  if $|\mathsf{T}_{\mathsf{pk}}| \geq t + 1$ **do**
    • interpolate $t+1$ elements in T_{pk} into $\mathbf{H}(0)\mathcal{G}$ and $\{\mathbf{H}(j)\mathcal{G}\}_{j\in[n]}$
    • output $\mathbf{pk} := \mathbf{H}(0)\mathcal{G}$ and $\mathbf{pk}_j := \mathbf{H}(j)\mathcal{G}$ for all $j \in [n]$
  else initiate Phase 5, the passive path

5.1 Concrete Scheme

Phase ❶: lightweight private key generation. The objective of this phase is to produce *high-threshold* private key shares. First, all nodes invoke Algorithm 2, the lightweight randomness sharing, to produce the (n, f)-threshold Shamir's sharings of $h(b+f+1)+f+1$ random values $\mathbf{u}$, such that every node $\mathcal{P}_i$ outputs a low-threshold share $[\mathbf{u}]_i$. The first $f+1$ entries are denoted as $[d_0]_i, \ldots, [d_f]_i$ and reserved as independent hidden seeds for delayed opening. The remaining (n, f)-threshold sharings are then locally transformed into (n, t)-threshold sharings through our non-interactive threshold lifting technique. Specifically, each node $\mathcal{P}_i$ partitions the $h(b+f+1)$ low-threshold shares $[\mathbf{u}]_i$ into h groups of size $b+f+1$, denoted as $[\mathbf{h}_0]_i, \ldots, [\mathbf{h}_{h-1}]_i$. The high-threshold shares are then calculated as $\mathbf{F}(i) = [\mathbf{h}_0]_i + i[\mathbf{h}_1]_i + \cdots + i^{h-1}[\mathbf{h}_{h-1}]_i$. By construction, each $[\mathbf{h}_k]_i$ represents the vector of evaluations on f-degree polynomials

$\mathbf{h}_k(x)$ at $x = i$. Then $\mathbf{F}(i)$ lies on t-degree $\mathbf{F}(x)$, where

$$\mathbf{F}(x) = \mathbf{h}_0(x) + x\mathbf{h}_1(x) + \cdots + x^{h-1}\mathbf{h}_{h-1}(x),\ |\mathbf{F}| = b+f+1 \quad (3)$$

Let $\mathbf{H}(i)$ denote the first b entries of $\mathbf{F}(i)$. The private key shares for each $\mathcal{P}_i$ are $\mathbf{sk}_i = \mathbf{H}(i)$ and the public key shares are $\mathbf{pk}_i = \mathbf{H}(i)\mathcal{G}$. The last $f+1$ entries $\mathbf{E}(i)$ of $\mathbf{F}(i)$ are the blinding shares, and their mappings $\mathbf{e}_i = \mathbf{E}(i)\mathcal{G}$ in $\mathbb{G}$ represent the public blinding shares.

Phase ❷: public key shares locking. This phase locks a globally consistent set $\mathsf{T}_{\mathsf{lock}}$ of public key shares from $n-f$ nodes, and is identical to Phase 2 in Algorithm 1, LT-Kairos.

Phase ❸: high-threshold script generation. The goal of this phase is to generate a globally consistent, high-threshold verification script for validating the locked public key shares. After constructing the complete locked set $\mathsf{T}_{\mathsf{lock}}$, each node $\mathcal{P}_i$ multicasts its share $[d_0]_i$ of the first hidden seed. The nodes robustly reconstruct d_0 through online error-correction and derive the b-dimensional challenge vector $\mathbf{r}$ by hashing a domain-separated encoding of the session identifier sid, $\mathsf{T}_{\mathsf{lock}}$, and d_0. Since $\mathsf{T}_{\mathsf{lock}}$ is irrevocably fixed before any honest share of d_0 is revealed, the challenge remains unpredictable when the locked public key shares are chosen.

Each node $\mathcal{P}_i$ then computes $v_k(i) := (\mathbf{r} \parallel \boldsymbol{\theta}_1) \cdot [\mathbf{h}_k]_i$ for every $k \in [0, h)$, where $\boldsymbol{\theta}_1 = (1, 0, \ldots, 0)$ is the $(f+1)$-dimensional standard basis vector selecting the first blinding entry. It multicasts $v_0(i), \ldots, v_{h-1}(i)$ to all nodes, continuously runs online error-correction, and eventually reconstructs the unique f-degree component polynomials $v_0(x), \ldots, v_{h-1}(x)$.

Denoting $\mathbf{h}_k(x)$ as the encoding polynomials for sharing $[\mathbf{h}_k]$, there is $v_k(x) = (\mathbf{r} \parallel \boldsymbol{\theta}_1) \cdot \mathbf{h}_k(x)$ for $k \in [0, h)$. The high-threshold verification script is calculated as the t-degree polynomial $v(x)$:

$$\begin{aligned} v(x) &= v_0(x) + xv_1(x) + \cdots + x^{h-1}v_{h-1}(x) \\ &= (\mathbf{r} \parallel \boldsymbol{\theta}_1) \cdot (\mathbf{h}_0(x) + x\mathbf{h}_1(x) + \cdots + x^{h-1}\mathbf{h}_{h-1}(x)) \end{aligned}$$

According to Equation (3), the inner polynomial equals $\mathbf{F}(x)$. Then:

$$v(x) = (\mathbf{r} \parallel \boldsymbol{\theta}_1) \cdot \mathbf{F}(x) = \mathbf{r} \cdot \mathbf{H}(x) + \boldsymbol{\theta}_1 \cdot \mathbf{E}(x)$$

Here, $\mathbf{H}(x)$ are the encoding polynomials for private key shares $\mathbf{H}(i)$, and $\boldsymbol{\theta}_1 \cdot \mathbf{E}(x)$ extracts the first entry in $\mathbf{E}(x)$ as the blinding polynomial, which hides the distribution of $\mathbf{H}(x)$ given $v(x)$. The remaining entries of $\mathbf{E}(x)$ are reserved for the passive path.

Phase ❹: public keys derivation. The objective of this phase is to identify valid high-threshold public key shares from the locked set. Specifically, given the high-threshold verification script $v(x)$ of t-degree, the nodes verify every tuple $(\mathbf{pk}_\tau, \mathbf{e}_\tau)$ in the locked set $\mathsf{T}_{\mathsf{lock}}$ by checking whether $\mathbf{r} \cdot \mathbf{pk}_\tau + \boldsymbol{\theta}_1 \cdot \mathbf{e}_\tau$ equals $v(\tau)\mathcal{G}$, where $\mathbf{pk}_\tau = \mathbf{H}(\tau)\mathcal{G}$ and $\mathbf{e}_\tau = \mathbf{E}(\tau)\mathcal{G}$ are the public key shares and public blinding shares, respectively. Shares that satisfy the check are collected into T_{pk}, and the rest are discarded. This phase proceeds similarly to Phase 4 in Algorithm 1, LT-Kairos. However, this high-threshold variant may not be able to collect $t+1$ valid public key shares for reconstructing the public outputs, as the locked set $\mathsf{T}_{\mathsf{lock}}$ contains the tuples from $n-f$ nodes and at most f may be adversarial, and in some cases $n-2f \le t+1$. Consequently, the passive path is triggered to complete the key derivation if $|\mathsf{T}_{\mathsf{pk}}| < t+1$.

Algorithm 4 Constant-round passive path of Kairos (as node $\mathcal{P}_i$)

input: T (the MVBA output in happy path, consisting of $n-f$ indices of locked broadcast instances), T_{pk} (the set of valid public key shares in happy path)
notation: $h = t-f+1$, and $\boldsymbol{\theta}_k$ (the k-th standard basis vector with $|\boldsymbol{\theta}_k| = f+1$)

1: **parallel for** $\ell \in [n] \backslash \mathsf{T}$ **do**
2: • wait $(\mathbf{pk}_\ell, \mathbf{e}_\ell)$ from the reliable broadcast instance initiated by $\mathcal{P}_\ell$
3: • denote ℓ as the μ-th index in sorted list $[n] \backslash \mathsf{T}$
4: • multicast $(\mathsf{ins}_\ell, [d_\mu]_i)$ among all nodes
5: • robustly reconstruct d_μ from online error-correction
6: • calculate b-dimension challenge $\mathbf{r}$ by hashing $(\mathsf{sid}, \mathsf{ins}_\ell, d_\mu, \mathbf{pk}_\ell, \mathbf{e}_\ell)$
7: • calculate $v_k(i) = (\mathbf{r} \parallel \boldsymbol{\theta}_{\mu+1}) \cdot [\mathbf{h}_k]_i$ for each $k \in [0, h)$
8: • multicast $(\mathsf{ins}_\ell, v_0(i), \ldots, v_{h-1}(i))$ to all nodes
9: **upon** receiving $(\mathsf{ins}_\ell, v_0(j), \ldots, v_{h-1}(j))$ from node $\mathcal{P}_j$ **do**
10: ∘ add $(j, \{v_0(j), \ldots, v_{h-1}(j)\})$ into the accumulating set $\mathsf{T}_{\mathsf{oec},\ell}$
11: **upon** $|\mathsf{T}_{\mathsf{oec},\ell}| = n - f$ **do** run online error-correction on $\mathsf{T}_{\mathsf{oec},\ell}$
12: **upon** outputting $v_0(x), \ldots, v_{h-1}(x)$ from online error-correction **do**
13: ∘ calculate $v(x) = v_0(x) + xv_1(x) + \cdots + x^{h-1}v_{h-1}(x)$
14: **if** $v(\ell)\mathcal{G} = \mathbf{r} \cdot \mathbf{pk}_\ell + \boldsymbol{\theta}_{\mu+1} \cdot \mathbf{e}_\ell$ **do** add $(\ell, \mathbf{pk}_\ell)$ into global set T_{pk}
15: **upon** $|\mathsf{T}_{\mathsf{pk}}| \ge t+1$ **do**
16: • interpolate $t+1$ elements in T_{pk} into $\mathbf{H}(0)\mathcal{G}$ and $\{\mathbf{H}(j)\mathcal{G}\}_{j \in [n]}$
17: • output $\mathbf{pk} = \mathbf{H}(0)\mathcal{G}$ and $\mathbf{pk}_j = \mathbf{H}(j)\mathcal{G}$ for all $j \in [n]$

5.2 Two Constructions of the Passive Path

The passive path relies on the following observation. Each invalid public key share in the locked set $\mathsf{T}_{\mathsf{lock}}$ certifies a corresponding malicious node that reliably broadcasts an invalid tuple for verification. In an asynchronous network, messages between honest parties are eventually delivered; thus, by awaiting additional unlocked public key shares from broadcasts, we can identify more valid shares and accumulate a sufficient pool from at least $n-f$ honest broadcasts eventually. Since $n-f \ge t+1$, reconstruction of the public outputs is guaranteed. In this section, we first present an intuitive construction that incrementally accumulates valid public key shares in multiple iterations, and then refine it into a constant-round construction.

Phase ❺: passive path (multi-iteration construction). As illustrated in Figure 5, if fewer than $t+1$ valid public key shares are obtained in the happy path, the nodes repeatedly lock additional, previously unselected broadcast instances through MVBA. After each new set is fixed, they open the corresponding unused hidden seed, derive a fresh challenge, and re-execute Phases ❸–❹ using the corresponding unused standard basis vector to select an independent blinding polynomial. The process terminates in at most $t-f$ iterations once $t+1$ valid public key shares have been collected.

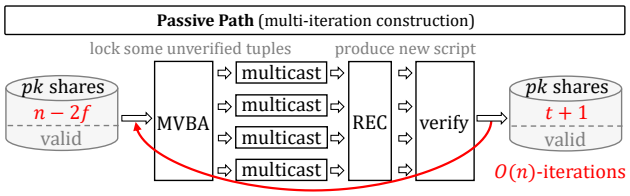


Figure 5: Illustration of the multi-iteration passive path

Phase #❺: passive path (constant-round construction). To reduce the round complexity, we further provide a constant-round construction, as shown in Algorithm 4. To avoid repeatedly invoking MVBA, Algorithm 4 verifies all remaining broadcast instances independently and in parallel. From the happy path, we inherit two

sets: T is the output of MVBA and contains $n - f$ reliable broadcast instances that are locked for verification; T_{pk} contains all valid public key shares that have been identified. For every *unlocked* broadcast instance ℓ in $[n]\backslash\mathsf{T}$ initiated by $\mathcal{P}_\ell$, let μ denote the position of ℓ in the sorted list $[n]\backslash\mathsf{T}$, and $\boldsymbol{\theta}_{\mu+1}$ is the $(\mu{+}1)$-th standard basis vector with 1 at coordinate $\mu{+}1$ and 0 elsewhere, which serves to extract an independent blinding entry for the verification of $\mathbf{pk}_\ell$.

After delivering $(\mathbf{pk}_\ell, \mathbf{e}_\ell)$ from the reliable broadcast instance initiated by $\mathcal{P}_\ell$, each node multicasts its share $[d_\mu]_i$ and robustly reconstructs the corresponding hidden seed d_μ. The nodes then derive an instance-specific challenge vector $\mathbf{r}$ by hashing a domain-separated encoding of sid, the broadcast-instance identifier ins_ℓ, d_μ, and $(\mathbf{pk}_\ell, \mathbf{e}_\ell)$. For every $k \in [0, h)$, node $\mathcal{P}_i$ computes $v_k(i) := (\mathbf{r} \parallel \boldsymbol{\theta}_{\mu+1}) \cdot [\mathbf{h}_k]_i$ and multicasts the resulting evaluations tagged with ins_ℓ. Here, $\mathbf{r}$ aggregates the b key coordinates, while the standard basis vector $\boldsymbol{\theta}_{\mu+1}$ selects the independent blinding coordinate assigned to this passive instance. Using an instance-specific online error-correction process, the nodes reconstruct $v_0(x), \ldots, v_{h-1}(x)$ and aggregate them into $v(x) = v_0(x) + xv_1(x) + \cdots + x^{h-1}v_{h-1}(x)$. The tuple is accepted if $v(\ell)\mathcal{G} = \mathbf{r} \cdot \mathbf{pk}_\ell + \boldsymbol{\theta}_{\mu+1} \cdot \mathbf{e}_\ell$, in which case $(\ell, \mathbf{pk}_\ell)$ is added to the global set T_{pk}. As soon as $|\mathsf{T}_{\mathsf{pk}}| \geq t + 1$, the nodes interpolate any $t + 1$ valid public key shares to recover the public keys and the complete vector of public key shares.

This construction is constant round because it replaces the $O(n)$ serial MVBA with $O(n)$ parallel multicast. Termination follows since the honest broadcasts outside the locked set are eventually delivered and are sufficient to obtain $t{+}1$ valid public key shares. Correctness follows because any inconsistent tuple passes its instance-specific random linear check only with negligible probability. Secrecy is preserved because each instance derives its challenge from an independently opened hidden seed after the tuple is fixed and uses a distinct independent blinding polynomial $\boldsymbol{\theta}_{\mu+1} \cdot \mathbf{E}(x)$.

6 Application

6.1 ADPSS from *sk*-NTL and *pk*-HVS

Motivation. Asynchronous dynamic proactive secret sharing enables proactive refreshing of secret shares across dynamic committee reconfiguration, and is widely used in multi-party computation [65], blockchain [35, 50], and distributed key management [60] to support private state transfer. Building on substantial progress in ADPSS [14, 29, 45, 46, 62, 67, 69], the best-known high-threshold protocol LongLive [67] attains an amortized communication of $O(\kappa n^2)$. Despite being communicationally efficient, its tight coupling to high-threshold ACSS under homomorphic commitments incurs a high computational cost. Recent efforts [10, 61] propose two hash-based low-threshold ADPSS protocols with amortized linear communication. Among them, the constant-round construction GoSSamer [61] achieves more than 10× speedups over the low-threshold variant of LongLive [67]. These advances, however, do not address the principal obstacle to efficient ADPSS with a high reconstruction threshold, where each share should be accompanied by a correctness proof to enable downstream consumption.

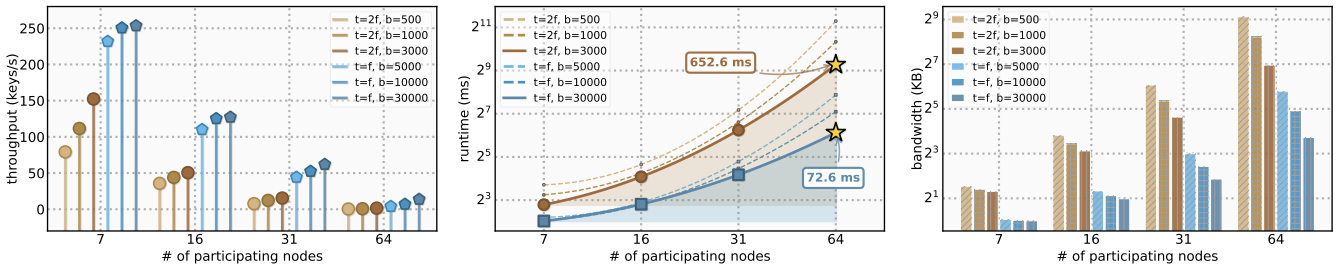
Definition 1. An $(\mathsf{com}_\mathsf{o}, \mathsf{com}, b)$-ADPSS protocol facilitates the *proactive* resharing of b secrets $\mathbf{s} \in \mathbb{Z}_p^b$ from an old committee com_o to a *dynamically* reconfigured new committee com. Specifically, the old committee consists of $n_\mathsf{o} = 3f_\mathsf{o} + 1$ nodes $\mathcal{P}_{\mathsf{o},1}, \ldots, \mathcal{P}_{\mathsf{o},n_\mathsf{o}}$ with f_o denoting the malicious node tolerance. Each node $\mathcal{P}_{\mathsf{o},i}$ in com_o holds an $(n_\mathsf{o}, t_\mathsf{o})$-threshold share $[\mathbf{s}]_{\mathsf{o},i}$ of $\mathbf{s}$. The protocol enables the nodes in the old committee to collaboratively reshare the secrets to the new committee of $n = 3f + 1$ nodes, such that each node $\mathcal{P}_i$ obtains an (n, t)-threshold share $[\mathbf{s}]_i$ of the same secrets $\mathbf{s}$. The protocol is low-threshold if $t = f$; high-threshold if it supports $t \in [f, n - f)$. An ADPSS protocol satisfies:

- **Correctness**: Suppose there exists a vector of t_o-degree polynomials $\mathbf{f}_\mathsf{o}(x)$ with $\mathbf{f}_\mathsf{o}(0) = \mathbf{s}$, and each node $\mathcal{P}_{\mathsf{o},i}$ in the old committee holds the share $[\mathbf{s}]_{\mathsf{o},i} = \mathbf{f}_\mathsf{o}(i)$. If an honest node $\mathcal{P}_i$ in the new committee com outputs its share $[\mathbf{s}]_i$, then all honest nodes in com output their shares eventually, and there exists a vector of t-degree polynomials $\mathbf{f}(x)$ such that $\mathbf{f}(0) = \mathbf{s}$ and $[\mathbf{s}]_i = \mathbf{f}(i)$ for each honest node $\mathcal{P}_i$.
- **Termination**: If all honest nodes in com_o initiate the protocol, all honest nodes in com output eventually.
- **Secrecy**: No information about $\mathbf{s}$ is revealed to the adversary $\mathcal{A}$ beyond the public verification information $\mathbf{s}\mathcal{G}$.

Construction. We integrate *sk*-NTL and *pk*-HVS into the low-threshold ADPSS workflow to derive high-threshold sharings and their public verification information without invoking $O(n)$ parallel high-threshold ACSS instances with homomorphic commitments. The old and new committees first execute the underlying light-weight ADPSS protocol, after which the new committee obtains sufficiently many independent (n, f)-threshold sharings of the original secrets and auxiliary randomness. Each new-committee node then locally applies *sk*-NTL to obtain the desired (n, t)-threshold sharings. To certify the corresponding public shares, the nodes reliably broadcast their group images, lock a common subset through MVBA, and invoke *pk*-HVS with delayed-seed challenges to identify at least $t{+}1$ valid evaluations. The new committee subsequently interpolates these evaluations in the exponent to obtain the complete public verification information. Meanwhile, it retains the low-threshold sharings required to refresh the secrets into the next committee. The resulting protocol achieves high-threshold reconstruction with $O(\kappa n^2)$ amortized communication, when instantiated with the hash-based ADPSS, GoSSamer [61]. Theoretically, our amortized communication matches the best-known result in LongLive [67], while avoiding the computationally expensive homomorphic commitments. Practically, our solution performs better in both runtime and communication, as evaluated in Section 7.7.

6.2 Distributed Key Management from ADKG

DKM infrastructures such as Web3Auth [60] in MetaMask exemplify the use of high-threshold ADKG to deliver decentralized and non-custodial control over user keys. In such systems, mutually distrustful nodes, operated by independent large organizations, jointly generate large volumes of unbiased threshold key pairs via ADKG [22]. The provisioning workflow naturally supports batching. Operators can pre-generate sizeable pools of such keys during low utilization periods and allocate to arriving requests. The stored key sharings are refreshed across epochs or reconfigured committees by ADPSS. These threshold keys support broad applications such as authentication platforms [60], wallet custody services [6], and cryptographic operations used in Byzantine agreement and blockchain, such as threshold signatures [12] and threshold decryption [26].



(a) Peak Throughput (b) Amortized Runtime (# per key) (c) Bandwidth (# per key, # per node)

Figure 6: Evaluation of Kairos

Current deployments [60] in this domain typically operate with a few tens of nodes, where the asymptotic batch size of $O(n^2 \log n)$ in Kairos falls in a practically manageable range. In these settings, Kairos mitigates deployment bottlenecks in robust key generation by delivering high batch-processing efficiency.

7 Evaluation

This section evaluates the practical performance of Kairos and its application in high-threshold ADPSS. We implement LT-Kairos, Kairos, GS24 [41] with GoSSamer [61] as the underlying ACSS, a theoretical high-threshold batched ADKG scheme HARTS [8], and a Kairos-based ADPSS built atop GoSSamer [61]. We also reproduce DXKR23 [22], DYX+22 [24], and LongLive [67] for comparison. Our code is available at https://github.com/xxxsh1x1annva/Kairos.

Implementation details. For a fair comparison, we ensure all implementations and the baselines are developed in Python 3.7.13, with the core cryptographic operations delegated to a Rust-based library from Zcash [39]. The `BLS12-381` curve is used to support downstream pairing-based primitives. We also equip all implementations with the same components: the reliable broadcast from [23], the online error-correction from [34], and the MVBA from [24]. Unless otherwise stated, the performance results reported in this paper primarily reflect good case executions. We acknowledge that prior work may have optimized fast paths. This consideration is particularly relevant to HARTS [8] and GS24 [41], which are mainly theoretical works, as well as to DYX+22 [24] and DXKR23 [22], whose original implementations do not support batch processing.

Evaluation environment. Evaluations are conducted over the wide-area networks (WAN) unless explicitly stated otherwise. Each node is instantiated as an independent Python process bound to a network port, and run on an independent Amazon EC2 r5n.xlarge instance with 4 vCPUs and 32GB of RAM. Nodes are pair-wise connected and evenly distributed across 4 regions, including California, Hong Kong, Sydney, and London, to emulate WAN conditions.

7.1 Practical Evaluation of Kairos

Throughput and runtime. To evaluate Kairos's practical performance, we measure the throughput and amortized runtime, defined as the number of key pairs generated per second, and the end-to-end latency divided by the batch size, respectively. We vary the committee size at n={7, 16, 31, 64}=$3f+1$. At the threshold $t = f$, we set the batch size $b = \{5\text{k}, 10\text{k}, 30\text{k}\}$; at a higher threshold $t = 2f$, we set $b = \{0.5\text{k}, 1\text{k}, 3\text{k}\}$. As shown in Figure 6(a), Kairos demonstrates high batch-processing efficiency, achieving a peak throughput of 253 keys/s at threshold $t = f$ and 152 keys/s at threshold $t = 2f$ with $n = 7$. As the network scales, the optimal batch size to achieve the best throughput increases. Even in a small batch size, Kairos can still achieve a throughput of 15 keys/s at $t = 2f, b = 3\text{k}$ with $n = 31$, and 14 keys/s at $t = f, b = 30\text{k}$ with $n = 64$. Figure 6(b) further illustrates the amortized runtime. At $t = 2f$, the runtime remains below 90 ms for $n \leq 31$ and $b \geq 1\text{k}$; while at $t = f$, the runtime decreases to below 20 ms for $n \leq 31$ and batch size $b \geq 10\text{k}$. These results indicate that Kairos sustains high processing rates with short per-key latency, which is well-suited to deployments for threshold Schnorr [12] and distributed key management systems [6, 60].

Bandwidth. Figure 6(c) reports the *amortized bandwidth usage*, defined as the total volume of data sent by each node, divided by the batch size. At threshold $t = 2f$, the amortized bandwidth usage remains below 50 KB for $n \leq 31$ and batch size $b \geq 1\text{k}$; while at threshold $t = f$, it remains below 10 KB for $n \leq 31$. Even within a 64-node-committee, Kairos can still achieve an amortized bandwidth usage of 124.3 KB for $t = 2f, b = 3\text{k}$ and 13.3 KB for $t = f, b = 30\text{k}$.

7.2 High-threshold ADKG Comparison

Baseline. For high-threshold ADKG, we adopt the state-of-the-art implementation DXKR23 [22] and a batched scheme HARTS [8] as our primary baselines. We first implement the theoretical solution HARTS [8], and then extend non-batched ADKG baseline DXKR23 [22] to support batching via minimal changes. The batched variant preserves the original protocol structure. The key modification is to replace the non-batched ACSS with the state-of-the-art batched ACSS instantiation, hbACSS [66], which retains the required homomorphic commitments and delivers high batch-processing efficiency. In this setting, we do not include FT25 [32] as an experimental baseline. The reason is that FT25 [32] relies on the silent setup and targets different optimization regimes from Kairos. While FT25 offers asymptotic quadratic communication, its concrete performance benefits become visible and outperform DXKR23 for batch size $b = 1$ and very large committees, typically in the several-hundreds range or above. This regime is less representative of the moderate committee sizes that arise in the distributed key management settings motivating this work. For these reasons, we regard DXKR23 and HARTS as the most informative baselines.

Runtime. We compare the runtime of Kairos with the baselines HARTS [8] and DXKR23 [22] at threshold $t = 2f$, enforce a per-run wall-clock time budget of 2 hours, and report the results in

Table 2: Comparison of running time (seconds, # in total) and bandwidth usage (MB, # per node) for high-threshold ADKG schemes at threshold $t = 2f$. We set batch size $b = 3k$ when the committee size n varies, and $n = 16$ when b varies.

	Value	Scheme					
		Kairos		DXKR23		HARTS	
		Time	Band.	Time	Band.	Time	Band.
Com. Size (# n)	4	13.3	2.7	83.3 ↑6.2×	12.8 ↑4.7×	2101.1↑157.5×	7.2 ↑2.6×
	7	19.7	7.3	153.6 ↑7.8×	38.0 ↑5.2×	3573.8↑181.5×	17.2 ↑2.4×
	16	59.4	25.5	479.2 ↑8.1×	187.8 ↑7.4×	4586.9 ↑77.2×	49.6 ↑1.9×
	31	194.4	72.6	1374.1 ↑7.1×	684.9 ↑9.4×	5024.9 ↑25.8×	107.1 ↑1.5×
	64	1957.7	364.2	5393.8 ↑2.8×	2867.3 ↑7.9×	6187.8 ↑3.2×	249.1 ↓0.7×
Batch Size (# b)	0.5k	14.0	6.9	80.7 ↑5.8×	31.4 ↑4.6×	150.4 ↑10.8×	9.2 ↑1.3×
	1k	22.7	10.6	161.0 ↑7.1×	62.7 ↑5.9×	539.1 ↑23.7×	17.2 ↑1.6×
	3k	59.4	25.5	479.2 ↑8.1×	187.8 ↑7.4×	4586.9 ↑77.2×	49.6 ↑1.9×
	5k	79.2	40.3	805.2 ↑10.2×	312.9 ↑7.8×	>7200	-
	10k	152.9	77.3	1629.2↑10.7×	625.6 ↑8.1×	>7200	-

Table 3: Comparison of running time (seconds, # in total) and bandwidth usage (MB, # per node) for low-threshold ADKG schemes at threshold $t = f$. We set batch size $b = 3k$ when the committee size n varies, and $n = 16$ when b varies.

	Value	Scheme							
		LT-Kairos		DYX+22 ↑		GS24 (full) ↑		GS24 (original) ↓	
		Time	Band.	Time	Band.	Time	Band.	Time	Band.
Com. Size (# n)	4	10.9	1.6	33.8	3.1	13.1	1.8	6.7	1.4
	7	14.3	3.2	107.1	27.8	18.7	3.4	7.5	2.6
	16	28.7	8.7	228.4	40.1	38.6	8.9	11.5	6.8
	31	81.4	34.1	727.2	142.2	99.7	36.7	46.6	29.1
	64	1158.1	265.3	2817.1	585.0	1201.7	267.3	1063.5	257.0
Batch Size (# b)	0.5k	8.7	4.1	37.4	6.8	9.3	4.4	4.7	3.7
	1k	12.2	5.0	76.9	13.5	15.3	5.5	5.9	4.3
	3k	28.7	8.7	228.4	40.1	38.6	8.9	11.5	6.8
	5k	43.4	12.3	380.7	66.9	61.3	12.7	16.0	9.2
	10k	77.5	21.4	760.0	133.6	106.4	22.4	28.0	15.4

Table 2, where blue arrows indicate the ratio of each baseline's running time to Kairos's running time, i.e., baseline.time/Kairos.time. Kairos consistently outperforms the baselines in all evaluated settings. With committee size $n = 16$ and varying the batch size, the relative advantage of Kairos increases as the batch size grows. Specifically, DXKR23 [22] requires 5.8-10.7× the runtime of Kairos, and HARTS [8] requires 10.8-77.2× the runtime of Kairos. With batch size $b = 3$k and varying the committee size, DXKR23 [22] requires 2.8-8.1× the runtime of Kairos, and HARTS [8] requires 3.2-157.5× the runtime of Kairos. Specifically, Kairos generates 3k keys in 194.4s with a 31-node committee, compared to 1374.1s for DXKR23 [22] and 5024.9s for HARTS [8]. Although $b = 3$k is not sufficient for Kairos to reach peak performance at $n = 64$, Kairos still reduces over 60% runtime compared to both solutions.

Bandwidth. Table 2 also reports the comparison of the per-node bandwidth usage at threshold $t = 2f$, where brown arrows indicate the ratio of each baseline's bandwidth usage to Kairos's bandwidth usage, i.e., baseline.band/Kairos.band. The results illustrate that Kairos incurs relatively low communication, followed by HARTS [8] and DXKR23 [22]. Specifically, Kairos saves 78.6-89.4% bandwidth compared to DXKR23 [22] and 32.3-62.5% compared to HARTS [8] at $n \leq 31$. At $n = 64$, Kairos requires slightly more bandwidth relative to HARTS [8], as the batch size $b = 3$k is not sufficient to demonstrate their asymptotic amortized performance, which is $O(\kappa n^2)$ for Kairos and $O(\kappa n^2 \log n)$ for HARTS [8]. Concretely, compared to Kairos, HARTS [8] reduces the bandwidth by 31.6% at $n = 64, b = 3$k, but requires 3.16× the runtime.

7.3 Low-threshold ADKG Comparison

Baseline. For low-threshold ADKG, we compare LT-Kairos against DYX+22 [24], which is built on homomorphic-commitment-based ACSS, and GS24 [41], which is built on hash-based ACSS. We first extend DYX+22 [24] to support batching using the same minimally invasive method as for DXKR23 [22]: we preserve the original protocol structure and replace its ACSS with hbACSS [66]. We then instantiate GS24 [41] with GoSSamer-ACSS [61] to enable a fair comparison of public-key derivation efficiency. Specifically, we consider two variants of GS24 [41]: the *original* version, which provides only the materials needed to derive public keys, and a *full* version, which provides all materials needed to compute both public keys and public-key shares, matching the full output notion of standard ADKG and of other baselines.

Runtime and bandwidth. Table 3 reveals three trends. First, DYX+22 [24] takes about 3.1-9.8× more runtime and 1.7-8.8× more bandwidth than LT-Kairos, confirming that the homomorphic-commitment-based ACSS is the dominant source of overhead. Second, when comparing the instantiations from hash-based ACSS, GS24 (*original*) achieves 8.2-63.8% lower runtime and 3.1-28.0% lower bandwidth than LT-Kairos. However, it produces the overall public keys required by its threshold Schnorr application, rather than the complete public-key-share vector for general usage. Third, under the straightforward full-output extension evaluated here, GS24 (*full*) is slightly more expensive than LT-Kairos, incurring 4-41% higher runtime and 0.7-12.8% higher bandwidth. This result shows that LT-Kairos retains favorable concrete efficiency even when both constructions are instantiated without homomorphic commitments. We do not claim this gap to be inherent: a more specialized full-output extension may further reduce the overhead, but would require an additional mechanism, which is outside the native GS24 construction. We further note that GS24 [41] does not provide the full-secrecy guarantee and high-threshold reconstruction for general downstream use. We therefore include it only as an auxiliary baseline for demonstrating key derivation efficiency.

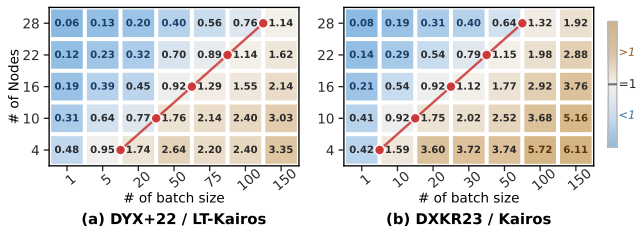


Figure 7: Empirical breakeven against DYX+22 and DXKR23, reporting the measured runtime ratio $\text{Time}_{\text{baseline}}/\text{Time}_{\text{Kairos}}$. The red frontier marks the first sampled batch size at which Kairos becomes faster than the baseline.

7.4 Breakeven Analysis

To make the practical crossover point explicit, we measure the runtime ratio ratio := $\text{Time}_{\text{baseline}}/\text{Time}_{\text{Kairos}}$, and report the results in Figure 7. A value greater than 1 indicates that Kairos is faster, and the breakeven point is reached once the ratio exceeds 1. Although the asymptotic analysis requires batch size $O(n^2 \log n)$ for Kairos to fully realize quadratic amortized communication, our measurements show that the empirical breakeven arises substantially earlier.

Figure 7 presents the empirical breakeven for LT-Kairos versus DYX+22 [24] in the low-threshold setting and for Kairos versus DXKR23 [22] in the high-threshold setting. Overall, the breakeven batch size increases with the committee size, yet remains modest throughout the tested range. For LT-Kairos versus DYX+22, the empirical breakeven intervals are $(5, 20]$, $(20, 50]$, $(50, 75]$, $(75, 100]$, and $(100, 150]$ for $n = 4, 10, 16, 22, 28$, respectively. For Kairos versus DXKR23, the corresponding intervals are $(1, 10]$, $(10, 20]$, $(20, 30]$, $(30, 50]$, and $(50, 100]$, respectively, which are consistently smaller. These measurements show that Kairos becomes preferable well before reaching the asymptotic batch regime.

7.5 Microbenchmarks for Performance Gains

Figure 8 decomposes the runtime and bandwidth into two stages: *ACSS+MVBA* and *public-key derivation*, where *ACSS+MVBA* generates random sharings within the committee. The results show that the first source for Kairos's gain is *structural decoupling from homomorphic-commitment-based ACSS*. Prior ADKG constructions such as DYX+22 [24] and DXKR23 [22] are structurally tied to homomorphic-commitment-based ACSS, as their public-key derivation reuses the homomorphic commitments. In contrast, GS24 [41] and (LT-)Kairos remove this dependency and thereby make lightweight ACSS directly usable for ADKG. This effect is visible in the first stage, where LT-Kairos and Kairos reduce the *ACSS+MVBA* cost from 189.7s and 378.9s in DYX+22 [24] and DXKR23 [22] to 5.0s and 20.3s, respectively.

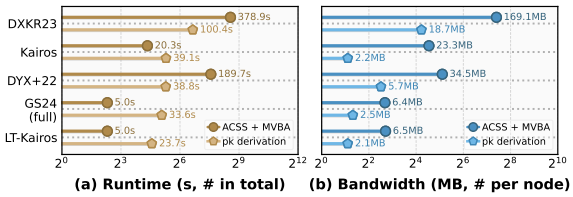


Figure 8: Microbenchmarks for Performance Gains

The second source is *faster public-key derivation with stronger functionality*. GS24 already benefits from hash-based ACSS, but it only supports biased low-threshold key generation and verifies $O(n)$ robust reconstruction for producing all public key shares. By contrast, Kairos supports high-threshold and unbiased key generation, while requiring only $O(1)$ robust reconstruction. This effect appears in the second stage: LT-Kairos reduces the public-key derivation cost from 33.6s in GS24 (full) [41] to 23.7s, while Kairos reduces it from 100.4s in DXKR23 [22] to 39.1s. The similar trend is also reflected in bandwidth, where Kairos reduces the public-key derivation bandwidth from 18.7MB in DXKR23 [22] to 2.2MB.

7.6 Evaluation under faults

Overview. LT-Kairos is our low-threshold ADKG construction. High-threshold Kairos is built on top of LT-Kairos by further incorporating the high-threshold lifting mechanism and the passive path for public key derivation. This layered design naturally leads to two separate evaluations of the two fault scenarios. Thus, we first evaluate LT-Kairos by capturing the fault handling cost of the low-threshold construction. We then evaluate the passive path in high-threshold Kairos, which measures the additional overhead introduced only in the high-threshold derivation phase.

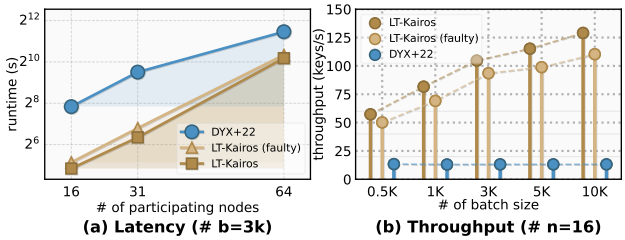


Figure 9: Evaluation of LT-Kairos with faults

LT-Kairos evaluation with faults. We evaluate LT-Kairos in a faulty setting by simulating 1/3 malicious nodes and injecting adversarial behavior into all major phases. Concretely, the malicious nodes distribute invalid shares to 1/3 honest nodes to trigger ACSS recovery, reliably broadcast forged public-key shares, and multicast forged linear-combination points $v(i)$ to trigger multiple iterations of robust interpolation for reconstructing $v(x)$. Figure 9 shows that fault handling increases the runtime of LT-Kairos by about 1.4-24.0%, demonstrating that LT-Kairos remains efficient under these failures and still outperforms baselines.

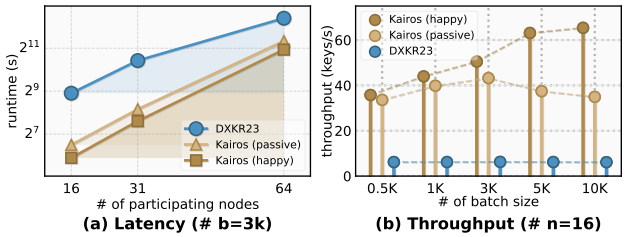


Figure 10: Evaluation of passive path

Passive path evaluation. Figure 10 illustrates the performance impact of the passive path. We simulate 25% invalid public key shares in the locked set, trigger verification of public key shares from an additional 25% reliable broadcast instances, and evaluate Kairos at threshold $t = 2f$. As shown in Figure 10(a), the passive path introduces 10.0-25.3 sec additional runtime compared to the normal path when $n = 16, 31, 64$ and $b = 3$k. As the network scales, the proportional increase drops from 16.9% to 1.3%. Figure 10(b) reports the throughput of $n = 16$ and $b = \{0.5\text{k}, 1\text{k}, 3\text{k}, 5\text{k}, 10\text{k}\}$. The passive path reaches a peak throughput of 43 keys/s at $b = 3$k, compared to 65 keys/s at $b = 10$k on the normal path. Beyond $b = 3$k, the extra batched verification, serialization, and communication costs in the passive path cause a mild throughput drop, after which the throughput gradually stabilizes. Compared to the implementation DXKR23 [22] from homomorphic-commitment-based ACSS, our passive path still achieves 63.2-85.5% runtime reduction and 5.8-10.7× throughput increase.

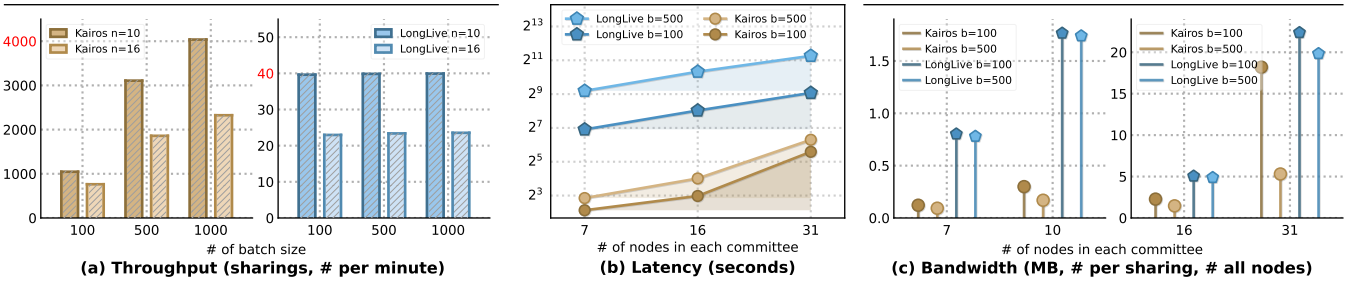


Figure 11: Application of sk-NTL and pk-HVS in high-threshold ADPSS with threshold $t = 2f$.

7.7 Application in High-Threshold ADPSS

Baseline. We reproduce the state-of-the-art high-threshold ADPSS implementation LongLive [67], which supports batch-processing and achieves $O(\kappa n^2)$ amortized communication. Our high-threshold ADPSS is built upon the hash-based low-threshold ADPSS [61], and achieves $O(\kappa n^2)$ amortized communication.

Throughput and Latency. Figure 11(a) reports the per-minute throughput comparison. When varying the committee size at $n = 10$ and $n = 16$, Kairos-based high-threshold ADPSS reshares 26.4-98.7× more secrets per minute than LongLive [67]. Figure 11(b) reports the total runtime comparison, where Kairos-based ADPSS reduces 91.0-98.7% of the runtime compared to LongLive [67], and reshares 500 secrets in 16.1s across 16-node committees.

Bandwidth. Figure 11(c) illustrates the amortized bandwidth usage, defined as the total volume of data sent among all nodes, divided by the batch size. Although both Kairos-based ADPSS and LongLive [67] achieve the same amortized communication of $O(\kappa n^2)$, our solution demonstrates better concrete efficiency. Specifically, Kairos reduces 74.7-94.4% of the bandwidth at $n \leq 16$. At $n = 31$, the evaluated batch size is not sufficient to demonstrate the asymptotic performance. However, Kairos still reduces 23.1% bandwidth at $b = 100$ and 78.3% at $b = 500$.

8 Conclusion

We present Kairos, an asymptotically and concretely efficient high-threshold ADKG from hash-based ACSS with $O(\kappa n^2)$ amortized communication, optimal resilience, and full secrecy. Theoretically, we propose a new threshold lifting technique and a lightweight verification script aggregator, under the lock-then-verify paradigm, for deriving unbiased threshold public and private keys derivation from lightweight cryptography. Concretely, these techniques achieve several-fold higher throughput than prior solutions, making it a practical and effective foundation for broad applications, including real-world distributed key management infrastructures and the first high-threshold ADPSS from lightweight cryptography.

9 Acknowledgments

This paper is supported by the Smart Grid-National Science and Technology Major Project with (2025ZD0808500); the National Natural Science Foundation of China (T2425023, 62472015, U22B2008, U2241213, 62172025); the Science and Technology Project of Guangxi Power Grid Corporation (GXKJXM20250222); Beijing Natural Science Foundation (L251003); Zhejiang Provincial Natural Science Foundation of China (LMS25F020014); Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University; and the Fundamental Research Funds for the Central Universities, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing. This work is also supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore. This work is also supported by the Beihang University Doctoral Academic Excellence Fund.

References

[1] Masayuki Abe and Serge Fehr. 2004. Adaptively secure Feldman VSS and applications to universally-composable threshold cryptography. In *CRYPTO*. Springer, 317–334.
[2] Ittai Abraham, Renas Bacho, Julian Loss, and Gilad Stern. 2025. Nearly quadratic asynchronous distributed key generation. *Cryptology ePrint Archive* (2025).
[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *CRYPTO*. Springer, 39–70.
[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *PODC*. 363–373.
[5] Nicolas Alhaddad, Mayank Varia, and Ziling Yang. 2024. Haven++: Batched and Packed Dual-Threshold Asynchronous Complete Secret Sharing with Applications. *Cryptology ePrint Archive* (2024).
[6] Lit Association. [n. d.]. Lit Protocol. https://www.litprotocol.com.
[7] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2024. GRandLine: adaptively secure DKG and randomness beacon with (log-) quadratic communication complexity. In *ACM CCS*. 941–955.
[8] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. 2024. HARTS: High-threshold, adaptively secure, and robust threshold schnorr signatures. In *ASIACRYPT*. Springer, 104–140.
[9] Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael K Reiter. 2024. Random beacons in monte carlo: Efficient asynchronous random beacon without threshold cryptography. In *ACM CCS*. 2621–2635.
[10] Akhil Bandarupalli, Xiaoyu Ji, Soham Jog, Aniket Kate, Chen-Da Liu-Zhang, and Yifan Song. 2025. Computationally and Communication Efficient Batched Asynchronous DPSS from Lightweight Cryptography. *Cryptology ePrint Archive* (2025).
[11] Michael Ben-Or, Ran Canetti, and Oded Goldreich. 1993. Asynchronous secure computation. In *STOC*. 52–61.
[12] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. 2024. Sprint: High-throughput robust distributed schnorr signatures. In *EUROCRYPT*. Springer, 62–91.
[13] Jakob Burkhardt, Ivan Damgård, Tore Kasper Frederiksen, Satrajit Ghosh, and Claudio Orlandi. 2023. Improved distributed rsa key generation using the miller-rabin test. In *ACM CCS*. 2501–2515.
[14] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. 2002. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM CCS*. 88–97.
[15] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*. Springer, 524–541.

[16] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantinpole: practical asynchronous byzantine agreement using cryptography. In *PODC*. 123–132.

[17] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Adaptive security for threshold cryptosystems. In *CRYPTO*. Springer, 98–116.

[18] John Canny and Stephen Sorkin. 2004. Practical large-scale distributed key generation. In *EUROCRYPT*. Springer, 138–152.

[19] Ignacio Cascudo and Bernardo David. 2024. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *EUROCRYPT*. Springer, 216–248.

[20] Ignacio Cascudo, Bernardo David, Omer Shlomovits, and Denis Varlakov. 2023. Mt. random: Multi-tiered randomness beacons. In *International Conference on Applied Cryptography and Network Security*. Springer, 645–674.

[21] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. 2024. Asynchronous consensus without trusted setup or public-key cryptography. In *ACM CCS*. 3242–3256.

[22] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *USENIX Security*. 5359–5376.

[23] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous data dissemination and its applications. In *ACM CCS*. 2705–2721.

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

[25] Rafael del Pino, Thomas Espitau, Guilhem Niot, and Thomas Prest. 2025. Simple and Efficient Lattice Threshold Signatures with Identifiable Aborts. *Cryptology ePrint Archive* (2025).

[26] Yvo G Desmedt. 1994. Threshold cryptography. *European Transactions on Telecommunications* 5, 4 (1994), 449–458.

[27] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. FIN: practical signature-free asynchronous common subset in constant time. In *ACM CCS*. 815–829.

[28] Thomas Espitau, Guilhem Niot, and Thomas Prest. 2024. Flood and submerse: Distributed key generation and robust threshold signature from lattices. In *CRYPTO*. Springer, 425–458.

[29] Hanwen Feng, Yingzi Gao, Yuan Lu, Qiang Tang, and Jing Xu. 2026. Practical asynchronous distributed key reconfiguration and its applications. *IEEE S&P* (2026).

[30] Hanwen Feng, Zhenliang Lu, and Qiang Tang. 2024. Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic dkg and interactive consistency. In *PODC*. 469–479.

[31] Hanwen Feng, Tiancheng Mai, and Qiang Tang. 2024. Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing. In *ACM CCS*. 2636–2650.

[32] Hanwen Feng and Qiang Tang. 2025. Asymptotically optimal adaptive asynchronous common coin and DKG with silent setup. In *CRYPTO*. Springer, 642–675.

[33] Pierre-Alain Fouque and Jacques Stern. 2001. One round threshold discrete-log key generation without private channels. In *PKC*. Springer, 300–316.

[34] Shuhong Gao. 2003. A new algorithm for decoding Reed-Solomon codes. In *Communications, information and network security*. Springer, 55–68.

[35] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Yuyi Wang, and Jing Xu. 2025. Turritopsis: Practical Dynamic Asynchronous BFT. *IEEE TIFS* (2025).

[36] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient asynchronous byzantine agreement without private setups. In *ICDCS*. IEEE, 246–257.

[37] Matthias Geihs and Hart Montgomery. 2024. LaKey: Efficient lattice-based distributed PRFs enable scalable distributed key management. In *USENIX Security*.

[38] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*. Springer, 295–310.

[39] Jack Grigg and Sean Bowe. [n. d.]. zkcrypto/pairing. https://github.com/zkcrypto.

[40] Jens Groth. 2021. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive* (2021).

[41] Jens Groth and Victor Shoup. 2024. Fast batched asynchronous distributed key generation. In *EUROCRYPT*. Springer, 370–400.

[42] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In *NDSS*. The Internet Society.

[43] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable distributed key generation. In *EUROCRYPT*. Springer, 147–176.

[44] Martin Hirt and Jesper Buus Nielsen. 2006. Robust multiparty computation with linear communication complexity. In *CRYPTO*. Springer, 463–482.

[45] Bin Hu, Jianwei Liu, Zhenliang Lu, Qiang Tang, Zhuolun Xiang, and Zongyang Zhang. 2026. Optimistic Asynchronous Dynamic-committee Proactive Secret Sharing. *IEEE S&P* (2026).

[46] Bin Hu, Zongyang Zhang, Han Chen, You Zhou, Huazu Jiang, and Jianwei Liu. 2022. DyCAPS: Asynchronous Dynamic-committee Proactive Secret Sharing. *Cryptology ePrint Archive* (2022).

[47] Stanisław Jarecki and Anna Lysyanskaya. 2000. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *EUROCRYPT*. Springer, 221–242.

[48] Yunfeng Ji, Yang Tao, and Rui Zhang. [n. d.]. Robust (T, N)-Threshold Lattice Signature. *Available at SSRN 4588269* ([n. d.]).

[49] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2024. Non-interactive VSS using class groups and application to DKG. In *ACM CCS*. 4286–4300.

[50] Aniket Kate, Pratyay Mukherjee, Samipa Samanta, and Pratik Sarkar. 2025. DynahinTS: Silent Threshold Signatures for Dynamic Committees. *CVC* (2025).

[51] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*. Springer, 177–194.

[52] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures.. In *ACM CCS*. 1751–1767.

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

[54] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*. Springer, 129–140.

[55] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *EUROCRYPT*. Springer, 522–526.

[56] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[57] Victor Shoup. 2026. AHAB: Asynchronous, High-throughput, Adaptively-secure, Batched Threshold Schnorr Signatures. *Cryptology ePrint Archive* (2026).

[58] Victor Shoup and Nigel P Smart. 2024. Lightweight asynchronous verifiable secret sharing with optimal resilience. *Journal of Cryptology* 37, 3 (2024), 27.

[59] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2021. Synchronous distributed key generation without broadcasts. *Cryptology ePrint Archive* (2021).

[60] Web3Auth. [n. d.]. Web3Auth: Decentralized authentication for DApps. https://web3auth.io/.

[61] Xinxin Xing, Yizhong Liu, Boyang Liao, Jianwei Liu, Bin Hu, Xun Lin, Yuan Lu, and Tianwei Zhang. 2026. GoSSamer: Lightweight and Linear-Communication Asynchronous (Dynamic Proactive) Secret Sharing and the Applications. *IEEE S&P* (2026).

[62] Yunzhou Yan, Yu Xia, and Srinivas Devadas. 2022. Shanrang: Fully asynchronous proactive secret sharing with dynamic committees. *Cryptology ePrint Archive* (2022).

[63] Linghe Yang, Jian Liu, Jingyi Cui, Guangquan Xu, Mingzi Zuo, Lei Zhang, and Zhongshan Li. 2025. Robust and Scalable Lattice-Based Distributed Key Generation for Asynchronous Networks. *Cryptology ePrint Archive* (2025).

[64] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*. 347–356.

[65] Wenxuan Yu, Minghui Xu, Bing Wu, Sisi Duan, and Xiuzhen Cheng. 2025. ADMPC: Asynchronous Dynamic MPC with Guaranteed Output Delivery. In *ACM CCS*. 2892–2906.

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

[67] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. 2023. Long Live The Honey Badger: Robust Asynchronous {DPSS} and its Applications. In *USENIX Security*. 5413–5430.

[68] Haibin Zhang, Sisi Duan, Chao Liu, Boxin Zhao, Xuanji Meng, Shengli Liu, Yong Yu, Fangguo Zhang, and Liehuang Zhu. 2023. Practical asynchronous distributed key generation: Improved efficiency, weaker assumption, and standard model. In *DSN*. IEEE, 568–581.

[69] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. 2005. APSS: Proactive secret sharing in asynchronous systems. *ACM TISSEC* 8, 3 (2005), 259–286.

A Related Work

Distributed key generation for public-key cryptography usually considers discrete-logarithm settings, RSA modulus settings [13], or post-quantum settings [25, 28, 48, 63]. In this work, we mainly focus on the discrete-logarithm-based distributed key generation, where most constructions [1, 7, 17–20, 30, 31, 33, 38, 40, 43, 47, 49, 51, 53, 55, 59] assume an underlying synchronous network model and have made significant progress toward building efficient threshold cryptographic systems. To obtain robustness for arbitrary message delays under a Byzantine adversary, distributed key generation in the asynchronous network settings has recently attracted attention.

A.1 ADKG

KMS20 [52] introduces an ADKG construction that provides high threshold reconstruction with $O(\kappa n^4)$ communication under $O(n)$ rounds. ZDC+23 [68] improves the concrete efficiency of this design. Under the CRS setup, AJM+21 [4] and GLL+22 [36] reduce the cost to $O(\kappa n^3)$ communication and $O(1)$ rounds. DYX+22 [24] presents a low threshold variant and a high threshold variant that both remove the CRS setup and achieve $O(\kappa n^3)$ communication with $O(\log n)$ rounds. Bingo [3] introduces the first adaptively secure ADKG with $O(\kappa n^3)$ communication complexity, $O(\log n)$ round complexity, and high-threshold reconstruction under the CRS setup.

DXKR23 [22] presents a concretely efficient high-threshold ADKG construction with $O(\kappa n^3)$ communication and $O(\log n)$ rounds. Instead of relying on expensive high-threshold ACSS, it employs a more efficient low-threshold ACSS, introduces an interactive polynomial sampling technique that converts low-threshold Shamir shares into a high-threshold Shamir share, and derives public key shares from homomorphic commitments in ACSS using zero-knowledge proofs. Both DXKR23 [22] and Kairos adopt this overall approach based on low-threshold ACSS with a subsequent threshold lift and public key share derivation. However, Kairos develops a distinct threshold lifting method in which each node locally transforms low-threshold Shamir shares to an evaluation on the high-degree polynomial, without requiring the all-to-all interaction as in DXKR23 [22]. Moreover, our method for deriving public key shares utilizes a lightweight high-threshold verification aggregator, avoiding the usage of expensive homomorphic commitments.

Recently, ABLS25 [2] and FT25 [32] present two asymptotically efficient ADKG protocols with near quadratic communication. ABLS25 [2] achieves $O(\kappa^2 n^{2+1/k})$ communication and $O(k)$ rounds via a recursive procedure, where $k \in [\log n]$ specifies the depth of the recursion tree. FT25 [32] constructs a common coin protocol and transforms it to ADKG through the technique in FGL+25 [29] under a silent setup. It achieves $O(\kappa c n^2)$ communication complexity and $O(1)$ rounds, where c denotes the size of the any-trust committee.

A.2 Batched ADKG

Recent work has focused on the batched setting [5, 8, 41], where multiple threshold key pairs are generated in a single protocol execution, and the amortized generation cost is reduced by adopting batch-processing techniques. HARTS [8] and Haven++ [5] design two homomorphic-commitment-based high-threshold ACSS, and further construct high-threshold batched ADKG atop their ACSS scheme through zero-knowledge proof and super-invertible matrix. In the ADKG scenario where all nodes obtain the same public keys and all public key shares, both schemes achieve a total communication complexity of $O(\kappa n^2 \log n(b+n))$ and an amortized communication complexity of $O(\kappa n^2 \log n)$.

Detailed analysis of GS24. GS24 [41] presents a low-threshold batched ADKG from lightweight cryptography. The technique core in GS24 [41] is GoACSS, a group-oriented ACSS that is built upon the hash-based ACSS scheme SS24 [58]. In each GoACSS instance, the dealer invokes SS24 [58] for secret sharing, distributes the mappings of secret sharing polynomials in the group through reliable broadcast, and uses random linear combination and OSV to decide a successful GoACSS instance. A batched ADKG then follows from n parallel invocations of GoACSS and the super-invertible matrix.

GS24 instantiated with SS24 [58] admits a highly efficient optimistic path when no sharing instance enters recovery and only the aggregate public keys are produced, achieving output-optimal $O(\kappa n)$ amortized communication for this aggregate-key output functionality. We note that this linear bound is not exclusive to GS24: if only aggregate public keys are required, LT-Kairos can be extended to achieve the same asymptotic communication. After the private sharings are fixed, each node $\mathcal{P}_i$ can locally compute the group-valued evaluation $\mathbf{H}(i)\mathcal{G}$, reducing the remaining task to robustly reconstructing $\mathbf{H}(0)\mathcal{G}$ from these distributed evaluations. This task can be realized by adapting the bivariate-polynomial row-to-column conversion technique used in GoSSamer-ADPSS. For sufficiently large batches, this extension incurs $O(\kappa n)$ amortized communication per aggregate public key. In fact, the quadratic communication of LT-Kairos stems from producing and certifying the complete public-key-share vectors required by full-output ADKG, rather than from deriving the aggregate public keys.

In Table 1, we report GS24's worst-case communication complexity $O(\kappa b n^2 + \kappa n^4 \log n)$. Specifically, up to $f = \Theta(n)$ of the n underlying SS24 [58] instances may enter the recovery path, each incurring $O(\kappa L n^2 + \kappa n^3 \log n)$ communication, where L denotes the number of secrets shared in each instance. Since the final batch size satisfies $b = L(n-2f) = \Theta(Ln)$, the resulting total communication is $O(\kappa b n^2 + \kappa n^4 \log n)$. When GS24 is instead instantiated with GoSSamer-ACSS, whose per-instance communication is $O(\kappa L n + \kappa n^3 \log n)$, the straightforward full-output extension considered in our evaluation still incurs $O(\kappa b n^2 + \kappa n^4 \log n)$ communication for certifying the complete public-key-share vectors.

GS24 represents an important step toward efficient batched key generation from lightweight cryptography. It deliberately realizes a weaker biased-key-generation functionality that is sufficient for the threshold Schnorr preprocessing analyzed therein, rather than the standard full-secrecy functionality targeted by general-purpose ADKG. Applying its generated keys to other threshold cryptosystems would therefore require an additional, application-specific security analysis. LT-Kairos addresses this gap by supporting full secrecy, full output, while retaining the output-optimal amortized communication of $O(\kappa n^2)$ in the worst-case bound.

Moreover, to the best of our knowledge, prior work does not provide a mechanism for deriving and certifying high-threshold public key shares from low-threshold Shamir sharings produced by hash-based AVSS, without relying on homomorphic commitments. Kairos

addresses this gap, and the underlying threshold-lifting technique may be of independent interest beyond ADKG; for example, it yields generic transformations from low-threshold ACSS (or ADPSS) to high-threshold ACSS (or ADPSS).

A very recent work, AHAB [57], extends the GoAVSS line with a hiding-GoAVSS primitive for unbiased DKG and establishes adaptive security. It commits to and encrypts the group elements associated with dealers' sharings, and delays their opening until the dealer set has been irrevocably fixed. This elegant approach is tailored to the low-threshold signing setting, whereas Kairos focuses on batch-efficient derivation of general ADKG outputs and further supports high reconstruction thresholds. AHAB became available during the review process of this work, and thus we discuss it here.

B Throughput-latency Trade-off

We evaluate computational efficiency under batched operation in two regimes: (1) low-threshold ADKG constructions LT-Kairos and DYX+22 [24] with threshold $t = f$; (2) high-threshold ADKG constructions Kairos and DXKR23 [22] with threshold $t = 2f$. These experiments are performed locally on the machine with a 24-core Intel i9-12900KF CPU and 64GB of RAM, where a Bash script simulates the distributed environment and launches each node as a Python process executed on an independent CPU core. Figure 12 reports the results in the throughput-latency trade-off, where our solutions demonstrate stronger batch-processing performance. In the low-threshold setting, LT-Kairos generates 5.4-5.7× keys/sec compared to DYX+22 [24]. It reaches a peak throughput of 207 keys/s with a low latency of 15.4s at $n = 10$, and 36 keys/s with a latency of 175.5s at $n = 22$. In the high-threshold setting, Kairos produces 11.0-12.3× keys/sec compared to DXKR23 [22], and reaches a peak throughput of 171 keys/s at $n = 10$ and 33 keys/s at $n = 22$.

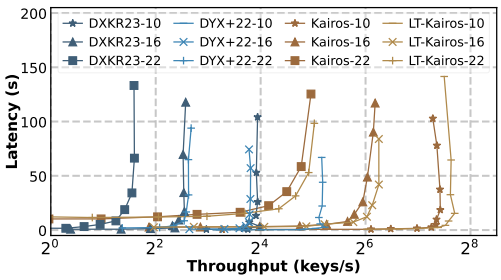


Figure 12: Evaluation of Throughput-latency trade-off

C Randomness Extraction Technique

In this section, we introduce a standard randomness extraction technique through super-invertible matrices, initially proposed in HN06 [44]. Let $\mathbf{M} \in \mathbb{Z}_p^{r\times c}$ be a matrix with r rows and $c \geq r$ columns. $\mathbf{M}$ is a super-invertible matrix if any collection of r of its columns is linearly independent. We instantiate the super-invertible matrix through the Vandermonde matrix, where each entry $\mathbf{M}_{i,j}$ at the i-th row and j-th column equals $\omega_j^{i-1} \in \mathbb{Z}_p$, with pairwise-distinct $\omega_1, \ldots, \omega_c$. Given inputs $\mathbf{x} = (x_1, \ldots, x_c)$ over $\mathbb{Z}_p$, define outputs $\mathbf{y} = \mathbf{Mx} = (y_1, \ldots, y_r)$. If at least r of the inputs are independent and uniformly random, then the outputs $y_1, \ldots, y_r$ are independent and uniformly random. Concretely, expanding $\mathbf{y} = \mathbf{Mx}$ gives

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \mathbf{M} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_c \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega_1 & \omega_2 & \cdots & \omega_c \\ \vdots & \vdots & \ddots & \vdots \\ \omega_1^{r-1} & \omega_2^{r-1} & \cdots & \omega_c^{r-1} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_c \end{bmatrix}$$

In Algorithm 2, lightweight randomness sharing, each node $\mathcal{P}_i$ holds Shamir's shares $\{[\mathbf{a}_u]_i\}_{u\in\mathsf{T}}$ for global consistent set T of size $n - f$, where each secret vector $\mathbf{a}_u$ is shared by an ACSS dealer $\mathcal{P}_u$. As at least $n - 2f$ dealers in set T are honest, there are at least $n - 2f$ vectors of independent and uniformly random secrets in $\{\mathbf{a}_u\}_{u\in\mathsf{T}}$. For simplicity, assume $|\mathbf{a}_u| = 1$, while the vector case applies element-wise. We now treat $\{\mathbf{a}_u\}_{u\in\mathsf{T}}$ as inputs to the extractor: set $c = n - f$ and $r = n - 2f$, and let vector $\mathbf{x}$ be any fixed ordering of these c inputs. Applying the super-invertible matrix $\mathbf{M} \in \mathbb{Z}_p^{r\times c}$ to $\mathbf{x}$ yields output $\mathbf{y} = \mathbf{Mx}$, which consists of $r = n - 2f$ independent and uniformly random values over $\mathbb{Z}_p$. By linearity of Shamir's secret sharing, each node computes its local shares of the outputs by multiplying matrix $\mathbf{M}$ with the vector encoded from its secret shares $\{[\mathbf{a}_u]_i\}_{u\in\mathsf{T}}$. This preserves the original privacy threshold and produces $n - 2f$ uniformly random shared outputs, which will be used in the subsequent protocol steps.

D Security Analysis

This section establishes correctness, secrecy, and termination for our high-threshold batched ADKG, Kairos (Algorithm 3). The low-threshold variant LT-Kairos (Algorithm 1) is the optimized boundary case $t = f$ of Kairos, and introduces no additional assumptions or proofs. Consequently, our analysis for $t \in [f, n - f)$ in Kairos already covers the low-threshold variant LT-Kairos.

Theorem 1 (Termination)**.** Kairos satisfies the termination property of asynchronous distributed key generation.

Proof. Assume that all honest nodes initiate Kairos. By the termination properties of ACSS and MVBA, all honest nodes eventually complete Algorithm 2 and obtain shares of the same extracted random values. They then locally compute their high-threshold private key shares and reliably broadcast their public key and public blinding shares. By reliable-broadcast completeness, the broadcasts initiated by all honest nodes eventually deliver. Consequently, every honest node eventually proposes a valid set of $n-f$ delivered broadcast instances to MVBA. By MVBA termination, agreement, and external validity, all honest nodes output the same set T of size $n-f$, and eventually construct the same locked set $\mathsf{T}_{\mathsf{lock}}$. After $\mathsf{T}_{\mathsf{lock}}$ is constructed, all honest nodes reveal their shares of d_0. Since there are $n-f$ honest evaluations of its f-degree sharing, online error-correction eventually reconstructs d_0. The honest nodes subsequently multicast correct evaluations of all partial scripts. For the same reason, online error-correction eventually reconstructs $v_0(x), \ldots, v_{h-1}(x)$, and hence the happy-path verification script $v(x)$. Let C be the set of corrupted nodes and define $c := |T \cap C|$. The locked set contains at least $|T \backslash C| = n-f-c = 2f+1-c$ honest nodes. Their tuples are valid and are therefore added to T_{pk}. If $|\mathsf{T}_{\mathsf{pk}}| \geq t+1$, the protocol terminates in the happy path.

Otherwise, the nodes execute Algorithm 4. The complement $[n]\backslash T$ contains exactly c honest nodes, since the committee contains $n-f$ honest nodes in total and T contains $n-f-c$ of them. For every honest node $\mathcal{P}_\ell \notin T$, its reliable-broadcast instance eventually delivers at every honest node. All honest nodes then execute the corresponding passive instance, reconstruct its hidden seed d_μ, reconstruct its partial scripts through the instance-specific online error-correction process, and accept the valid tuple $(\ell, \mathbf{pk}_\ell)$. Thus, the honest tuples obtained from the happy and passive paths together number at least $(n-f-c)+c = n-f \geq t+1$.

The protocol outputs as soon as $|\mathsf{T}_{\mathsf{pk}}| \geq t+1$ and does not wait for passive instances initiated by corrupted nodes. Hence, every honest node eventually obtains enough valid public-key shares, interpolates the public outputs, and terminates. □

Lemma 1. Against any static adversary corrupting at most f nodes, the outputs of Algorithm 2 are independent uniform f-degree Shamir sharings conditioned on the adversary's view.

Proof. The agreed set contains at least $f+1$ ACSS instances initiated by honest dealers. Their inputs are independent and uniform and remain hidden from the adversary by ACSS secrecy. Consider each extraction block and fix the inputs contributed by corrupted dealers. Since every $(f+1)\times(f+1)$ submatrix of the super-invertible extraction matrix is invertible, the inputs of any $f+1$ honest dealers are mapped bijectively to the $f+1$ extracted outputs. Hence, these outputs are independent and uniform even conditioned on the adversary's view and the corrupted dealers' inputs. The same linear transformation is applied locally to the corresponding shares. By the linearity of Shamir sharing, the resulting outputs remain valid f-degree Shamir sharings with privacy threshold f. □

Theorem 2 (Correctness)**.** Assuming that $p = 2^{\Theta(\kappa)}$, Kairos satisfies the correctness property of asynchronous distributed key generation except with negligible probability.

Proof. By the correctness and completeness of the underlying ACSS instances, the agreement and external-validity properties of MVBA, and the deterministic randomness-extraction procedure, there exist vectors of degree-at-most-f polynomials $\mathbf{h}_0(x), \ldots, \mathbf{h}_{h-1}(x)$ such that every honest node $\mathcal{P}_i$ holds the corresponding evaluations $\mathbf{h}_0(i), \ldots, \mathbf{h}_{h-1}(i)$. Define $\mathbf{F}(x) := \sum_{k=0}^{h-1} x^k \mathbf{h}_k(x) = \mathbf{H}(x) \parallel \mathbf{E}(x)$. Since $h = t-f+1$, every coordinate of $\mathbf{F}(x)$ has degree at most $f+h-1 = t$. Therefore, every honest node $\mathcal{P}_i$ holds the private key share $\mathbf{sk}_i = \mathbf{H}(i)$ and broadcasts the public key share $\mathbf{pk}_i = \mathbf{H}(i)G$ and public blinding share $\mathbf{e}_i = \mathbf{E}(i)G$.

Consider any verification instance $\mu \in [0, f]$. For the happy-path instance, $\mu = 0$, its statement is the locked set $\mathsf{T}_{\mathsf{lock}}$ and it uses d_0 and $\boldsymbol{\theta}_1$. For a passive instance associated with $\ell \in [n]\backslash T$, $\mu \in [1, f]$ is the position of ℓ in the sorted complement, its statement contains $(\mathsf{ins}_\ell, \mathbf{pk}_\ell, \mathbf{e}_\ell)$, and it uses d_μ and $\boldsymbol{\theta}_{\mu+1}$. In either case, the complete verification statement is irrevocably fixed before any honest share of the corresponding hidden seed is revealed. Let $\mathsf{Bad}_{\mathsf{seed}}$ be the event that the adversary queries a challenge-oracle point with the correct hidden seed before that seed is opened. By Lemma 1, every unopened seed remains uniform conditioned on the adversary's view. Therefore, if Q_H denotes the total number of such pre-opening oracle queries across all verification instances, $\Pr[\mathsf{Bad}_{\mathsf{seed}}] \leq Q_H/p$. Conditioned on $\neg\mathsf{Bad}_{\mathsf{seed}}$, every challenge $\mathbf{r}^{(\mu)}$ is uniform and independent.

For each $k \in [0, h)$, all honest nodes multicast evaluations of the degree-at-most-f polynomial $v_k^{(\mu)}(x) := (\mathbf{r}^{(\mu)} \parallel \boldsymbol{\theta}_{\mu+1}) \cdot \mathbf{h}_k(x)$. Since at least $n-f$ correct evaluations are eventually delivered, online error-correction reconstructs exactly these component polynomials at every honest node. Their aggregation therefore satisfies $v^{(\mu)}(x) = \left(\mathbf{r}^{(\mu)} \parallel \boldsymbol{\theta}_{\mu+1}\right) \cdot \mathbf{F}(x) = \mathbf{r}^{(\mu)} \cdot \mathbf{H}(x) + \boldsymbol{\theta}_{\mu+1} \cdot \mathbf{E}(x)$.

Now consider a tuple $(\widetilde{\mathbf{pk}}_\ell, \widetilde{\mathbf{e}}_\ell)$ broadcast by a corrupted node and fixed before $\mathbf{r}^{(\mu)}$ is revealed. Let $\boldsymbol{\Delta}_{\mathsf{pk}} \in \mathbb{Z}_p^b$ and $\Delta_{\mathsf{e}} \in \mathbb{Z}_p$ be defined by $\widetilde{\mathbf{pk}}_\ell = (\mathbf{H}(\ell) + \boldsymbol{\Delta}_{\mathsf{pk}})\mathcal{G}$ and $\boldsymbol{\theta}_{\mu+1} \cdot \widetilde{\mathbf{e}}_\ell = (\boldsymbol{\theta}_{\mu+1} \cdot \mathbf{E}(\ell) + \Delta_{\mathsf{e}})\mathcal{G}$. The tuple passes verification only if $\mathbf{r}^{(\mu)} \cdot \boldsymbol{\Delta}_{\mathsf{pk}} + \Delta_{\mathsf{e}} = 0$.

If $\boldsymbol{\Delta}_{\mathsf{pk}} = \mathbf{0}$, then the public-key share is already correct. Otherwise, since $\mathbf{r}^{(\mu)}$ is uniform in $\mathbb{Z}_p^b$, the above nontrivial linear equation holds with probability exactly $1/p$. Thus, conditioned on $\neg\mathsf{Bad}_{\mathsf{seed}}$, an incorrect public-key share broadcast by a corrupted node is accepted with probability at most $1/p$.

Each node contributes only one broadcast tuple, and the happy and passive paths verify disjoint sets of nodes. Since at most f tuples originate from corrupted nodes, a union bound gives $\Pr[\mathsf{Bad}_{\mathsf{corr}}] \leq (Q_H + f)/p$, where $\mathsf{Bad}_{\mathsf{corr}}$ denotes the event that an incorrect public-key share is accepted in any verification instance.

Conditioned on $\neg\mathsf{Bad}_{\mathsf{corr}}$, every element $(\ell, \mathbf{pk}_\ell)$ added to T_{pk} satisfies $\mathbf{pk}_\ell = \mathbf{H}(\ell)\mathcal{G}$. Hence, any $t+1$ elements of T_{pk} uniquely interpolate the same degree-at-most-t group-polynomial vector $\mathbf{H}(x)\mathcal{G}$. All honest nodes therefore output $\mathbf{pk} = \mathbf{H}(0)\mathcal{G}$ and $\mathbf{pk}_j = \mathbf{H}(j)\mathcal{G}$ for every $j \in [n]$. These public-key shares are consistent with the private-key shares $\mathbf{sk}_i = \mathbf{H}(i)$ held by every honest node. Since $(Q_H + f)/p$ is negligible in κ, Kairos satisfies correctness. □

Lemma 2. Let $h_0(x), \ldots, h_{h-1}(x)$ be independent uniform polynomials of degree at most f, where $h = t-f+1$. Define $H(x) := \sum_{k=0}^{h-1} x^k h_k(x)$. Conditioned on the evaluations of all $h_k(x)$ held by the corrupted nodes, $H(x)$ is uniformly distributed over all degree-at-most-t polynomials consistent with the corrupted view.

Proof. Without loss of generality, assume that the adversary corrupts exactly f nodes, denoted by C, and let $L(x) := \prod_{i \in C}(x-i)$. For each k, let $\bar{h}_k(x)$ be the unique polynomial of degree at most $f-1$ interpolating $\{h_k(i)\}_{i \in C}$. Conditioned on these evaluations, every $h_k(x)$ can be uniquely written as $h_k(x) = \bar{h}_k(x) + z_k L(x)$, where the z_k's are independent and uniform in $\mathbb{Z}_p$. Let $\bar{H}(x) := \sum_{k=0}^{h-1} x^k \bar{h}_k(x)$ and $Z(x) := \sum_{k=0}^{h-1} z_k x^k$, there is $H(x) = \bar{H}(x) + L(x)Z(x)$. Since $h = t-f+1$, the coefficients of $Z(x)$ are independent and uniform, and hence $Z(x)$ is uniform over all polynomials of degree at most $t-f$. Moreover, every degree-at-most-t polynomial consistent with the corrupted evaluations has a unique representation $\bar{H}(x) + L(x)Z(x)$ with $\deg Z \leq t-f$. Therefore, $H(x)$ is uniform over this set. □

Theorem 3 (Secrecy)**.** Kairos satisfies the secrecy property of asynchronous distributed key generation.

Proof. We employ a hybrid argument to prove that for any P.P.T static adversary $\mathcal{A}$ that corrupts up to f nodes, there exists a P.P.T simulator that can produce a simulated view $\mathsf{view}_{\mathsf{sim}}$ given only the uniformly sampled $\mathbf{e}\mathcal{G}$ in group $\mathbb{G}^b$, such that $(\mathsf{view}_{\mathsf{sim}}, \mathbf{e}\mathcal{G})$ is computationally indistinguishable from $(\mathsf{view}_{\mathsf{real}}, \mathbf{pk})$, where $\mathsf{view}_{\mathsf{real}}$ is

the view of $\mathcal{A}$ in the real execution with $\mathbf{pk}$ as the output public keys. The simulator $\mathcal{S}_{\text{Kairos}}$ is described in Figure 13. Without loss of generality, assume that exactly f nodes are corrupted.

Hybrid 0. This is a real-world execution of Kairos, where all honest nodes adhere to the protocol.

Hybrid 1. In this hybrid, we program the challenge oracle for every verification instance $\mu \in [0, f]$. For each μ, the simulator samples the challenge $\mathbf{r}^{(\mu)} \leftarrow \mathbb{Z}_p^b$ in advance. Once the corresponding verification statement is irrevocably fixed, but before revealing any honest share of the hidden seed d_μ, it programs the corresponding challenge-oracle point to $\mathbf{r}^{(\mu)}$. The simulator aborts if this point has already been queried. Before any honest share of d_μ is revealed, the adversary holds at most f evaluations of its f-degree sharing. By Lemma 1, d_μ remains uniform conditioned on the adversary's view. If the adversary makes at most Q_H such pre-opening challenge-oracle queries in total across all verification instances, the probability that it queries any programmed point in advance is at most Q_H/p. Conditioned on no abort, the two hybrids are identically distributed. Therefore, we get

$$\left|\Pr[\mathbf{Hybrid}_1 \Rightarrow 1] - \Pr[\mathbf{Hybrid}_0 \Rightarrow 1]\right| \leq \frac{Q_H}{p}.$$

Hybrid 2. In this hybrid, we reverse the generation order by first sampling the partial scripts and then deriving the public blinding polynomials. Write $\mathbf{h}_k^*(x) = (\mathbf{a}_k^*(x), \beta_{k,1}^*(x), \ldots, \beta_{k,f+1}^*(x))$, where $\mathbf{a}_k^*(x)$ contains the key coordinates and $\beta_{k,\mu+1}^*(x)$ is the blinding coordinate assigned to instance μ. For every $\mu \in [0, f]$ and $k \in [0, h)$, sample a uniform degree-f polynomial $v_k^{(\mu)}(x)$ which satisfies $v_k^{(\mu)}(i) = (\mathbf{r}^{(\mu)} \parallel \boldsymbol{\theta}_{\mu+1}) \cdot \mathbf{h}_k^*(i)$ for every $i \in C$. Let $v^{(\mu)}(x) := \sum_{k=0}^{h-1} x^k v_k^{(\mu)}(x)$ and define the $(\mu{+}1)$-th entry in $\widehat{\mathbf{E}}(x)\mathcal{G}$ as $\widehat{E}_{\mu+1}(x)G := v^{(\mu)}(x)\mathcal{G} - \mathbf{r}^{(\mu)} \cdot \mathbf{H}^*(x)\mathcal{G}$. In the real execution, each partial script is obtained by adding the independent uniform blinding polynomial $\beta_{k,\mu+1}^*(x)$ to the fixed polynomial $\mathbf{r}^{(\mu)} \cdot \mathbf{a}_k^*(x)$. Conditioned on the corrupted evaluations, this addition is an affine bijection onto exactly the space from which $v_k^{(\mu)}(x)$ is sampled above. Since the blinding coordinates are independent across all k and μ, the joint distribution of the partial scripts is unchanged. The definition of $\widehat{E}_{\mu+1}(x)\mathcal{G}$ also preserves its evaluation at every corrupted node. Moreover, conditioned on the corrupted evaluations and the challenges, the resulting partial scripts are jointly uniform and independent of the remaining coefficients of $\mathbf{H}^*(x)$. Therefore

$$\Pr[\mathbf{Hybrid}_2 \Rightarrow 1] = \Pr[\mathbf{Hybrid}_1 \Rightarrow 1].$$

Hybrid 3. In this hybrid, we replace $\mathbf{H}^*(x)\mathcal{G}$ by a degree-at-most-t polynomial vector $\widehat{\mathbf{H}}(x)\mathcal{G}$ sampled uniformly subject to $\widehat{\mathbf{H}}(0)\mathcal{G} = \mathbf{e}\mathcal{G}$ and $\widehat{\mathbf{H}}(i)\mathcal{G} = \mathbf{H}^*(i)\mathcal{G}$ for every $i \in C$. For every instance μ, redefine $\widehat{E}_{\mu+1}(x)\mathcal{G} := v^{(\mu)}(x)\mathcal{G} - \mathbf{r}^{(\mu)} \cdot \widehat{\mathbf{H}}(x)\mathcal{G}$. By Lemma 2, conditioned on the corrupted low-threshold evaluations, each coordinate of $\mathbf{H}^*(x)$ is uniform over all degree-at-most-t polynomials consistent with the corrupted view. In particular, $\mathbf{H}^*(0)$ is uniform conditioned on that view. Therefore, conditioned further on $\widehat{\mathbf{H}}(0)\mathcal{G} = \mathbf{e}\mathcal{G}$, the group-polynomial vector $\mathbf{H}^*(x)\mathcal{G}$ has exactly the distribution used to sample $\widehat{\mathbf{H}}(x)\mathcal{G}$. The partial scripts remain unchanged, while the public blinding polynomials are derived deterministically from the sampled group-polynomial vector, the challenges, and the partial scripts. They also retain their evaluations at every corrupted node. Hence, the complete joint distribution is unchanged, and

$$\Pr[\mathbf{Hybrid}_3 \Rightarrow 1] = \Pr[\mathbf{Hybrid}_2 \Rightarrow 1].$$

Hybrid 3 is exactly the simulated execution generated by $\mathcal{S}_{\text{Kairos}}$ on input $\mathbf{e}\mathcal{G}$. Combining the above transitions, we obtain

$$\left|\Pr[\mathbf{Hybrid}_0 \Rightarrow 1] - \Pr[\mathbf{Hybrid}_3 \Rightarrow 1]\right| \leq \frac{Q_H}{p}.$$

Since Q_H is polynomially bounded and p is exponential in the security parameter, $(\text{view}_{\text{sim}}, \mathbf{e}\mathcal{G})$ is computationally indistinguishable from $(\text{view}_{\text{real}}, \mathbf{pk})$ and Kairos satisfies full secrecy. □

Simulator $\mathcal{S}_{\text{Kairos}}$

input: $\mathbb{Z}_p$, n, f (malicious node tolerance), $t \in [f, n-f)$ (reconstruction threshold), C (corrupted nodes set), b (batch size), $\mathcal{G}$ (group generator), and $\mathbf{e}\mathcal{G}$ (ideal public keys, uniformly sampled vector of group elements)
notation: $\mathcal{H} = [n]\backslash C$ (honest nodes set), $h = t{-}f{+}1$, and $\boldsymbol{\theta}_k$ (the k-th standard basis vector with 1 at coordinate k, 0 elsewhere. $|\boldsymbol{\theta}_k| = f{+}1$)

1: Simulate the randomness-sharing phase and obtain the seed polynomials $d_0^*(x), \ldots, d_f^*(x)$ and polynomials $\mathbf{h}_0^*(x), \ldots, \mathbf{h}_{h-1}^*(x)$ of degree f
2: Construct $(b{+}f{+}1)$-dimension vector of t-degree polynomials $\mathbf{F}^*(x)$ with $\mathbf{F}^*(x) = \mathbf{h}_0^*(x) + \mathbf{h}_1^*(x)x + \cdots + \mathbf{h}_{h-1}^*(x)x^{h-1}$, whose first b coordinates are $\mathbf{H}^*(x)$ and last $f{+}1$ coordinates are $\mathbf{E}^*(x)$
3: Sample degree-at-most-t group polynomials $\widehat{\mathbf{H}}(x)\mathcal{G}$ uniformly subject to $\widehat{\mathbf{H}}(0)\mathcal{G} = \mathbf{e}\mathcal{G}$ and $\widehat{\mathbf{H}}(i)\mathcal{G} = \mathbf{H}^*(i)\mathcal{G}$ for every $i \in C$
4: For every verification instance $\mu \in [0, f]$, where $\mu = 0$ is the happy path, calculate the $(\mu+1)$-th entry in $\widehat{\mathbf{E}}(x)\mathcal{G}$ as follows:
- independently sample challenge $\mathbf{r}^{(\mu)} \leftarrow \mathbb{Z}_p^b$
- sample f-degree $v_k^{(\mu)}(x)$ with $v_k^{(\mu)}(i) = (\mathbf{r}^{(\mu)} \parallel \boldsymbol{\theta}_{\mu+1}) \cdot \mathbf{h}_k^*(i)$ for $k \in [0, h), i \in C$, calculate $v^{(\mu)}(x) = v_0^{(\mu)}(x) + xv_1^{(\mu)}(x) + \cdots + x^{h-1}v_{h-1}^{(\mu)}(x)$
- calculate the $(\mu+1)$-th entry in $\widehat{\mathbf{E}}(x)\mathcal{G}$ as $v^{(\mu)}(x)\mathcal{G} - \mathbf{r}^{(\mu)} \cdot \widehat{\mathbf{H}}(x)\mathcal{G}$

5: Reliable broadcast $(\mathbf{pk}_i, \mathbf{e}_i) = (\widehat{\mathbf{H}}(i)\mathcal{G}, \widehat{\mathbf{E}}(i)\mathcal{G})$ as every honest node $\mathcal{P}_i$
6: Upon constructing the locked set T_{lock}
- program the random oracle to output $\mathbf{r}^{(0)}$ on input $(\text{sid}, \mathsf{T}_{\text{lock}}, d_0^*(0))$
- multicast $d_0^*(i)$ on behalf of every honest node $\mathcal{P}_i$
- after reconstructing $d_0 = d_0^*(0)$, multicast $v_0^{(0)}(i), \ldots, v_{h-1}^{(0)}(i)$ on behalf of every honest node $\mathcal{P}_i$

7: For the passive instance associated with the μ-th index ℓ in the sorted set $[n]\backslash\mathsf{T}$, upon delivering $(\mathbf{pk}_\ell, \mathbf{e}_\ell)$ from the broadcast initiated by $\mathcal{P}_\ell$:
- program the oracle to output $\mathbf{r}^{(\mu)}$ on input $(\text{sid}, \text{ins}_\ell, d_\mu^*(0), \mathbf{pk}_\ell, \mathbf{e}_\ell)$
- multicast $(\text{ins}_\ell, d_\mu^*(i))$ on behalf of every honest node $\mathcal{P}_i$
- after reconstructing $d_\mu = d_\mu^*(0)$, multicast $(\text{ins}_\ell, v_0^{(\mu)}(i), \ldots, v_{h-1}^{(\mu)}(i))$ on behalf of every honest node $\mathcal{P}_i$

8: Simulate the remaining protocol messages honestly.

Figure 13: The simulator of Kairos

E Communication Complexity

E.1 Analysis of LT-Kairos

Theorem 4. Consider a committee of n nodes with threshold $t = f$, let the batch size be $b = \text{poly}(n)$. LT-Kairos implements an (n, t, f, b)-ADKG with a total communication of $O(\kappa bn^2 + \kappa n^4 \log n)$ and an amortized communication of $O(\kappa n^2)$, when instantiated with the hash-based ACSS schemes with a per-secret communication of up to $O(\kappa n^2)$, e.g., GoSSamer [61] or SS24 [58].

Proof. Let $O_{\text{acss}}(x)$ denote the communication complexity of sharing x secrets through ACSS, $O_{\text{rbc}}(x)$ denote the communication complexity of reliably broadcasting a message of length x, and O_{mvba} denote the communication complexity of MVBA. As discussed in Section 3.1, $O_{\text{rbc}}(x) = O(nx + \kappa n^2)$, and $O_{\text{mvba}} = O(\kappa n^3)$. Thus, the expected communication complexity of LT-Kairos is:

$$\begin{aligned} &nO_{\text{acss}}(b/n) + nO_{\text{rbc}}(b) + O_{\text{mvba}} + O(\kappa n^2) \\ &= nO_{\text{acss}}(b/n) + O(\kappa bn^2 + \kappa n^3) \end{aligned}$$

Suppose the ACSS scheme incurs a per-secret communication of up to $O(\kappa n^2)$, the amortized communication of LT-Kairos is $O(\kappa n^2)$.

Instantiation. Consider the hash-based ACSS schemes SS24 [58] and GoSSamer [61], which has a total communication of $O_{\text{acss}}(x) = O(\kappa xn^2 + \kappa n^3 \log n)$ and $O_{\text{acss}}(x) = O(\kappa xn + \kappa n^3 \log n)$, respectively. Plugging either scheme into LT-Kairos yields total communication $O(\kappa bn^2 + \kappa n^4 \log n)$. When the batch size satisfies $b = \Omega(n^2 \log n)$, the amortized communication of LT-Kairos is $O(\kappa n^2)$. □

E.2 Analysis of Kairos

Theorem 5. Consider a committee of n nodes with threshold $t \in [f, n-f)$, let the batch size be $b = \text{poly}(n)$. Kairos implements an (n, t, f, b)-ADKG with a total communication of $O(\kappa bn^2 + \kappa n^4 \log n)$, and an amortized communication of $O(\kappa n^2)$, when instantiated with hash-based ACSS with a per-secret communication of $O(\kappa n)$, e.g., GoSSamer-ACSS [61] or the happy path of SS24 [58].

Proof. The expected communication in the happy path is:

$$nO_{\text{acss}}(b + n) + O_{\text{mvba}} + O(\kappa n^3) + nO_{\text{rbc}}(\kappa b + \kappa n)$$

As the expected communication of the constant-round passive path is $O(\kappa n^4)$, the total communication in the worst case is given by:

$$nO_{\text{acss}}(b + n) + O(\kappa bn^2) + O(\kappa n^4)$$

Suppose that the ACSS scheme incurs a per-secret communication of $O(\kappa n)$, then the amortized communication of Kairos is $O(\kappa n^2)$.

Instantiation. GoSSamer [61] and the happy path of SS24 [58] achieve the same communication of $O_{\text{acss}}(x) = O(\kappa xn + \kappa n^3 \log n)$. Plugging either scheme into Kairos yields total communication $O(\kappa bn^2 + \kappa n^4 \log n)$. When the batch size satisfies $b = \Omega(n^2 \log n)$, the amortized communication of Kairos is $O(\kappa n^2)$. □