Dyna-hinTS: Silent Threshold Signatures for Dynamic Committees

Aniket Kate
Purdue University, Supra Research
aniket@purdue.edu

Pratyay Mukherjee
Supra Research
pratyay85@gmail.com

Samipa Samanta
Indian Institute of Technology Bombay
samipasamanta1998@gmail.com

Pratik Sarkar
Supra Research
iampratiksarkar@gmail.com

***Abstract*—The works of Garg et al. [S&P'24] (aka hinTS) and Das et al. [CCS'23] introduced the notion of silent threshold signatures (STS) - where a set of signers silently perform local computation to generate a public verification key. To sign a message, any set of t signers sign the message non-interactively and these are aggregated into a constant-sized signature. This paradigm avoids performing expensive Distributed Key Generation procedure for each set of signers while keeping the public verification key constant-sized.**

In this work, we propose the notion of committee-based silent threshold signature (c-STS) scheme. In a c-STS scheme, a set of signers initially perform a one-time setup to generate the verification key, and then a subset of signers are randomly chosen for an epoch to perform the threshold signing while the other signers are not authorized to sign during that epoch. This captures existing systems like Ethereum Altair and Dfinity where only a specific committee is authorized to sign in a designated epoch. The existing STS schemes cannot be extended to the committee setting because the signature verification only attests to the number of signing parties, not which committee they belong to.

So, we upgrade hinTS to the committee setting by proposing Dyna-hinTS. It is the *first* c-STS scheme and it requires a one-time silent setup and generates a one-time public verification key that does not vary with the committee. Assuming a set of 1024 signers (with corrupt 682 signers), hinTS generates an aggregated signature in 1.7s whereas Dyna-hinTS generates it in 0.35s within a committee of 80 signers. This yields a $4.9\times$ improvement over hinTS for signature generation at the cost of increasing signature verification time by 4% over hinTS. Dyna-hinTS supports general access structure, weighted signatures and improves existing multiverse threshold signatures.

1. Introduction

Threshold signatures [33, 34] allows a set of N signers to sign a message such that any $t \leq N$ signers are necessary and sufficient to create a valid signature for the set. Due to the fundamentally decentralized nature they have gained significant traction recently in the blockchain setting for applications such as cross-chain bridges [2, 63, 65, 68, 61], threshold wallets [6, 3], state proofs [35, 25], multiparty credential transactions [7], distributed randomness [5, 1], optimizing Byazantine consensus protocols [45, 69] and many more. Indeed, given their vast application space, NIST initiated the standardization [23] of threshold signatures in early 2023.

In earlier application efforts, threshold signatures have been widely employed [55, 5, 1] in settings with a fixed set of N signers and where any t signers can sign the message. Security holds as long as adversary corrupts $f < t$ parties. In this setting, the signers need to perform distributed key generation (DKG) [46] to set up their shared secret keys (in a Shamir-secret sharing fashion) and a common public verification key VK. To sign a message m, any t signers generate partial signatures using their individual secret key shares, and these partial signatures are aggregated into a single signature. A higher f is essential to protect against strong adversary, but that also necessitates a larger $t > f$, resulting into higher cost in the aggregation (which scales with t).

Dynamic Committee-based Signatures: In practice large-scale blockchain systems [8, 49, 41] are increasingly adopting the idea of randomly selecting rather short-lived committees. This evades the dependency $t > f$ and yields protocols with enhanced efficiency without sacrificing security. In this setting, among N signers in the system, a smaller committee Com of $n \ll N$ signers are selected at random (using, for example, a trusted source of randomness, such as randomness beacon service [5]) as the valid set of signers for each predefined epoch. To sign messages in this epoch, any $t \leq n$ parties from Com generate the signature. Now, an adversary that corrupts $f < N$ parties needs to control at least t parties *within the committee* to forge a signature. Nevertheless, as long as the adverdsary's choice is independent of the randomness used to choose the committee[1] this event occurs with negligible probability. In effect, the aggregator's computation (proportional to t) becomes significantly more efficient, as a much smaller $t \leq n$ can be afforded, while security holds against much higher $f \gg t$.

In the committee setting, since the set of signers change more frequently, a fresh DKG protocol is required to run at every epoch (whenever a new committee is chosen). As

1. This is justified as long as the randomness is generated via a randomness beacon (or similar procedure), that has pseudorandom output.

DKGs [46, 51] are more expensive, this incurs substantial overhead. To avoid DKGs in threshold signatures, Ethereum Altair uses the BLS multi-signature scheme [22] in the committee setting [2]. In that each validator node within a committee uses its independent secret key to generate signatures, that are aggregated – the aggregated signature is then checked with respect to the aggregated verification key, computed from the individual verification keys of the same committee of validators. Crucially, it necessitates storing all individual public keys (whereas threshold signatures need to store only a single public key). Storing data on the blockchain incurs gas cost. In particular, Garg et al. [44] showed that storing 1024 BLS public keys consumes 60 million EVM gas, which is roughly $1200 in October 2024.

To resolve this DKG vs large-storage dilemma, two recent works [44, 30] introduced the idea of *silent threshold signatures* (STS), in that the signers sample their keys independently (without DKG), similar to multi-signatures, yet similar to threshold signature, storing a short $O(1)$-sized aggregated public key suffices for verification. Unfortunately, the notion is not suitable in the dynamic committee setting. This is because STS guarantees that, if an aggregated signature verifies correctly, then any $\geq t$ out of all N parties signed correctly. This does not convey whether the signers belong to a specific subset/committee. Therefore, any corruption of $f \geq t$ parties leads to forgery – in absence of no committee-specific verifiability, a subset of t corrupt signers is able to produce valid signatures. This leads to a natural question:

Can we construct an efficient threshold signature (with a short verification key) that supports dynamic committees without requiring a per-committee setup?

We answer the above question in the affirmative by building upon the works of [44, 30].

1.1. Our Contributions

Committee-based STS: We propose the new notion of committee-based silent threshold signature (c-STS) by extending the idea of STS to support dynamic committees. In c-STS, each party in the universe of all N signers generates their individual secret and public key independently and then performs a one-time non-interactive pre-processing to compute a short ($O(1)$-size) aggregated public verification key VK that is posted on the blockchain. At every predefined epoch e, a set of n signers are chosen (pseudo-)randomly (from the N signers) as the signing committee Com. To sign a message m in the epoch e, any $t \leqslant n$ signers *within* Com non-interactively produce an aggregated short signature σ that verifies w.r.t. verification key VK, committee Com and epoch e. The security of c-STS guarantees that an adversary (that initially/statically corrupts $f = \mathcal{O}(N)$ of the signers)cannot forge a signature on a message m in an epoch e. This is ensured with overwhelming probability since

1) the adversary fails to corrupt t signers from any signing committee (since the committee members would be sampled randomly), and
2) the adversary cannot forge the signatures of honest signers from the committee.

Dyna-hinTS: We propose Dyna-hinTS, the *first* committee-based silent threshold signatures, based on BLS signatures [22] and the Plonk proof system [43] in the common reference string (crs) model. On a high level, we extend the STS construction of [44, 30] to the committee setting without incurring substantial verification cost. We follow the notation and exposition of [44] (aka hinTS)[3] – hence the name Dyna-hinTS is chosen. The setting is similar to hinTS, except that we use a random beacon to select a committee of signers in each epoch and commit to it. During the signing phase, Dyna-hinTS follows the hinTS protocol. In addition, we reuse the KZG commitment to the signers, used in the SNARK proof of the same, to generate proof that the set of signers belongs to the committee. This additional proof adds an insignificant overhead of $0.77ms$ (or 4%) on the verifier side while augmenting hinTS to support dynamic committees. Upgrading hinTS to the committee setting allows Dyna-hinTS to reduce the reconstruction threshold by $8.5\times$ when $2/3$rds of the signers are corrupt while guaranteeing the same level of security. This yields a concrete improvement of $4.9\times$ in signature generation time - which includes partial signature generation and aggregation.

Additionally, we propose a generic technique to reduce the one-time setup cost drastically – this applies not only to Dyna-hinTS, but to existing STS [44, 30] as well. The setup cost of prior STS protocols [44, 30] scale quadratically with the number of signers. We observe that the computation of the terms, that contribute to the quadratic cost, does not involve any secret information and can be delegated to untrusted parties. Once these untrusted parties respond their output can be efficiently verified. This results in a significant $30\times$ improvement of the setup computational cost. Our optimized setup takes 66 seconds to generate the individual keys and hinTS by each signer, verify the hinTS of all signers and then compute the public verification key.

Empirical Comparison: We implement Dyna-hinTS in Rust and compare it against hinTS and BLS multi-signatures. We run all three protocols with $N = 1024$ signers. We consider multiple corruption thresholds where the adversary can corrupt up to $f = 79, \frac{N}{3}, \frac{N}{2}$ and $\frac{2N}{3}$ signers. In each of these settings, a committee size of size $n = 128, 128, 95, 82$ (following a hypergeometric distribution) is chosen respectively using a random beacon and Dyna-hinTS only requires $t = 80$ signers to generate a signature for all four setting. Whereas, BLS multi-signature and hinTS always require $t = f + 1$ parties to sign a message. This yields a concrete improvement of $0.09\times$,

2. In the Ethereum Altair update, a set of 512 validator nodes is randomly [10] chosen as the *sync committee* [4] for signing every 27 hours from a set of around million among the Ethereum validator nodes [9].

3. We remark that, the constructions of Garg et al. [44] and Das et al. [30] are essentially same with some minor differences. Except when specifics are mentioned, by hinTS we mean either (or both) of these constructions.

$2.6\times$, $4\times$ and $4.9\times$ respectively for the different values of f. Our signing protocol is the same as BLS signatures and each signer takes roughly 1.5 ms (regardless of the signer's weight) to generate the partial signature. The aggregator takes $357ms$ to aggregate $t = 80$ partial signatures. Verifying the signature takes 15.77 ms and costs 501K (or) gas on-chain. In comparison, verifying a hinTS signature costs 15.06 ms and consumes 441K gas.

Extension: Similar to hinTS and [30], Dyna-hinTS also supports weighted signatures and works over a more general access structure among the signers, now with support for dynamic committees. It also enables dynamic thresholds - i.e. the threshold t can vary for different messages, even if they are signed by the same committee. It allows signers to dynamically join/leave the signing set. In such a case the public verification key can be updated without any communication/computation among the existing signers. Finally, we show that Dyna-hinTS supports multiverse threshold signatures (MTS) [16] with minimal changes and can be used to remove the universe setup phase in MTS schemes. We discuss these extensions in Appendix H.

1.2. Related Works

We provide a comprehensive comparison of Dyna-hinTS with other threshold signing protocols in Tab. 1 and we elaborate further next.

DKG-based Signatures: The most popular paradigm of threshold signatures involves sharing the secret key among the signers using a DKG protocol [46]. There has been enormous progress in this paradigm due to constant improvements to the DKG protocol in terms of communication [40, 17, 47], computation [66, 40], round complexity [48, 52, 47, 20], and many works have explored constructions assuming asynchronous [12, 32, 54] networks. As discussed earlier, DKGs are inherently problematic for committee setting, as per committee DKG is just not scalable in reality.

Multi-signatures: In this approach [22, 59, 60], the signers post their public keys on the blockchain or a public bulletin board – this is a one time silent setup. To sign a message, t participants need to provide their partial signatures w.r.t. to an aggregated public key that is computed from the signers' public keys. This paradigm can be extended to yield committee-based threshold signatures due to one time silent setup. However, there are two limitations in this extension. Firstly, the size of the signature grows linearly with the number of signers[4] and secondly, it requires storing the signers' public keys on-chain. The first limitation was addressed by the compressed Σ protocol works of [15, 14] that use bulletproofs [24] to prove that the aggregated public key correctly generated and the signature size grows logarithmic with the number of signers at the cost of increasing the verification time to be linear in the number of signers. However, they still incur the on-chain storage overhead as BLS multi-signatures. The second limitation is a major downside since it incurs a lot of gas to store public keys on-chain. For example, the hinTS paper [44] pointed out that it costs approximately \$1200 (equivalently 60 million EVM) to store 1024 BLS public keys on-chain. The work of compact certificates [58] constructed threshold signatures with constant public key size at the cost of making the verification time linear in the number of signers and logarithmic-sized signature, increasing the signature size by an order of magnitude.

Silent Threshold Signatures: The work of [44, 30] addressed the storage cost limitation by introducing silent threshold signatures where the public keys are stored on-chain in a succinct fashion using a KZG commitment [53]. From high level, the construction generates an efficient SNARK proof of correct computation of the aggregated public key. As a result, the signature size and the verifier runtime are independent of the number of signers. However, the proof only attests to the fact that there are at least t signers. Hence, the hinTS (and also [30]) can tolerate $f < t$ corruptions among the signing set. Whereas, Dyna-hinTS tolerates a much larger fraction of signer corruptions while keeping the reconstruction threshold low. This occurs since we dynamically switch the active signing committee using a randomness beacon. Concretely, consider $N = 1024$ signers and assume that there are $f = \frac{2N}{3} = 683$ corrupt signers. Then hinTS requires 684 signers to sign a message whereas Dyna-hinTS requires only 80 signers to guarantee the same level of security. This yields a $8.5\times$ reduction in the number of signing parties of Dyna-hinTS and $4.9\times$ improvement in signature aggregation time over hinTS. The improvements would be even better when to [30] since [30] fares worse compared to hinTS. We note that in Dyna-hinTS there is probability, that the corruption set contains more than t parties for a given committee. This probability is zero for hinTS. However, due to random selection of committee, Dyna-hinTS's design ensures that the probability (which follows a hypergeometric distribution) stays small enough – our choice of parameters ensure this probability is fixed to 2^{-40}. This is a standard statistical error which is permitted by other works like [41].

We compare with Multiverse Threshold Signatures (MTS), Threshold ECDSA and Schnorr Signatures, and accountable threshold signatures in Appendix.A due to lack of space in the main body. We also discuss how Dyna-hinTS yields an improved MTS over the state-of-the-art MTS scheme of [16].

2. Preliminaries

We denote the computational and statistical security parameters by κ and μ. $\mathsf{negl}(\kappa)$ as the negligible function in κ. We denote $[N]$ as the set $\{1, 2, \ldots, N\}$. $\mathbb{F}$ is a finite field of order p and $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of order p with a non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. g_1 , g_2 are uniformly sampled generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. $g_T = e(\mathbb{G}_1, \mathbb{G}_2)$. For a given $a \in \mathbb{F}$, we denote g_1^a and g_2^a over groups $\mathbb{G}_1$ and $\mathbb{G}_2$ as $[a]_1$ and $[a]_2$ respectively. $\mathbb{H}$ is a multiplicative subgroup of $\mathbb{F}$. $\mathbb{H} = \{\omega, \omega^2, ..\omega^{|\mathbb{H}|} = 1\}$

4. This happens even when the partial signatures are aggregated, yet it should be accompanied by the bit-vector denoting the set of signers.

Protocols	Supports Dynamic Committees?	Aggregated Signature Size (Bytes)	Verification Time	Verification Key Size (bits)	Setup Type
Threshold ECDSA/Schnorr[37, 56, 26]	×	64	2 $\mathbb{G}_1$ mul	$\mathcal{O}(\kappa)$	DKG (Each epoch)
Schnorr Multisignatures [59, 60]	✓	$\mathcal{O}(\kappa) + \min(N, n\log_2 N)$	$n\mathbb{G}_1$ add, $2P$	$\mathcal{O}(N\kappa)$	Silent setup (One time)
BLS Multisignatures [22]	✓	224	$n\mathbb{G}_1$ add, $2P$	$\mathcal{O}(N\kappa)$	Silent setup (One time)
Compact Certificate [58]	×	24576	$816\mathbb{G}_1$ add, $544\mathbb{G}_1$ mul	$\mathcal{O}(\kappa)$	Silent setup (One time)
Compressed Σ protocol [15, 14]	✓	$\mathcal{O}(\kappa\log_2 n)$	$\mathcal{O}(n)$ $\mathbb{G}_1$ mul, 2 P	$\mathcal{O}(N\kappa)$	Silent setup (One time)
Das et al. STS [30]	×	536	15 $\mathbb{G}_1$ mul, 15 P	$\mathcal{O}(\kappa)$	Silent setup (Each epoch)
hinTS [44]	×	688	8 $\mathbb{G}_1$ mul, 10 P	$\mathcal{O}(\kappa)$	Silent setup (Each epoch)
Dyna-hinTS (This work)	✓	1040	13 $\mathbb{G}_1$ mul, 10 P	$\mathcal{O}(\kappa)$	Silent setup (One time) + Beacon

TABLE 1: Comparison of threshold signing protocols, where N = total # of signers, n = committee size, κ = security parameter, P = the # of pairings and $\mathbb{G}_1$ mul = scalar multiplication in additive group $\mathbb{G}_1$.

i.e ω is a generator. N is the total number of parties and $|\mathbb{H}| = N+1$. $\mathsf{T}(x) = \prod_{i=1}^{|\mathbb{H}|}(x-\omega^i)$ is the vanishing polynomial on $\mathbb{H}$. Since $\mathbb{H}$ is a multiplicative subgroup $\mathsf{T}(x) = x^{|\mathbb{H}|} - 1$. Lagrange polynomials on $\mathbb{H}$ are defined by $L_i(x) = \frac{\omega^i}{|\mathbb{H}|}.\frac{x^{|\mathbb{H}|}-1}{x-\omega^i}$ and $L_i(0) = |\mathbb{H}|^{-1}$ for $i \in [N]$.

Our construction of the threshold signature is based on the BLS signature [22] defined as follows.

Definition 1 (BLS Signature)**.** *Let $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_2$ be a random oracle. The BLS signature consists of the following algorithms.*

- $\mathsf{BLS.Gen}$*: It samples a random* $\mathsf{sk} \leftarrow\$ \mathbb{F}$ *and output a public/secret key pair as* $(\mathsf{pk} = [\mathsf{sk}]_1, \mathsf{sk})$.
- $\mathsf{BLS.Sign}(m)$*: It signs as* $\sigma = \mathcal{H}(m)^{\mathsf{sk}}$.
- $\mathsf{BLS.Verify}(\mathsf{pk}, m, \sigma)$*: It verifies the validity of the signature by* $e(\mathsf{pk}, \mathcal{H}(m) \stackrel{?}{=} e([1]_1, \sigma)$.

Definition 2 (Polynomial Commitment)**.** *We use the KZG polynomial commitment [53]. In this scheme, the CRS is $([1]_1, [\tau]_1, \ldots, [\tau^D]_1, [\tau]_2)$ for some random τ and a maximum degree D. The commitment to a polynomial $f(x) = a_0 + a_1 \cdot x + \cdots + a_n \cdot x^n$ is $\sigma = [f(\tau)]_1$, which can be computed as $\prod_{i=0}^{n}[\tau^i]_1^{a_i}$. To open the polynomial at x^*, one computes the quotient polynomial $Q(x) = \frac{f(x)-x^*}{x-x^*}$. The opening proof is $\mathsf{open}_r = [Q(\tau)]_1$. To verify the proof, one checks $e(\sigma, [1]_2) \stackrel{?}{=} e(\mathsf{open}_r, [\tau - x^*]_2)$.*

If the prover wants to open ℓ commitment $\sigma_1, \ldots, \sigma_\ell$ at the same place $x = x^*$, we have the following batching optimization. Let $\pi_1, \ldots, \pi_\ell$ be the corresponding opening proof. The verifier can pick a random r and check $e(\sigma_1\sigma_2^{r^2}\cdots\sigma_\ell^{r^\ell}, [1]_2) \stackrel{?}{=} e(\pi_1\pi_2^{r^2}\cdots\pi_\ell^{r^\ell}, [\tau - x^*]_2)$. In particular, the random challenge r can be picked non-interactively by the Fiat-Shamir heuristic, and the prover only needs to send one group element $\pi = \pi_1\pi_2^{r^2}\cdots\pi_\ell^{r^\ell}$ as the batched proof.

Our IPA protocol relies on the generalized sum-check lemma. We refer the readers to Theorem 1 of [62] for proof.

Lemma 1 (Generalized Sumcheck [62])**.** *Let $A(x) = \sum_{i=1}^{|\mathbb{H}|} a_i \cdot L_i(x)$, $B(x) = \sum_{i=1}^{|\mathbb{H}|} b_i \cdot L_i(x)$. It holds that*

$$A(x)\cdot B(x) = \frac{\sum_i a_i \cdot b_i}{|\mathbb{H}|} + X(x)\cdot x + Z(x)\cdot \mathsf{T}(x),$$

where both $\vec{X}$ and $\vec{Z}$ are polynomials with degree $\leqslant |\mathbb{H}| - 2$ defined as

$$X(x) = \sum_i a_i \cdot b_i \cdot \frac{L_i(x) - L_i(0)}{x},$$

$$Z(x) = \sum_i a_i \cdot b_i \cdot \frac{L_i^2(x) - L_i(x)}{\mathsf{T}(x)} + \sum_{i\neq j} a_i \cdot b_j \cdot \frac{L_i(x)\cdot L_j(x)}{\mathsf{T}(x)}$$

Here $\mathsf{T}(x) = \prod_{i=1}^{|\mathbb{H}|}(x-\omega^i)$ is the vanishing polynomial on $\mathbb{H}$.

Definition 3 (Randomness Beacon)**.** *A randomness beacon RB [31] is defined using a tuple of four algorithms:*

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

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

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

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

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

In Appendix. B we present additional preliminaries that we need for our security proof.

3. Technical Overview

We initiate our technical discussion by recalling the BLS multi-signature protocol and the protocol of hinTS [44] as we build on top of them. And then we present our results.

BLS Multi-signature: In a BLS [22] multi-signature protocol, the signers perform a silent setup to sample BLS [22]

public-secret key pairs (pk_i, sk_i), where $pk_i = g^{sk_i}$, and some additional information that we will gradually discuss. Suppose there are N signers. Now if a set S of signers want to sign a message m, then they generate their partial signatures as $\sigma_i = \mathcal{H}(m)^{sk_i}$. These signatures are aggregated into $\sigma = \prod_{i\in\mathsf{S}} \sigma_i$ and can be verified against $\mathsf{aPK}_\mathsf{S} = \prod_{i\in\mathsf{S}} pk_i$. However, constructing aPK_S requires storing all N public keys on the blockchain, which is expensive.

The hinTS Protocol: To address this problem, the work of hinTS [44] proposed a way to succinctly store all the public keys in a short verification key VK. This VK will be used to verify the correct computation of aPK_S using a succinct non-interactive argument (SNARK). In more detail, the aggregator proves the well-formedness of aPK_S as follows. Let $\overrightarrow{\mathsf{SK}}(x)$ be the polynomial that encodes all the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$. VK is a polynomial commitment [53] to the secret key vector $\overrightarrow{\mathsf{SK}}(x)$. The signers generate it in a distributed fashion by providing hinTS to an aggregator. The aggregator runs a public algorithm on the public keys and hinTS to compute VK.

When the set S of signers sign a message m, the signature would verify w.r.t. aPK_S. So, the aggregator computes the aggregated public key aPK_S as:

$$\mathsf{aPK}_\mathsf{S} = \prod_{i\in\mathsf{S}} pk_i = \prod_{i\in\mathsf{S}} g^{sk_i} = g^{\sum_{i\in\mathsf{S}} \mathsf{sk}_i} = g^{\mathsf{aSK}_\mathsf{S}}.$$

The aggregator needs to prove that aPK_S is correctly computed from the polynomial commitment to $\overrightarrow{\mathsf{SK}}(x)$. To perform this, the aggregator encodes the set of signers as $\overrightarrow{\mathsf{B}}_\mathsf{S}(x) = (b_1, b_2, \ldots, b_N)$ where $b_i = 1$ if $i \in \mathsf{S}$, else $b_i = 0$. The aggregator also computes a polynomial commitment to $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$. Then the aggregator proves two statements: (1) the Hamming weight of $(b_1, \ldots, b_n)$ is t; and (2) the inner product between $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and $(b_1, \ldots, b_n)$ is aSK_S. The first statement is computed using a Plonk-based SNARK for threshold t and polynomial commitment to $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$. [44] used sum-check arguments of [62] (cf. Lemma 1) for proving the second statement.

$$\overrightarrow{\mathsf{SK}}(x)\,\overrightarrow{\mathsf{B}}_\mathsf{S}(x) - \mathsf{aSK}_\mathsf{S} = \mathsf{Z}_\mathsf{S}(x)\cdot\mathsf{T}(x) + \mathsf{X}_\mathsf{S}(x)\cdot x,$$

where $\mathsf{T}(x)$ is the vanishing polynomial over the multiplicative subgroup $\mathbb{H}$ (Ref. to Sec. 2). The aggregator computes polynomial commitments to $\mathsf{X}(x)$ and $\mathsf{Z}(x)$ from the hinTS. Then, the verifier verifies the above argument given the polynomial commitments to $\overrightarrow{\mathsf{SK}}(x)$, $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$, aPK_S, $\mathsf{X}(x)$ and $\mathsf{Z}(x)$. The verification is performed in the exponent via pairings. The final succinct signature consists of the aggregated public key aPK_S, aggregated signature σ, the commitment to $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$, and two proofs $\pi_\mathsf{S}^{\mathsf{WT}}$ (that Hamming weight of $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$ is t) and π^{IPA} (that proves the correct computation of aPK_S).

Committee-Setting: In hinTS [44] (and also [30]), any set of t signers can generate partial signatures, which are then aggregated into the final signature. The set S of signers is encoded in the polynomial commitment to $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$. However, if we restrict the signers to a specific committee Com instead of allowing the entire set of participants, it becomes impossible to determine whether the aggregated signature includes signatures from signers outside Com, due to the succinctness of the polynomial commitment to $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$.

Dyna-hinTS: Let us denote the size of the committee as n. The naive solution is to send the committee vector as part of the signature and then prove that $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$ is a subset of this. But that blows up the signature size by $\min(n\log_2 N, N)$ bits, i.e. the number of bits required to encode the committee. So instead, the aggregator encodes the committee of signers as $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x) = (b_1, b_2, \ldots, b_N)$ where $b_i = 1$ if $i \in \mathsf{Com}$, else $b_i = 0$ and provides a succinct polynomial commitment to $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$. Now, the aggregator needs to prove that (1) $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$ is a subset of $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$; and (2) $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$ encodes the committee Com.

To prove the first statement, the aggregator encodes the set $\overline{\mathsf{S}}$ of signers in Com that is not in S as $\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(x) = \overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x) - \overrightarrow{\mathsf{B}}_\mathsf{S}(x) = (b_1, b_2, \ldots, b_N)$ where $b_i = 1$ if $i \in \mathsf{Com} \setminus \mathsf{S}$, else $b_i = 0$. To perform the proof, the aggregator computes the polynomial commitment to $\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(x)$ from the polynomial commitments to $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$ and $\overrightarrow{\mathsf{B}}_\mathsf{S}(x)$ as

$$\left[\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(x)\right] = \frac{\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)\right]}{\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(x)\right]}.$$

Then the aggregator proves that the Hamming weight of $\left[\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(x)\right]$ is $n - t$ using a SNARK proof. The aggregator includes $\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)\right]$ and the SNARK proof as part of the signature.

Next, the aggregator needs to prove that $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$ encodes the committee Com. Before going into the proof let's consider how the committee is sampled. We assume that all the parties have access to a randomness beacon that periodically outputs a random value $\mathsf{y} \in \{0,1\}^\kappa$ along with a proof of correct computation π^{RB}. Given y, we apply a random oracle to expand it to $n\log_2 N$ bits and then select n committee members (each denoted by $\log_2 N$ bits) from the set of N signers. This extended output represents the committee $\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)$. The aggregator commits to as $\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)\right]$ using a polynomial commitment. The aggregator attaches $(\mathsf{y}, \pi^{\mathsf{RB}}, \left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)\right])$ as part of the signature. To verify the correct computation of $\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}}(x)\right]$ the verifier recomputes the above computation. This requires verification of one beacon proof, one hash computation and n group multiplications while keeping the signature size independent of the committee size n. This step can be reused by multiple signatures that are supposed to be generated by the same committee. That would effectively reduce the work that is proportional to the committee size amortized over multiple signatures.

To summarize, our work extends the work of hinTS [44] and [30] to work in the setting where signers are restricted to specific committees. We do this by adding two steps on top of the signature of [44] while keeping the signature size independent of the committee size.

4. Dyna-hinTS Construction

In this section, we present the Dyna-hinTS protocol that provides the *first* silent threshold signatures in the committee setting. We provide the definition of c-STS in Appendix C. We assume that there are N signer parties. For simplicity, the weight of each signer w_i is assumed to be 1 but it can be extended for a general weight $w_i \in \mathbb{N}$. At each predefined epoch e a designated committee Com_e is chosen using a randomness beacon output y, where the beacon is run on input e. Each committee contains n signers. To sign a message m, t signers from the committee provide their partial signatures σ_i which are publicly aggregated into the final signature σ. We present our protocol in Figs. 1 and 2. It proceeds in three phases:

Preprocessing Phase: Each party P_i samples $sk_i \leftarrow\$ \mathbb{F}$ and computes $pk_i = \left[sk_i\right]_1$. We refer to Sec. 2 for the notations. In addition, each party generates the hinTS as follows. P_i computes the Lagrange basis polynomials $L_1(x), L_2(x), ..L_{n+1}(x)$ where $L_i(x) = \frac{\omega^i}{|\mathbb{H}|} \cdot \frac{x^{|\mathbb{H}|}-1}{x-\omega^i}$ over the multiplicative subgroup $\mathbb{H} = \{\omega, \omega^2, .., \omega^n, \omega^{n+1}\} \subset \mathbb{F}$ and uses this to compute the following as hinTS_i:

$$\left[sk_i \cdot L_i(\tau)\right]_1, \left[sk_i \cdot \frac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\right]_1, \left\{\left[sk_i \cdot \frac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\right]_1\right\}_{i\neq j},$$

$$\left[sk_i \cdot \frac{L_i(\tau)-L_i(0)}{\tau}\right]_1, \left[sk_i \cdot (L_i(\tau) - L_i(0))\right]_1,$$

where $\mathsf{T}(x)$ is the vanishing polynomial over $\mathbb{H}$. Party P_i sends (pk_i, hinTS_i) to the aggregator.

The aggregator checks the validity of the hinTS_i of each party P_i. It samples a $\gamma \leftarrow \mathbb{F}$, performs a random linear combination (in the exponent) on each part using $\gamma, \gamma^2, \ldots, \gamma^{N+4}$ and then using a single pairing check to verify the entire hinTS_i. The detailed protocol can be found in Fig. 3. We also use Pippinger's method [18] for optimizing the computation of $N+3$ multi-scalar exponentiation used in the random linear combination yielding concrete boost in the preprocessing phase. The aggregator keeps track of the parties whose hinTS did not verify in the set E and sets $pk_i = 1$ (as a result of $sk_i = 0$) for all the $P_i \in E$. The aggregator also sets the weight of all those corrupt parties as $w_i = 0$, removing them from being included in the set of valid signers. Then the aggregator computes the succinct verification key $\text{VK} = \left(\left[\overrightarrow{\mathsf{SK}}(\tau)\right]_1, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, \left[\mathsf{T}(\tau)\right]_2\right)$ as the polynomial commitments to the secret keys, their weights, and evaluation of the vanishing polynomial on roots of unity. This computation is performed by multiplying individual parts of the public hinTS_i sent by each party. The aggregator posts VK on the public bulletin board.

Committee-Selection-Verification Phase: At each epoch e, the public beacon outputs a random value y and a proof of correct computation π^{RB}. The committee Com_e for this epoch is chosen by deriving randomness from y. The aggregator computes the initial committee. It then removes the list of corrupted parties (whose hinTS did not verify) from the committee. It then denotes the list of parties in the committee whose hinTS verified as $\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}$. It then generates a polynomial commitment to Com_e as $\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2$ using the crs. The list of corrupt parties is denoted as E_{Com_e}. The aggregator posts $(\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2, \pi_{\mathsf{Com}_e})$ on the bulletin board, where $\pi_{\mathsf{Com}_e} = (\mathsf{y}, \pi^{\mathsf{RB}}, E_{\mathsf{Com}_e})$. Anyone can publicly verify the committee by redoing the aggregator computation.

Threshold Signing, Aggregation and Verification: To sign a message m, each party of Com_e generates the partial signature $\sigma_i = (\mathcal{H}(m, e))^{sk_i}$ using their secret key sk_i. Each party also generates a proof of correct signing by generating a proof $\pi_i^{\mathsf{eq\text{-}DL}}$ that $(\left[1\right]_1, \mathcal{H}(m, e))$ and (pk_i, σ_i) share the same exponent. We use the NIZK proof from [50] for this purpose to replace the expensive pairing-based check for verifying each partial signature. Note that we include the epoch e inside the hash (of the BLS signature) to bind the signature to epoch e. The same partial signature cannot be reused for a different epoch e'. This will be crucial for our security (discussed later in the proof) in the committee setting.

Upon obtaining the partial signatures and the proofs of correct signing, the aggregator verifies each proof and discards the partial signatures whose proofs did not verify. Next, the aggregator considers the valid partial signatures and denotes the set of valid signers as S. Then, it computes aggregated signature σ, aggregated public key aPK_S and the commitment $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$ to the signers. In addition, the aggregator computes the proof of correct computation of aPK_S as π^{IPA}(details discussed in Sec. 3). It also computes the proofs - $(\pi_\mathsf{S}^{\mathsf{WT}}, \pi_{\overline{\mathsf{S}}}^{\mathsf{WT}})$ that S is a subset of Com_e. The overview of these two proofs is discussed in Sec. 3. The details of the proofs can be found in Fig. 2, and Fig. 6, 7 respectively. The aggregator sets the final signature as σ, aPK_S, $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$, π^{IPA}, $\pi_\mathsf{S}^{\mathsf{WT}}$ and $\pi_{\overline{\mathsf{S}}}^{\mathsf{WT}}$, and sends it to the verifier.

The verifier verifies the proofs of correct computation of aPK_S and the proofs that S is a subset of Com_e using VK. Once these two proofs verify, the verifier verifies the BLS signature as $e(\mathsf{aPK}_\mathsf{S}, \mathcal{H}(m, e)) \stackrel{?}{=} e(\left[1\right]_1, \sigma)$. This completes the full description of our protocol.

We prove the security of our protocol by proving the following theorem in Appendix E.

Theorem 1. *Assuming RB is a randomness beacon, $\mathcal{H}$ is a random oracle, NIZK is a non-interactive ZK proof of equal exponent [50], there are N signers out of which a PPT adversary $\mathcal{A}$ initially/statically corrupts f signers maliciously. We assume that n parties are in a committee and t partial signatures are required to sign a message, then the protocol in Figs. 1 and 2 satisfies (N, f, n, t)-correctness (Def. 7) and (N, f, n, t)-unforgeability (Def. 8), except with $\mathsf{negl}(\kappa)$ probability and statistical error $\binom{f}{i} \cdot \binom{N-f}{n-i} / \binom{N}{n}$.*

We set the total number of signers as $N = 1024$, statistical error as 2^{-40} (similar to [41]) and the reconstruction threshold as $t = 80$ so that a message needs partial signatures from t parties in a committee. Based on that we consider four different corruption scenarios where $f = 79, \frac{N}{3}, \frac{N}{2}, \frac{2N}{3}$ and observe that the committee size varies as $n = 128, 128, 95$ and 82 respectively by following the

Notations: There are N parties in the universe. Each committee has n parties where at least t parties from the committee sign each message. RB is a randomness beacon that periodically outputs $(\mathsf{y}, \pi^{\mathsf{RB}})$. $L_i(x)$ is the Lagrange basis polynomial over the multiplicative subgroup $\mathbb{H} = \{\omega, \omega^2, .., \omega^N, \omega^{N+1}\} \subset \mathbb{F}$, and $\mathsf{T}(\tau)$ is the vanishing polynomial over $\mathbb{H}$.

Public Parameters: $\mathsf{crs} = (\mathcal{H}, [\tau]_1, [\tau^2]_1, .., [\tau^N]_1, [\tau]_2, [\tau^2]_2, .., [\tau^N]_2)$, where $\tau \leftarrow\!\$\ \mathbb{F}$ is secret and $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}_2$ is a hash function modeled as a random oracle. $\mathsf{NIZK} = (\mathsf{Prove}, \mathsf{Ver})$ over statement $([a]_1, [b]_1, [ac]_2, [bc]_2)$ is a NIZK proof system for proof of equal (secret) exponent c [50].

KeyGen(1^κ): Each party P_i ($i \in [N]$) samples $sk_i \leftarrow\!\$\ \mathbb{F}$ and computes $pk_i = [sk_i]_1$.

HintsGen(crs, sk_i, N): Each party P_i (for $i \in [N]$) computes hinTS_i which consists of $[sk_i \cdot L_i(\tau)]_1$, $[sk_i \cdot \frac{L_i^2(\tau) - L_i(\tau)}{\mathsf{T}(\tau)}]_1$, $\{[sk_i \cdot \frac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}]_1\}_{i \neq j}$, $[sk_i \cdot \frac{L_i(\tau) - L_i(0)}{\tau}]_1$, $[sk_i \cdot (L_i(\tau) - L_i(0))]_1$. Party P_i sends (pk_i, hinTS_i) to the aggregator Agg.

Preprocess$(\mathsf{crs}, \{pk_i, \text{hinTS}_i\}_{i \in [N]})$: The aggregator Agg computes the verification key VK as follows:

1) Computes the set of parties whose hints did not verify as $E \leftarrow \mathsf{Hints\text{-}Verify}(N, \{pk_i, \text{hinTS}_i\}_{i \in [N]})$ (Fig. 3).
2) Agg sets $pk_i = 1$, $sk_i = 0$ and weight $\mathsf{w}_i = 0$ for $i \in E$, and $\mathsf{w}_i = 1$ for $i \notin E$. Set $\overrightarrow{\mathsf{W}} = \{\mathsf{w}_i\}_{i \in [N]}$.
3) For each $i \in [N] \setminus E$, the aggregator sets $[\mathsf{Z}_i]_1$, $[\mathsf{X}_i]_1$ and $[\mathsf{X}_i \cdot \tau]_1$ as follows:

$$[\mathsf{Z}_i]_1 = [sk_i \cdot \tfrac{L_i^2(\tau) - L_i(\tau)}{\mathsf{T}(\tau)}]_1 \cdot \prod_{j \in [N] \setminus (i \cup E)} (sk_j \cdot [\tfrac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}]_1),$$

$$[\mathsf{X}_i]_1 = [sk_i \cdot \tfrac{L_i(\tau) - L_i(0)}{\tau}]_1, [\mathsf{X}_i \cdot \tau]_1 = [sk_i \cdot (L_i(\tau) - L_i(0))]_1$$

4) The aggregator sets aggregation key $\mathsf{AK} = (E, \overrightarrow{\mathsf{W}}, \{[\mathsf{Z}_i]_1, [\mathsf{X}_i]_1, [\mathsf{X}_i \cdot \tau]_1\}_{i \in [N] \setminus E})$.
5) Agg computes $\mathsf{VK} = ([\overrightarrow{\mathsf{SK}}(\tau)]_1, [\overrightarrow{\mathsf{W}}(\tau)]_1, [\mathsf{T}(\tau)]_2)$ where $[\overrightarrow{\mathsf{SK}}(\tau)]_1 = \sum_{i \in [N]} [sk_i . L_i(\tau)]_1$, $[\overrightarrow{\mathsf{W}}(\tau)]_1 = \sum_{i \in [N]} [\mathsf{w}_i . L_i(\tau)]_1$, $[\mathsf{T}(\tau)]_2 = \sum_{i \in [N+1]} [\tau - \omega^i]_2$. Agg posts $\mathsf{VK} = ([\overrightarrow{\mathsf{SK}}(\tau)]_1, [\overrightarrow{\mathsf{W}}(\tau)]_1, [\mathsf{T}(\tau)]_2)$ on the public bulletin board.

Committee-Selection: At epoch $e \in \mathbb{N}$, the beacon RB outputs $(\mathsf{y}, \pi^{\mathsf{RB}})$. Parse $\mathsf{y} \in \{0,1\}^{n \log_2 N}$ as the committee Com_e consisting of n committee members.

5) Agg considers the list of parties in Com_e whose hints failed to verify as $E_{\mathsf{Com}_e} = \mathsf{Com}_e \cap E$. Agg removes those parties from the committee and updates $\mathsf{Com}_e = \mathsf{Com}_e \setminus E_{\mathsf{Com}_e}$.
6) Compute the commitment $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2$ to the vector of parties for the committee as $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2 = \prod_{i \in \mathsf{Com}_e} [L_i(\tau)]_2$.
7) The aggregator sets the proof as $\pi_{\mathsf{Com}_e} = (\mathsf{y}, \pi^{\mathsf{RB}}, E_{\mathsf{Com}_e})$ and commitment to the committee as $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2$. Post $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2$ and π_{Com_e} on the bulletin board.

Committee-Verification$(\mathsf{crs}, e, [\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2, \pi_{\mathsf{Com}_e})$: Verify the commitment to the committee $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2$:

8) Parse the proof $\pi_{\mathsf{Com}_e} = (\mathsf{y}, \pi^{\mathsf{RB}}, E_{\mathsf{Com}_e})$.
9) Check $\mathsf{RB.Ver}(pk_{\mathsf{RB}}, e, \mathsf{y}, \pi^{\mathsf{RB}}) == 1$.
10) Parse $\mathsf{y} \in \{0,1\}^{n \log_2(N)}$ as the committee Com_e and verify that it consists of n committee members. Update $\mathsf{Com}_e = \mathsf{Com}_e \setminus E_{\mathsf{Com}_e}$.
11) Verify that $[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}(\tau)]_2 = \prod_{i \in \mathsf{Com}_e} [L_i(\tau)]_2$.

Figure 1: One-time Non-interactive Preprocessing Phase and Committee-Selection at epoch e

Partial-Sign($\mathsf{crs}, sk_i, m,\ e$): To sign message m, party P_i ($\in \mathsf{Com}_e$) computes partial signature σ_i and a proof of partial signing $\pi_i^{\mathsf{eq\text{-}DL}}$ as:

$$\sigma_i = (\mathcal{H}(m,e))^{sk_i}, \quad \pi_i^{\mathsf{eq\text{-}DL}} \leftarrow \mathsf{NIZK.Prove}((\left[1\right]_1, pk_i, \mathcal{H}(m,e), \sigma_i), sk_i).$$

The party sends $(\sigma_i, \pi_i^{\mathsf{eq\text{-}DL}})$ to the aggregator.

SignAgg($\mathsf{crs}, \mathsf{AK}, m, \{\sigma_i, \pi_i^{\mathsf{eq\text{-}DL}}\}_{i\in\mathsf{S}},\ e$):

12) PartialVerify: Agg performs by verifying the partial signatures sent by the signers in set S. For $i \in \mathsf{S}$: If $\mathsf{NIZK.Ver}((\left[1\right]_1, pk_i, \mathcal{H}(m), \sigma_i), sk_i, \pi_i^{\mathsf{eq\text{-}DL}}) = 0$ then remove P_i from signing set as $\mathsf{S} = \mathsf{S} \setminus i$.
13) Agg computes the aggregated signature σ, aggregated public key $\mathsf{aPK_S}$ and the commitment $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$ to the signers as

$$\sigma = \left(\prod_{i\in\mathsf{S}} \sigma_i\right)^{-|\mathbb{H}|}, \quad \mathsf{aPK_S} = \left(\prod_{i\in\mathsf{S}} pk_i\right)^{-|\mathbb{H}|}, \quad \left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2 = \left[\sum_{i\in\mathsf{S}} L_i(\tau)\right]_2.$$

14) Agg computes the proof of correct computation of $\mathsf{aPK_S}$ and $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$ by proving the following equation, $\overrightarrow{\mathsf{SK}}(x)\overrightarrow{\mathsf{B}}_\mathsf{S}(x) - \mathsf{aSK_S} = \mathsf{Z_S}(x)\cdot\mathsf{T}(x) + \mathsf{X_S}(x)\cdot x$. The aggregator computes the commitment to the quotient polynomials as $\left[\mathsf{Z_S}\right]_1$ and $\left[\mathsf{X_S}\right]_1$. Furthermore, to prove that $\mathsf{X_S}(x)$ has a degree of $\leq N-1$, the aggregator computes $\left[\mathsf{X_S}\cdot\tau\right]_1$.

$$\left[\mathsf{Z_S}\right]_1 = \prod_{i\in\mathsf{S}} \left[\mathsf{Z}_i\right]_1, \quad \left[\mathsf{X_S}\right]_1 = \prod_{i\in\mathsf{S}} \left[\mathsf{X}_i\right]_1, \quad \left[\mathsf{X_S}\cdot\tau\right]_1 = \prod_{i\in\mathsf{S}} \left[\mathsf{X}_i\cdot\tau\right]_1.$$

The aggregator sets the proof as $\pi^{\mathsf{IPA}} = (\left[\mathsf{Z_S}\right]_1, \left[\mathsf{X_S}\right]_1, \left[\mathsf{X_S}\cdot\tau\right]_1)$.

15) Aggregator proves that S is a subset of Com_e by proving that $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$ has weight t and $\left[\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(\tau)\right]_2 = \frac{\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2}{\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2}$ has weight $\bar{t} = n - (t + |E_{\mathsf{Com}_e}|)$, where $\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2$ has weight $n - |E_{\mathsf{Com}_e}|$.
   - It sets $\overrightarrow{\mathsf{B}}_\mathsf{S} = \{b_i\}_{i\in[N]}$, where $b_i = 1$ for $i \in \mathsf{S}$, otherwise $b_i = 0$. It sets $\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}} = \{b'_i\}_{i\in[N]}$, where $b'_i = 1$ for $i \in \mathsf{Com}_e \setminus \mathsf{S}$, otherwise $b'_i = 0$. It sets $\mathsf{aux} = (\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2, |E_{\mathsf{Com}_e}|, \mathsf{aPK_S}, \left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2, \pi^{\mathsf{IPA}})$.
   - It computes the proofs (using Fig. 6, 7) as $\pi_\mathsf{S}^{\mathsf{WT}}$ = Prove($\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1$, t, N, aux; $\overrightarrow{\mathsf{B}}_\mathsf{S}$, $\overrightarrow{\mathsf{W}}$) and $\pi_{\overline{\mathsf{S}}}^{\mathsf{WT}}$ = Prove($\left[\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}(\tau)\right]_2$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1$, $\bar{t}$, N, aux; $\overrightarrow{\mathsf{B}}_{\overline{\mathsf{S}}}$, $\overrightarrow{\mathsf{W}}$) respectively.
16) Outputs $(\sigma, \mathsf{aPK_S}, \left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2, \pi^{\mathsf{IPA}}, \pi_\mathsf{S}^{\mathsf{WT}}, \pi_{\overline{\mathsf{S}}}^{\mathsf{WT}})$.

Verify($m, (\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2, |E_{\mathsf{Com}_e}|, \sigma, \mathsf{aPK_S}, \left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2, \pi^{\mathsf{IPA}}, \pi_\mathsf{S}^{\mathsf{WT}}, \pi_{\overline{\mathsf{S}}}^{\mathsf{WT}}), t, \mathrm{VK}, e)$:

17) Parse $\mathrm{VK} = \left(\left[\overrightarrow{\mathsf{SK}}(\tau)\right]_1, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, \left[\mathsf{T}(\tau)\right]_2\right)$.
18) Verify aggregated public key $\mathsf{aPK_S}$ and $\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$ by checking $e(\left[\overrightarrow{\mathsf{SK}}(\tau)\right]_1, [\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2)\cdot e(\mathsf{aPK_S}, [1]_2)^{-1} \stackrel{?}{=} e(\left[\mathsf{Z_S}\right]_1, \left[\mathsf{T}(\tau)\right]_2)\cdot e(\left[\mathsf{X_S}\right]_1, [\tau]_2)$.
    Also, verify that $\mathsf{X_S}(x)$ has a degree of $\leq N-1$ by checking $e(\left[\mathsf{X_S}\right]_1, [\tau]_2) = e(\left[\mathsf{X_S}\cdot\tau\right]_1, [1]_2)$.
19) Verify that the signer set S is a part of Com_e. Set $\mathsf{aux} = (\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2, |E_{\mathsf{Com}_e}|, \mathsf{aPK_S}, \left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2, \pi^{\mathsf{IPA}})$. Check Verify($\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1$, t, N, aux, $\pi_\mathsf{S}^{\mathsf{WT}}$)=1 and Verify($\frac{\left[\overrightarrow{\mathsf{B}}_{\mathsf{Com}_e}\right]_2}{\left[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)\right]_2}$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1$, $\bar{t}$, N, aux, $\pi_{\overline{\mathsf{S}}}^{\mathsf{WT}}$)=1.
20) Verify BLS signature as $e(\mathsf{aPK_S}, \mathcal{H}(m,e)) \stackrel{?}{=} e(\left[1\right]_1, \sigma)$.

If any of the above checks fail then return 0, else return 1.

Figure 2: Committee-Based Threshold Signing Protocol at epoch e

Public Parameters: $\mathsf{crs} = (\mathcal{H}, [\tau]_1, [\tau^2]_1, .., [\tau^N]_1, [\tau]_2, [\tau^2]_2, .., [\tau^N]_2)$, where $\tau \leftarrow\$ \mathbb{F}$ is secret and $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}_2$ is a hash function modeled as a random oracle.

Hints-Verify$(N, \{pk_i, \text{hinTS}_i\}_{i\in[N]})$: Repeat the following steps for each party P_i for $i \in [N]$:

- Parse $\text{hinTS}_i = \big[sk_i \cdot L_i(\tau)\big]_1, \big[sk_i \cdot \frac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\big]_1, \big\{\big[sk_i \cdot \frac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\big]_1\big\}_{i\neq j}, \big[sk_i \cdot \frac{L_i(\tau)-L_i(0)}{\tau}\big]_1, \big[sk_i \cdot (L_i(\tau) - L_i(0))\big]_1$.
- Compute $\big[L_i(\tau)\big]_2$, $\big[\frac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\big]_2$, $\big[\frac{L_i(\tau)-L_i(0)}{\tau}\big]_2$, $\big[(L_i(\tau) - L_i(0))\big]_2$ and $\big\{\big[\frac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\big]_2\big\}_{j\in[N]\setminus i}$ using the crs.
- Sample $\gamma \leftarrow \mathbb{F}_p$ and perform random linear combination to verify hinTS_i as follows:

$$e\Big(\big[sk_i \cdot L_i(\tau)\big]_1^{\gamma} \cdot \big[sk_i \cdot \tfrac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\big]_1^{\gamma^2} \cdot \big[sk_i \cdot \tfrac{L_i(\tau)-L_i(0)}{\tau}\big]_1^{\gamma^3} \cdot \big[sk_i \cdot (L_i(\tau) - L_i(0))\big]_1^{\gamma^4} \cdot \prod_{j\in[N]\setminus i} \big[sk_i \cdot \tfrac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\big]_1^{\gamma^{j+4}}, [1]_2\Big)$$

$$\stackrel{?}{=} e\Big(pk_i, \big[L_i(\tau)\big]_2^{\gamma^1} \cdot \big[\tfrac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\big]_2^{\gamma^2} \cdot \big[\tfrac{L_i(\tau)-L_i(0)}{\tau}\big]_2^{\gamma^3} \cdot \big[(L_i(\tau) - L_i(0))\big]_2^{\gamma_4} \cdot \prod_{j\in[N]\setminus i} \big[\tfrac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\big]_2^{\gamma^{j+4}}\Big)$$

If the above equation does not verify for party P_i then set $E = E \cup i$.

Return E as a set of parties whose hints did not verify.

Figure 3: Hints Verification Algorithm

hypergeometric distribution according to Thm. 1. We discuss more about this in the next section.

5. Performance Analysis

We implement Dyna-hinTS in Rust and compare it with the Rust implementations of BLS multisignature and hinTS[44]. We open source our code at https://github.com/dynaHintsRepo/dynaHints.git.

Experimental Setup: All three protocols have three algorithms - the signer algorithm, the aggregator algorithm, and the verifier algorithm. We implement all three protocols using a single-threaded implementation and using the same configuration. More specifically, the signer and aggregator algorithms are run on a Google Cloud Platform (GCP) instance with an Intel Xeon 2.8GHz CPU with 32 cores and 32 GB RAM. We run the verifier's algorithm on a Macbook Air with Apple M2 chip, 8 GB RAM, and 8 cores. For a fair comparison of all schemes, we only consider the single-threaded implementation of all three protocols, and there are obvious opportunities for massive parallelism. We use the arkworks [13] library for Elliptic Curve operations and hash-to-curve operations. We use the Crate vrf library [11] to implement the VRF protocol for committee generation and use BLS-12381 curve to implement the BLS signatures. Each element in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{F}$ is of size 48 bytes, 96 bytes, and 32 bytes respectively. The gas cost estimations are based on the pre-compiled gas costs from EIP-2537 [67].

Parameter Selection: For our benchmarking purposes, we set the total number of signers as $N = 1024$ and the reconstruction threshold as $t = 80$, so that a message needs partial signatures from t parties in a committee. Based on these two fixed values, we consider three different corruption scenarios by varying f and see how the committee size n varies s.t. Thm. 1 yields 2^{-40} statistical error:

- $f = \frac{N}{3} - 1$ parties are statically corrupt. In this case, we observe that the committee size can be 128.
- $f = \frac{N}{2} - 1$ parties are statically corrupt. In this case, we observe that the committee size can be at most 95.
- $f = \frac{2N}{3} + 1$ parties are statically corrupt. In this case, the committee size can be at most 82.

Higher committee size allows for better liveness since $n - t$ signers can remain inactive during the signing process. However, for BLS multisignatures and hinTS the reconstruction threshold is always $f + 1$. So, we extensively benchmark for different corruption scenarios and compare in Table. 2. In Appendix. F and Figs. 8, 9 and 10 we further show that t in Dyna-hinTS gradually increases with increasing f whereas it grows linearly for BLS multisignatures and hinTS. This yields a massive aggregation time boost, as shown in Table. 2.

5.1. Signature Verification

We compare the signature size, verification time, and gas cost of the three protocols. Table 2 reports the verification complexity of the schemes in terms of signature size, algebraic operations, CPU time, and gas cost. BLS multisignature works in the committee setting. It performs this by encoding the identity of the signers using a bit vector and so the signature size scales linearly with the number of signers since the signature size is 1 $\mathbb{G}_2$ group element and N bits. As a result, the aggregated signature becomes linear in the total number of signers. Similarly, the verification time and gas cost scales linearly with the number (i.e. t) of signers that signed. Concretely, the signatures obtained from BLS multisignature is 224 bytes (128 bytes to represent the signers and one $\mathbb{G}_2$ element of 96 bytes) and it is very efficient for smaller values of N and t, but this grows linearly with N and t. The major downside of multi-signature is the gas cost of storing the public verification key. It takes 60 million gas (as per [44]) to store the BLS multi-signatures public keys of 1024 signers. This is equivalent to $1200 (computed using [67], as of October 2024). This cost grows with the number of signers.

On the other hand, hinTS and Dyna-hinTS both have constant-sized verification keys, constant-sized signatures, and constant verification time, independent of N and t. The storage gas cost is 390K, equivalent to $7.8, and it remains fixed independent of the number of signers. In terms of signature verification, hinTS requires additional pairing checks to verify the SNARK for the correct computation of the aggregated public key – the signature consists of 9 $\mathbb{G}_1$ group elements and 5 field elements for the Plonk

Protocols	Corruption Threshold f	Reconstruction Threshold t	Storage Gas Cost	Signature Size	Partial Signature Aggregation Time	Verification Time	Verification Gas Cost
BLS Multisig	$f = 79$	80	60 M (or $1200)	224 B	0.75 ms	3.15 ms	209K
hinTS [44]		80	390K (or $7.8)	688 B	338 ms	15.06 ms	441K
Dyna-hinTS		80	390K (or $7.8)	1040 B	357 ms	15.77 ms	501K
BLS Multisig	$f \leqslant \frac{N}{3} = 340$	341	60 M (or $1200)	224 B	3.09 ms	3.15 ms	209K
hinTS [44]		341	390K (or $7.8)	688 B	958.48 ms	15.06 ms	441K
Dyna-hinTS		80	390K (or $7.8)	1040 B	357 ms	15.77 ms	501K
BLS Multisig	$f \leqslant \frac{N}{2} = 510$	511	60 M (or $1200)	224 B	4.68 ms	3.15 ms	209K
hinTS [44]		511	390K (or $7.8)	688 B	1460 ms	15.06 ms	441K
Dyna-hinTS		80	390K (or $7.8)	1040 B	357 ms	15.77 ms	501K
BLS Multisig	$f \leqslant \frac{2N}{3} = 682$	683	60 M (or $1200)	224 B	6.59ms	3.15 ms	209K
hinTS [44]		683	390K (or $7.8)	688 B	1776 ms	15.06 ms	441K
Dyna-hinTS		80	390K (or $7.8)	1040 B	357 ms	15.77 ms	501K

TABLE 2: Comparison of BLS multisignature, hinTS and Dyna-hinTS, where the total number of signers is set to $N = 1024$ and Dyna-hinTS's security has an additional statistical error of 2^{-40}.

proof, and 1 $\mathbb{G}_2$ group element as the signature. Concretely, it turns out to be 688 bytes. To verify this signature it takes 10 pairings and 8 $\mathbb{G}_1$ group multiplication to verify a signature. Concretely, it takes 15ms on an Apple M2 chip Macbook Air, demonstrating reasonably fast verification, and consumes 441K gas. But, hinTS does not support dynamic committees. As a result, the signature aggregation time of hinTS drastically increases with the number of signers. Meanwhile, Dyna-hinTS supports committees and results in a lower reconstruction threshold t. When $f = 682$ signers are corrupt, both BLS multi signature and hinTS aggregate 683 partial signatures whereas Dyna-hinTS only aggregates 80 partial signatures yielding a $4.9\times$ improvement in signature generation time.

We obtain this improvement by marginally increasing the signature verification time by 0.7ms (or 4%) over hinTS and also increasing the signature size. This is because we require an additional KZG commitment to verify that the set of signers belongs to the committee. This increases our signature size, w.r.t. hinTS, by 4 $\mathbb{G}_1$ elements and 5 field elements. In terms of computation, we merge this check with the Hamming weight check for the set of t signers limiting the overhead within < 1 ms (or 4%) over hinTS verification. In terms of gas cost for verification, both hinTS and Dyna-hinTS verifiers need to perform 2 pairings to verify the BLS signature and 6 pairings to verify the SNARK proofs from Fig. 2. Additionally, hinTS [44] uses techniques from [19] to batch verify multiple opening proofs of KZG commitments using 2 more pairings and 8 $\mathbb{G}_1$ multiplication. We need 5 additional group multiplications to combine the KZG commitment openings (used in the committee proof) into a single pairing check.

5.2. Committee Selection/Verification Cost

Both BLS multisignatues and Dyna-hinTS work in the committee setting. At each epoch e, the committee is chosen using the output of a beacon protocol. We implement the beacon using a verifiable random function (VRF) run on input e. We used [11] crate to implement the committee generation using the VRF output. The verifier of the signature verifies the committee using the VRF proof. This takes roughly 1.45 ms. In addition, Dyna-hinTS requires a KZG commitment to the identity of the committee members. To compute/verify this commitment involves n group multiplications and requires 0.47 ms. We report these numbers in Tab. 3. The total time to verify the committee is around 1.98 ms in Dyna-hinTS. This cost gets amortized over multiple signatures that are generated in the same epoch by the same committee (but possibly a different set of signers). Furthermore, the cost of VRF proof verification can be reduced to $< 1ms$ if we use optimized VRF protocols, like the one by Chainlink [39].

Total # of Signers	Committee size	VRF proof verification time	Polynomial Com Computation time
256	64	1.44 ms	0.26 ms
512	64	1.47 ms	0.26 ms
512	128	1.45 ms	0.28 ms
1024	82	1.45 ms	0.38 ms
1024	95	1.45 ms	0.40 ms
1024	128	1.45 ms	0.47 ms

TABLE 3: Verification of Committee time for Dyna-hinTS

5.3. Setup Cost

hinTS and Dyna-hinTS both use a one-time silent setup, where each signer together with their public keys broadcast a linear-sized auxiliary material (informally called the hints), whose computation takes $O(N)$ group multiplication and $O(N \log N)$ field operations. Then the Aggregator verifies all such hints sent by the signers in the universe. This step consists of two subprotocols. Firstly, the aggregator needs to compute KZG commitments to $O(N^2)$ Lagrange coefficient polynomials. Then using these polynomial commitments, the aggregator verifies the hints of each signer by performing random linear combinations using Pippenger's algorithm [18] and then performs a pairing check. Generating KZG commitments to $O(N^2)$ Lagrange coefficient polynomials is very expensive as it scales quadratically with N. In Appendix. G, we show it can be delegated to untrusted parties and the aggregator can simply perform random linear combinations to verify these commitments using N pairing checks. After performing the above optimizations, we obtain the total setup cost for $N = 1024$ signers as 66 seconds. We provide the

detailed cost analysis of our optimized setup protocol in Fig. 11.

Acknowledgement

Samipa Samanta worked on this project during her research associateship at Supra Research.

6. Conclusion

The concept of silent threshold signature (STS) merged the benefits of multi-signatures (no DKG) and threshold signatures (low storage and efficient verification). However, in effect, it missed an important feature, namely the support for dynamic committees, which is gaining more popularity for scalability and security. In this work, we add support for the dynamic committee to the silent signature paradigm, and that too, incurring very little overhead on the existing STS. We provide rigorous empirical evidence (and comparison with prior works) to establish practicality.

References

[1] Aptos: The World's Most Production-Ready Blockchain. https://aptosfoundation.org/.

[2] Axelar: The Shortest Path to Scale. https://www.axelar.network/.

[3] Digital Asset Management with MPC (Whitepaper). https://www.coinbase.com/blog/digital-asset-management-with-mpc-whitepaper.

[4] Ethereum: Minimal Light Client. https://github.com/ethereum/annotated-spec/blob/master/altair/sync-protocol.md.

[5] drand: Randomness Beacon Service, 2017. https://drand.love/docs/cryptography/.

[6] Threshold Signature Wallets, 2021. https://sepior.com/mpc-blog/threshold-signature-wallets.

[7] Dock Introduces Threshold Signatures for Secure Multi-Party Credential Transactions, 2023. https://tinyurl.com/3mmpxn6n.

[8] Ethereum Sync Committee. 2024. https://www.inevitableeth.com/home/ethereum/network/consensus/sync-committee.

[9] Ethereum Validator list, 2024. https://beaconcha.in/charts/validators.

[10] Randao, 2024. https://inevitableeth.com/home/ethereum/network/consensus/randao.

[11] Crate vrf library, https://docs.rs/vrf/0.2.4/vrf/.

[12] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 363–373, New York, NY, USA, 2021. Association for Computing Machinery.

[13] arkworks contributors. `arkworks` zksnark ecosystem, 2022.

[14] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543, Santa Barbara, CA, USA, August 17–21, 2020.

[15] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 526–556, Singapore, December 6–10, 2021.

[16] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1454–1470. IEEE, 2023.

[17] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. Threshold cryptography as a service (in the multiserver and YOSO models). In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 323–336, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[18] Daniel J. Bernstein. Pippinger's exponentiation algorithm. 2002.

[19] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *Cryptology ePrint Archive*, 2020.

[20] Dan Boneh, Iftach Haitner, and Yehuda Lindell. Exponent-vrfs and their applications. *IACR Cryptol. ePrint Arch.*, page 397, 2024.

[21] Dan Boneh and Chelsea Komlo. Threshold signatures with private accountability. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 551–581, Santa Barbara, CA, USA, August 15–18, 2022.

[22] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001.

[23] Luís Brandão and Rene Peralta. NIST First Call for Multi-Party Threshold Schemes. 2023. https://csrc.nist.gov/publications/detail/nistir/8214c/draft.

[24] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[25] Chia Network. BLS signatures in C++ using the RELIC toolkit. https://github.com/Chia-Network/bls-signatures. Accessed: 2019-05-06.

[26] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In *Advances in Cryptology – CRYPTO 2023, Part I*, Lecture Notes in Computer Science, pages 743–773, Santa Barbara, CA, USA, August 2023.

[27] Annalisa Cimatti, Francesco De Sclavis, Giuseppe Galano, Sara Giammusso, Michela Iezzi, Antonio Muci, Matteo Nardelli, and Marco Pedicini. Dynamic-frost: Schnorr threshold signatures with a flexible committee. *IACR Cryptol. ePrint Arch.*, page 896, 2024.

[28] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The flexible round-optimized schnorr threshold (FROST) protocol for two-round schnorr signatures. *RFC*, 9591:1–47, 2024.

[29] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In *Advances in Cryptology – CRYPTO 2023, Part I*, Lecture Notes in Computer Science, pages 678–709, Santa Barbara, CA, USA, August 2023.

[30] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 356–370. ACM Press, November 2023.

[31] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy*, pages 2502–2517, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

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

[33] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988.

[34] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990.

[35] DFINITY. go-dfinity-crypto. https://github.com/dfinity/go-dfinity-crypto. Accessed: 2019-05-06.

[36] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[37] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA in three rounds. pages 3053–3071. IEEE Computer Society Press, 2024.

[38] Jack Doerner, Yashvanth Kondi, and Leah Namisa Rosenbloom. Sometimes you can't distribute random-oracle-based proofs. Lecture Notes in Computer Science, pages 323–358, Santa Barbara, CA, USA, August 2024.

[39] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network. *Retrieved March*, 11:2018, 2017.

[40] Hanwen Feng, Tiancheng Mai, and Qiang Tang. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. *IACR Cryptol. ePrint Arch.*, page 1773, 2023. Accepted in ACM CCS'24.

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

[42] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018.

[43] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.

[44] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 3034–3052. IEEE, 2024.

[45] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371, Saragossa, Spain, May 12–16, 1996.

[46] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 1999.

[47] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 458–487, Trondheim, Norway, May 30 – June 3, 2022.

[48] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339, 2021.

[49] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY Technology Overview Series, Consensus System. *CoRR*, abs/1805.04548, 2018.

[50] Aniket Kate, Easwar Vivek Mangipudi, Siva Maradana, and Pratyay Mukherjee. FlexiRand: Output private (distributed) VRFs and application to blockchains. In *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 1776–1790. ACM Press, November 2023.

[51] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive vss using class groups and application to dkg. Cryptology ePrint Archive, Paper 2023/451, 2023. https://eprint.iacr.org/2023/451.

[52] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. *IACR Cryptol. ePrint Arch.*, page 451, 2023. Accepted in ACM CCS'24.

[53] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010.

[54] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. New York, NY, USA, 2020. Association for Computing Machinery.

[55] Chelsea Komlo and Ian Goldberg. FROST: flexible round-optimized schnorr threshold signatures. In *SAC 2020*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2020.

[56] Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In *Advances in Cryptology – CRYPTO 2023, Part I*, Lecture Notes in Computer Science, pages 646–677, Santa Barbara, CA, USA, August 2023.

[57] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *IACR Commun. Cryptol.*, 1(1):25, 2024.

[58] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 626–641, 2021.

[59] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221, Virtual Event, August 16–20, 2021.

[60] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1717–1731, Virtual Event, USA, November 9–13, 2020. ACM Press.

[61] Poly Network. Poly Network, 2020. https://poly.network/.

[62] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 774–804, Virtual Event, August 16–20, 2021.

[63] Rainbow Bridge. Rainbow Bridge, 2020. https://near.org/bridge/.

[64] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2551–2564, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[65] Succinct. Telepathy. https://docs.telepathy.xyz/.

[66] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893, 2020.

[67] Alex Vlasov, Kelly Olson, Alex Stokes, and Antonio Sanso. Eip-2537: Precompile for bls12-381 curve operations. Ethereum Improvement Proposals, 2020. https://eips.ethereum.org/EIPS/eip-2537.

[68] Wormhole. Wormhole. https://docs.wormhole.com/wormhole/.

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

Appendix A. Additional Related Works

Multiverse Threshold Signatures: The work of [16] introduced the notion of multiverse threshold signatures (MTS) where there is an initial set of signers. A verifier defines a universe as a subset of signing parties and a security threshold for the universe. The multiverse encompasses all such universes, which may overlap and have varying security thresholds. A signing party can sign a message regardless of the universes it belongs to, while an aggregator can collect a threshold number of partial signatures from a universe and generate a compact aggregate signature. This signature remains independent of the universe's size and can be verified using the corresponding universe's verification key. In Dyna-hinTS, we can consider each committee as a universe and require the signers to sign as $\mathcal{H}(m)^{\mathsf{sk}_i}$ instead of $\mathcal{H}(m,e)^{\mathsf{sk}_i}$, where e is the committee identifier and sk_i is the secret key share of the signer. This makes the partial signatures committee-agnostic and yields an MTS scheme in a straightforward manner. The aggregator aggregates the partial signatures from the signers into a valid signature and that can be verified against the universe's verification key. We also improve over the MTS scheme of [16] since we can non-interactively setup the universe's verification key. In [16], the signers need to interact among themselves and the aggregator to setup the universe's verification key when they are assigned to a particular universe. Whereas, in the MTS scheme of Dyna-hinTS, the aggregator can compute the universe's verification key without any interaction from the signers. Anyone can publicly verify that the verification key was correctly computed. This gets rid of any interaction among the signers during setting up of the universe's verification key and hence improves over the MTS scheme of [16].

Threshold ECDSA/Schnorr Signatures: There is a long line of works in threshold ECDSA [37, 56, 55, 36] and threshold Schnorr [26, 64, 60, 29, 28, 20, 57]. They require at least two rounds of interaction between the signers to generate the final signature. In fact, it is impossible [38] to make them non-interactive without making the signature size grow with the number of signers. In contrast, our signing protocol is non-interactive in nature. Additionally, all of these signing protocols, except [20], use a DKG protocol to setup the secret shares and the public key, hence, suffering the same limitations as the DKG-based signature protocols and do not support committee-based signing. The protocol of [20] uses an additive sharing to compute the public verification key and as a result, it requires all the parties to remain active for signing. The recent work of [27] extended the FROST protocol [55] for threshold Schnorr to work for committees. In that, an initial committee runs a DKG to setup the public key and perform the threshold signing. When a new committee is selected there is a key re-sharing between the members of the old and new committee. This added communication is expensive in practice as it involves peer-to-peer communication over WAN networks. In contrast, Dyna-hinTS switches committees without any interaction between the committee members.

Accountable Threshold Signatures: Boneh and Komlo [21] puts forward the idea of privately accountable threshold signature, which lets a designated party with a private key to find out the set of signers from an aggregated signature. This is somewhat in between full public accountability of multi-signatures, and no accountability of

threshold signatures. Our construction does not have any such accountability of signers. However, it specifies the committee within the universe of signers, which is publicly checkable.

Appendix B. Additional Preliminaries

In this section, we present the additional preliminaries from Sec. 2.

Definition 4 (Algebraic Group Model)**.** *We work in the Algebraic Group Model (AGM), introduced by [42]. In such a model, adversaries are considered* algebraic. *That is, for any $b \in \{0,1\}$, whenever the adversary outputs a group element $h_b \in \mathbb{G}_b$, it must also output a vector $\vec{v}$ that* explains *h_b. In particular, it must hold that $h_b = \langle \vec{v}, \mathsf{inp}_b \rangle$, where inp_b stands for the vector of group elements from $\mathbb{G}_b$ that the adversary takes as input. Similar to [42], we also assume the following q-DLOG problem is hard for algebraic adversaries.*

Definition 5 (q-DLOG Assumption)**.** *For any positive integer q and algebraic adversary $\mathcal{A}$, it holds that*

$$\Pr\left[x' = x \;\middle|\; \begin{array}{c} x \leftarrow\!\$\, \mathbb{F} \\ x' = \mathcal{A}\begin{pmatrix} [1]_1, [x]_1, \ldots, [x^q]_1, \\ [1]_2, [x]_2, \ldots, [x^q]_2 \end{pmatrix} \end{array}\right] = \mathsf{negl}(n).$$

Assuming the q-DLOG assumption, the following lemma [42] simplifies the security analysis in AGM.

Lemma 2. *Let $[f_1(x_1,\ldots,x_\ell)],\ldots,[f_t(x_1,\ldots,x_\ell)]$ be a sequence of group elements (in either $\mathbb{G}_1$ or $\mathbb{G}_2$) given to an algebraic adversary $\mathcal{A}$ as input, where $x_1,\ldots,x_\ell \leftarrow \mathbb{F}$. Let (g_1,g_2,h_1,h_2) be the output of $\mathcal{A}$. If it holds that $e(g_1,h_1) = e(g_2,h_2)$, with $1-\mathsf{negl}(n)$ probability, the adversary $\mathcal{A}$ must know the corresponding polynomials $g_1(x_1,\ldots,x_\ell) = \sum_{i=1}^t \alpha_i \cdot f_i$, $h_1(x_1,\ldots,x_\ell) = \sum_{i=1}^t \beta_i \cdot f_i$, $g_2(x_1,\ldots,x_\ell) = \sum_{i=1}^t \gamma_i \cdot f_i$, $h_2(x_1,\ldots,x_\ell) = \sum_{i=1}^t \theta_i \cdot f_i$ such that*

$$g_1(x_1,\ldots,x_\ell)\cdot h_1(x_1,\ldots,x_\ell) = g_2(x_1,\ldots,x_\ell)\cdot h_2(x_1,\ldots,x_\ell)$$

holds as a multivariate polynomial identity.

Appendix C. Definition of Committee-based Silent Threshold Signature

The work of [44] introduced the notion of silent threshold signatures (STS). In an STS scheme, parties publish "hints" together with their public key non-interactively in a *silent* manner. Given all the hints, a public algorithm, denoted as the aggregator, verifies the validity of the hints. Furthermore, a *succinct* verification key will be deterministically computed from the hints. Later on, to sign a message m, the signing parties provide partial signatures with proof of correct computation. The aggregator verifies the partial signatures and aggregates the signatures into a *succinct* signature that can be verified against the *succinct* verification key.

We expand the above definition to work for the committee setting, i.e. the set of signers should hail from a specific committee instead of the entire set of signers. Moreover, this committee keeps changing with time at every predefined epoch. Unforgeability requires that for each epoch only the valid committee of signers sign a particular message. Formally, we have the following.

Definition 6 (c-STS)**.** *A Silent Threshold Signature consists of the following algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{HintGen}, \mathsf{Preprocess}, \mathsf{ComSelect}, \mathsf{Sign}, \mathsf{Agg}, \mathsf{Verify})$*:*

- $\mathit{crs} \leftarrow \mathsf{Setup}(1^\kappa)$*: On input the security parameter κ, the* Setup *algorithm outputs a common reference string* crs*.*
- $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$*: On input the security parameter κ, the* KGen *algorithm outputs a public/secret key pair* $(\mathsf{pk},\mathsf{sk})$*.*
- $\mathit{hinTS} \leftarrow \mathsf{HintGen}(\mathit{crs},\mathsf{sk},N)$*: On input the* crs*, the secret key* sk*, and the number of parties N, the* HintGen *algorithm outputs a hint hinTS.*
- $(\mathit{AK},\mathit{VK}) \leftarrow \mathsf{Preprocess}(\mathit{crs},\{\mathit{hinTS}_i,pk_i\}_{i\in[n]})$*: On input the* crs*, all pairs $\{\mathit{hinTS}_i,pk_i\}_{i\in[N]}$, the* Preprocess *algorithm computes an aggregation key* AK *and a (succinct) verification key VK.*
- $\mathit{Com}_e \leftarrow \mathsf{ComSelect}(e,N,n)$*: On input epoch e, the number of parties N, and committee size n, the* ComSelect *algorithm selects a committee* $\mathit{Com}_e \subset [N]$ *of n signing parties for epoch e.*
- $\sigma_i \leftarrow \mathsf{Sign}(sk_i,m,e)$ *: On input some secret key sk_i, some message m, and epoch e, the* Sign *algorithm outputs a partial signature σ_i if $i \in \mathit{Com}_e$, else return $\perp$.*
- $1/0 \leftarrow \mathsf{PartialVerify}(m,\sigma_i,pk_i,e)$ *: On input a message m, a partial signature σ_i, a public key pk_i, and epoch e, it returns 1 if and only if the partial signature is verified correctly and generated by a signer from* Com_e*.*
- $\sigma \leftarrow \mathsf{Agg}(\mathit{crs},\mathsf{AK},\{\sigma_i\}_{i\in B},e)$ *: On input the* crs*, an aggregation key* AK*, a set of signatures $\{\sigma_i\}_{i\in B}$, and epoch e, the* Agg *algorithm outputs a (succinct) signature σ.*
- $1/0 \leftarrow \mathsf{Verify}(m,\sigma,t,\mathit{VK},e)$ *: On input a message m, a signature σ, a threshold t, the verification key VK, and epoch e, it verifies the signature.*

Moreover, the c-STS scheme must have the following efficiency *requirements:*

- *The aggregated verification key* vk *and the aggregated signature σ should be constant size.*
- *The verification time* Verify *should be constant.*

We define the properties of correctness and unforgeability below. Both of these properties are defined against a malicious adversary who may send arbitrary messages (for the hints and partial signatures) on behalf of the corrupted parties, whereas the honest parties are controlled by the challenger.

Definition 7 (Correctness)**.** *The c-STS scheme satisfies (N, f, n, t)- correctness if, for any adversary $\mathcal{A}$, the output of the* Correctness − Game *defined in Figure 4 is* 1 *with probability $\geqslant 1 - \mathsf{negl}(\kappa)$ for every epoch $e \in \mathbb{N}$.*

1) *The challenger runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$ and gives crs to $\mathcal{A}$.*
2) *The adversary picks N and a subset of $f(< N)$ parties to corrupt $A \leftarrow \mathcal{A}(\mathsf{crs})$.*
3) *For all $i \in [N] \setminus A$, the public key and hint are sampled honestly $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^n)$ and $hinTS_i = \mathsf{HintGen}(\mathsf{crs}, sk_i, N)$.*
4) *For all $i \in A$, the adversary returns a public key pk_i and a hint $hinTS_i$ to the challenger.*
5) *The public pre-processing is invoked by the challenger as $(\mathsf{AK}, VK) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{hinTS_i, pk_i\}_{i\in[N]})$ and the output are given to $\mathcal{A}$.*
6) *At epoch e, the challenger obtains the committee $\mathsf{Com}_e \leftarrow \mathsf{ComSelect}(e, N, n)$ and gives it to $\mathcal{A}$.*
7) *To sign a message m of its choice the adversary performs the following:*
   a) *The adversary requests a subset of honest parties $B_1 \subseteq \mathsf{Com}_e \setminus A$ for partial signatures, which are returned by computing $\sigma_i \leftarrow \mathsf{Sign}(sk_i, m, e)$ on a given message for all $i \in B_1$. This process can be repeated by the adversary multiple times on the same message m.*
   b) *The adversary prepares the partial signatures $\{\sigma_i\}_{i\in B_2}$ for some subset of malicious parties $B_2 \subseteq A$. Let $B_2' \subseteq B_2$ be the subset of maliciously generated signatures that verifies under* PartialVerify.
   c) *The challenger computes the aggregated signature as $\sigma \leftarrow \mathsf{Agg}(\mathsf{crs}, \mathsf{AK}, \{\sigma_i\}_{i\in B}, e)$, where $B = B_1 \cup B_2'$ is the set of all partial signatures that verifies.*
8) *The output of this game is 1 if,* for all $t \leqslant |B \cap \mathsf{Com}_e|$, *we have* $\mathsf{Verify}(m, \sigma, t, VK, e) = 1$.[5]

Figure 4: Correctness − Game

In the correctness game, the aggregation process (Step. 7c) by the challenger takes $|B|$ partial signatures that verify. To perform aggregation only $t < |B|$ valid partial signatures are needed. So, the aggregator aggregates the valid partial signatures in B_2' and the first $t - |B_2'|$ partial signatures in B_1 to compute the aggregated signature σ.

Definition 8 (Unforgeability)**.** *The c-STS scheme (N, f, n, t)-unforgeability if, for any adversary $\mathcal{A}$, the output of the game in Figure 5 is* 1 *with probability $\leqslant \mathsf{negl}(n)$ for every epoch $e \in \mathbb{N}$.*

1) *The challenger runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$ and gives crs to $\mathcal{A}$.*
2) *The adversary picks N and a subset of $f(< N)$ parties to corrupt $A \leftarrow \mathcal{A}(\mathsf{crs})$.*
3) *For all honest parties $i \in [N]\setminus A$, the public key and hint are sampled honestly by the challenger $(pk_i, sk_i) \leftarrow \mathsf{KGen}(1^\kappa)$ and $hinTS_i = \mathsf{HintGen}(\mathsf{crs}, sk_i, N)$.*
4) *For all $i \in A$, the adversary picks a public key pk_i and the corresponding hint $hinTS_i$.*
5) *The challenger preprocesses $\{hinTS_i, pk_i\}_{i\in[N]}$ as $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{hinTS_i, pk_i\}_{i\in[N]})$ which are given to $\mathcal{A}$.*
6) *At epoch e, the challenger obtains the committee $\mathsf{Com}_e \leftarrow \mathsf{ComSelect}(e, N, n)$ and gives it to $\mathcal{A}$.*
7) *The adversary may make a partial signature query with a message m and an honest party $i \in \mathsf{Com}_e \setminus A$ – the query is returned by computing $\sigma_i = \mathsf{Sign}(\mathsf{sk}, m, e)$. This is run as many times as desired by $\mathcal{A}$.*
8) *Finally, the adversary shall output a challenge message m^* and an aggregated signature σ^*. Let B^* be the subset of honest parties queried by the adversary to sign m^*.*
9) *The adversary wins the forgery game at epoch e if there exists a threshold t such that either*
   a) *The adversary corrupts an entire committee, i.e. $t \leqslant |A \cap \mathsf{Com}_e|$, or*
   b) *The adversary forges a signature on message m^*, i.e. $t > |(B^* \cup A) \cap \mathsf{Com}_e|$ and $\mathsf{Verify}(m^*, \sigma^*, t, VK, e) = 1$.*

   If the adversary wins the game then the output of the game is 1, otherwise the output is 0.

Figure 5: Forgery − Game

Definition 9 (Silent Weighted Threshold Signature)**.** *In the weighted-setting there is a predefined weight vector $(w_1, w_2, \ldots, w_N)$ associated with all N parties. For every message m and an aggregated signature σ aggregating from $\{\sigma_i\}_{i\in S}$, one claims that a subset of parties with cumulative weight $\sum_{i\in B} w_i$ have signed the message. The formal definition for the weighted setting is essentially analogous to the definition for the threshold case except for the last step of the correctness and forgery games.*

- ***Correctness:*** *The output of this game is 1 if and only if, for all $t \leqslant \sum_{i\in B} w_i$, we have $\mathsf{Verify}(m, \sigma, t, VK, e) = 1$.*
- ***Forgery:*** *The adversary wins the forgery game if there exists a threshold $t > \sum_{i\in B^*\cup A} w_i$ such that $\mathsf{Verify}(m, \sigma, t, VK, e) = 1$, in which case, the output of the game is 1.*

Appendix D. Protocol for Hamming Weight Test on Committed Vectors

We present the protocol for Hamming Weight Test on Committed Vectors in this section due to lack of space in the main body. Our protocol can be found in Fig. 6, 7.

Appendix E. Proof of Theorem 1

We analyze the correctness and unforgeability properties of Dyna-hinTS by proving Theorem 1 as follows:

Correctness: We consider the correctness of the scheme against a malicious adversary $\mathcal{A}$ (similar to the correctness analysis of [44]). An adversarial signer P_i breaks the

5. Note that, since σ is the aggregation of $|B|$ honest signatures, it should verify all threshold $t \leqslant |B|$.

Input: $\left[\overrightarrow{\mathsf{B}}(\tau)\right]_2$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1$, t, N.

Witness: $\overrightarrow{\mathsf{B}} = \{b_1, b_2, .., b_N\}$, $\overrightarrow{\mathsf{W}} = \{\mathsf{w}_i\}_{i\in[N]\setminus E} \cup \{\mathsf{w}_i = 0\}_{i\in E}$.

Statement: $t = \sum_{i\in N} b_i \cdot \mathsf{w}_i$, where $\left[\overrightarrow{\mathsf{B}}(\tau)\right]_2 = \sum_{i\in[N]} \left[b_i \cdot L_i(\tau)\right]_2$, $\left[\overrightarrow{\mathsf{W}}(\tau)\right]_1 = \sum_{i\in[N]} \left[\mathsf{w}_i.L_i(\tau)\right]_1$.

Prove:$(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}; \overrightarrow{\mathsf{B}}, \overrightarrow{\mathsf{W}})$:

- Computes the partial sum polynomial as

$$\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x) = \sum_{i=1}^{N+1}(\sum_{j=1}^{i-1} b_j \cdot \mathsf{w}_j)L_i(x). \quad (1)$$

  Commit to $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x)$ as $\left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1$.
- To verify $\left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1$ check that:

$$\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x\cdot\omega) - \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x) - (\overrightarrow{\mathsf{W}}(x) - t\cdot L_{n+1}(x))\overrightarrow{\mathsf{B}}(x) \stackrel{?}{=} \mathsf{T}(x)\mathsf{Q}_1(x) \quad (2)$$

  and $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega) = 0$ which is the same as checking

$$L_1(x)\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x) = \mathsf{Q}_3(x)\mathsf{T}(x) \quad (3)$$

  and $\overrightarrow{\mathsf{B}}(\omega^{N+1}) = 1$ which is the same as checking

$$L_{N+1}(x)\cdot(1 - \overrightarrow{\mathsf{B}}(x)) = \mathsf{Q}_4(x)\mathsf{T}(x) \quad (4)$$

- Prove $\overrightarrow{\mathsf{B}}$ is a binary vector compute the quotient polynomial $\mathsf{Q}_2(x)$ such that

$$\overrightarrow{\mathsf{B}}(x)(1 - \overrightarrow{\mathsf{B}}(x)) = \mathsf{T}(x)\mathsf{Q}_2(x), \quad (5)$$

- A random challenge v is generated as

$$v = \mathcal{H}(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}, \left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1)$$

- Batch all the quotient polynomials into one $\mathsf{Q}(x)$:

$$\mathsf{Q}(x) = \mathsf{Q}_1(x) + v\cdot\mathsf{Q}_2(x) + v^2\mathsf{Q}_3(x) + v^3\mathsf{Q}_4(x) \quad (6)$$

  and compute the commitment to it $\left[\mathsf{Q}(\tau)\right]_1$.
- A random challenge r is generated as

$$r = \mathcal{H}(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}, \left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1, \left[\mathsf{Q}(\tau)\right]_1).$$

- Compute the opening proofs of the committed polynomials at the point $x = r$ i.e compute $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r)$, $\overrightarrow{\mathsf{W}}(r)$, $\overrightarrow{\mathsf{B}}(r)$ and $\mathsf{Q}(r)$. Denote the batch opening proof of all the proofs and openings at $x = r$ as open_r.
- Compute the value of $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r\cdot\omega)$ and the opening proof called $\mathsf{open}_{r\cdot\omega}$.
- Output proof $\pi = \{\left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1, \left[\mathsf{Q}(\tau)\right]_1, \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r), \overrightarrow{\mathsf{W}}(r), \overrightarrow{\mathsf{B}}(r), \mathsf{Q}(r), \mathsf{open}_r, \mathsf{open}_{r\cdot\omega}\}$.

Figure 6: Hamming Weight test on committed vectors (Cont. in Fig. 7)

Verify:$(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}, \pi)$:

- Parse the proof π as:

$$\pi = \{\left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1, \left[\mathsf{Q}(\tau)\right]_1, \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r), \overrightarrow{\mathsf{W}}(r), \overrightarrow{\mathsf{B}}(r), \mathsf{Q}(r), \mathsf{open}_r, \mathsf{open}_{r\cdot\omega}\}$$

- Compute challenges r and v as

$$r = \mathcal{H}(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}, \left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1, \left[\mathsf{Q}_1(\tau)\right]_1, \left[\mathsf{Q}_2(\tau)\right]_1, \left[\mathsf{Q}(\tau)\right]_1),$$

$$v = \mathcal{H}(\left[\overrightarrow{\mathsf{B}}\right]_2, \left[\overrightarrow{\mathsf{W}}(\tau)\right]_1, t, N, \mathsf{aux}, \left[\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\tau)\right]_1, \left[\mathsf{Q}_1(\tau)\right]_1, \left[\mathsf{Q}_2(\tau)\right]_1)$$

- Verify open_r and $\mathsf{open}_{r\cdot\omega}$.
- Using Eq. 6 check that the identities (2),(3),(4),(5) satisfy at $x = r$ i.e

$$\mathsf{Q}(r)\cdot\mathsf{T}(r) = (\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r\cdot\omega) - \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r) - (\overrightarrow{\mathsf{W}}(r) - t\cdot L_{n+1}(r))\overrightarrow{\mathsf{B}}(r)) + v\cdot(\overrightarrow{\mathsf{B}}(r)(1 - \overrightarrow{\mathsf{B}}(r))) + v^2(L_1(r)\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(r)) + v^3(L_{N+1}(r)\cdot(1 - \overrightarrow{\mathsf{B}}(r))) \quad (7)$$

- If the above two checks succeed then output 1, else output 0.

Figure 7: Hamming Weight test on committed vectors (Cont. from Fig. 6)

correctness of the scheme in either of the two following cases:

- It provides incorrect hinTS_i in HintsGen() and the aggregator fails to detect it in the Hints-Verify() algorithm (Fig. 3). An incorrect hinTS_i passes the verification check for a single value of γ'. Since γ is chosen at random for each party, we know that $\gamma = \gamma'$ with probability $\frac{1}{\mathbb{F}_p}$.
- it provides an invalid partial signature σ_i in PartialSign() but the aggregator fails to detect it in Step.12. This occurs when $(\left[1\right]_1, \mathcal{H}(m, e))$ and (pk_i, σ_i) does not share the same discrete logs and yet the proof $\pi_i^{\mathsf{eq\text{-}DL}}$ verifies. Such an adversary can be used to break the soundness of NIZK where $\pi_i^{\mathsf{eq\text{-}DL}}$ is the response to the NIZK challenger for the statement $(\left[1\right]_1, pk_i, \mathcal{H}(m, e), \sigma_i)$. Let ϵ^{NIZK} be the soundness error of NIZK. Then the adversary's advantage is upper bounded by ϵ^{NIZK}.

Summing over all $\frac{N}{3}$ corrupt parties, the adversary breaks the correctness of the protocol with probability $\frac{N}{3}(\frac{1}{\mathbb{F}_p} + \epsilon^{\mathsf{NIZK}}) = \mathsf{negl}(\kappa)$ where $\mathbb{F}_p = \mathcal{O}(2^\kappa)$ and $\epsilon^{\mathsf{NIZK}} = \mathsf{negl}(\kappa)$.

Unforgeability: We consider the unforgeability (according to Game 5) of the scheme against a malicious adversary $\mathcal{A}$ that corrupts upto $f(< N)$ parties from the set of N parties. In the unforgeability game, the adversary wins if (1) it manages to corrupt an entire committee, or (2) it forges a signature for a committee where less than threshold partial signatures are known to the adversary.

We analyze the probability of the first event occurring.

There are N signers out of which we assume that f are statically corrupt by the adversary. Assuming the output of RB is unpredictable and $\mathcal{H}$ is a random oracle then the beacon chooses a random committee of n parties. A committee is considered to be completely corrupt if t parties are corrupt. The total number of possible committees is $\binom{N}{n}$, $\binom{n}{r}$ denotes choosing r items without replacements from n items. The probability that there are $X = i$ corrupt parties in a committee is:

$$\Pr[X = i] = \frac{\binom{f}{i} \cdot \binom{N-f}{n-i}}{\binom{N}{n}}.$$

A committee is considered to be corrupt when $i \geqslant t$. The probability of this occurring is:

$$\Pr[X \geqslant t] = \sum_{i=t}^{n} \frac{\binom{f}{i} \cdot \binom{N-f}{n-i}}{\binom{N}{n}}$$

This equation yields a statistical error of 2^{-41} on parameters $N = 1024, f = 341, n = 128$ and $t = 80$ that we use for evaluating our protocol.

Next, we prove that the adversary is unable to forge a signature for an epoch e if less than t partial signatures from the committee Com_e are known to the adversary. To prove unforgeability we prove some helper lemmas and then use them for our proof.

Lemma 3. *Suppose that at the end of the* $\mathsf{Forgery-Game}$ *defined in Figure 5, the adversary $\mathcal{A}$ outputs a message and signature pair (m^*, σ^*) such that $\mathsf{Verify}(m^*, \sigma^*, t, VK) = 1$. Then, with $1 - \mathsf{negl}(\kappa)$ probability, we can extract (multivariate) polynomials $\mathsf{ParSum}_{\overrightarrow{B}}(x)$, $\overrightarrow{B}(x)$, $Q(x)$, $Q_1(x)$, $Q_2(x)$, $X(x, \{sk_i\}_i)$, $X^*_x(x, \{sk_i\}_i)$, $Z(x, \{\mathsf{sk}_i\}_i)$, and $\mathsf{aSK}(x, \{sk_i\}_i)$ from $\mathcal{A}$ such that the following identities hold.*[6]

$$\mathsf{ParSum}_{\overrightarrow{B}}(\omega) = 0, \overrightarrow{B}(\omega^{N+1}) = 1, \quad (8)$$

$$\mathsf{ParSum}_{\overrightarrow{B}}(x \cdot \omega) - \mathsf{ParSum}_{\overrightarrow{B}}(x) - (\overrightarrow{W}(x) - t \cdot L_{n+1}(x))\overrightarrow{B}(x) \stackrel{?}{=} T(x)Q_1(x) \quad (9)$$

$$\overrightarrow{B}(x)(1 - \overrightarrow{B}(x)) = T(x)Q_2(x), \quad (10)$$

$$Q(x) \cdot T(x) = (\mathsf{ParSum}_{\overrightarrow{B}}(r \cdot \omega) - \mathsf{ParSum}_{\overrightarrow{B}}(x) - (\overrightarrow{W}(x) - t \cdot L_{n+1}(x))\overrightarrow{B}(x)) + v \cdot (\overrightarrow{B}(x)(1 - \overrightarrow{B}(x))) + v^2(L_1(x)\mathsf{ParSum}_{\overrightarrow{B}}(x)) + v^3(L_{N+1}(x) \cdot (1 - \overrightarrow{B}(x))) \quad (11)$$

$$\overrightarrow{SK}(x) \cdot \overrightarrow{B}(x) - \mathsf{aSK}(x, \{\mathsf{sk}_i\}_i) = Z(x, \{\mathsf{sk}_i\}_i) \cdot T(x) + X(x, \{\mathsf{sk}_i\}_i) \cdot x, \quad (12)$$

$$X(x, \{\mathsf{sk}_i\}_i) \cdot x = X^*(x, \{\mathsf{sk}_i\}_i). \quad (13)$$

Proof of Lemma 3. The commitments output by the adversary $\mathcal{A}$ in the signature is a linear combination of the group elements that $\mathcal{A}$ takes as input since $\mathcal{A}$ is an algebraic adversary. The output of $\mathcal{A}$ consists of multivariate polynomials depending on $\tau, \{\mathsf{sk}_i\}_{i \in B^*}$, and (discrete log of) $\mathcal{H}(m_i)$.We argue that these equations hold w.r.t. τ.

6. We only treat the sk_i from honest parties as variables. The secret keys of corrupt parties are known to the adversary $\mathcal{A}$.

The opening proofs for the polynomial commitments verify. By Lemma 2, we know that, with $1 - \mathsf{negl}(\kappa)$ probability, $\mathcal{A}$ must know a set of polynomials such that (1) $\mathsf{ParSum}_{\overrightarrow{B}}(\omega) = 0$; (2) $B(\omega^{N+1}) = 1$; and (3) and equation Eq. 7 verifies. By Schwarz-Zippel, with all but $\mathsf{poly}(\kappa)/|\mathbb{F}| = \mathsf{negl}(\kappa)$ probability, the following polynomial identity holds and assuming $\mathcal{H}$ is a random oracle, equations 2, 3, 4, 5 hold.

Furthermore, the following pairing equations hold.

$$e([\overrightarrow{\mathsf{SK}}(\tau)]_1, [[\overrightarrow{\mathsf{B}}_\mathsf{S}(\tau)]_2) \cdot e(\mathsf{aPK}_\mathsf{S}, [1]_2)^{-1} \stackrel{?}{=} e([\mathsf{Z}_\mathsf{S}]_1, [\mathsf{T}(\tau)]_2) \cdot e([\mathsf{X}_\mathsf{S}]_1, [\tau]_2),$$

and

$$\overrightarrow{\mathsf{SK}}(x)\overrightarrow{\mathsf{B}}_\mathsf{S}(x) - \mathsf{aSK}_\mathsf{S} = \mathsf{Z}_\mathsf{S}(x) \cdot \mathsf{T}(x) + \mathsf{X}_\mathsf{S}(x) \cdot x,$$

By Lemma 2, the adversary $\mathcal{A}$ must know polynomials $\overrightarrow{\mathsf{B}}(x) = \overrightarrow{\mathsf{B}}_\mathsf{S}(x)$, $X(x) = \mathsf{X}_\mathsf{S}(x)$, $X^*(x)$, and $Z(x) = \mathsf{Z}_\mathsf{S}(x)$, which satisfy the polynomial identities

$$\overrightarrow{\mathsf{SK}}(x) \cdot \overrightarrow{\mathsf{B}}(x) - \mathsf{aSK} = Z(x) \cdot \mathsf{T}(x) + X(x) \cdot x,$$
$$X(x) \cdot x = X^*(x).$$

Next, we argue that $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x)$, $\overrightarrow{\mathsf{B}}(x)$, $Q_1(x)$, and $Q_2(x)$ are independent of the honest parties' sk_i and $\mathcal{H}(m)$ where m is queried by the adversary for partial signatures. We prove this by contradiction. $\mathsf{T}(x)$ is independent of sk_i. The group elements that $\mathcal{A}$ takes as input are all linearly dependent in sk_i in the exponent. If $\overrightarrow{\mathsf{B}}(x)$ depended on sk_i, then $\overrightarrow{\mathsf{B}}(x) \cdot (1 - \overrightarrow{\mathsf{B}}(x))$ would be a quadratic equation in sk_i, and as a result the equation $\overrightarrow{\mathsf{B}}(x)(1 - \overrightarrow{\mathsf{B}}(x)) = \mathsf{T}(x) \cdot Q_2(x)$ would not hold since $Q_2(x)$ would be quadratic in sk_i, and the adversary would have to compute a quadratic equation (in sk_i) in the exponent given only linear equations (in sk_i) in the the exponent. Both $\overrightarrow{\mathsf{B}}(x)$ and $Q_2(x)$ are single variate polynomials depending only on τ and independent of sk_i, except with negligible probability. By a similar argument, if $Q_1(x)$ depends on sk_i, then $\mathsf{T}(x) \cdot Q_1(x)$ contains a term $\mathsf{sk}_i \cdot \mathsf{T}(x)$. But, all inputs of $\mathcal{A}$ that depend on sk_i, in the exponent, only have degree $\leqslant \mathbb{H} - 1$ in terms of τ. As a result, the well-formedness check (Eq. 2) of $\mathsf{ParSum}()$ would never satisfy. Therefore, both $\mathsf{ParSum}(x)$ and $Q_1(x)$ are single variate polynomials depending only on τ, except with negligible probability. Finally, we argue that $Z(x)$ and $X(x)$ is independent of $\mathcal{H}(m_i)$ since computing $\mathcal{H}(m_i)^\tau$ is computationally hard. Hence, $Z(x)$ and $X(x)$ depends on τ and sk_i. This argument extends to $X^*(x)$ since the last equation holds. □

Lemma 4. *The terms in polynomial $X(x, \{\mathsf{sk}_i\}_i)$ that depends on sk_i has degree $\leqslant |\mathbb{H}| - 2$ in terms of x.*

Proof of Lemma 4. By lemma 3, we know that $X(x) \cdot x = X^*(x)$. All the inputs to the adversary that depends on sk_i have degree $\leqslant \mathbb{H} - 1$ in terms of τ (for example, the terms in hinTS_i). In particular, the terms in $X^*(x)$ that depends on sk_i have degree $\leqslant \mathbb{H} - 1$ in terms of τ. Hence, the terms in

$X(x, \{\mathsf{sk}_i\}_i)$ that depends on sk_i will have degree $\leqslant |\mathbb{H}| - 2$ in terms of τ. □

Lemma 5. *For polynomial $\overrightarrow{B}(x)$, it must hold that $\sum_{i=1}^{N} \overrightarrow{B}(\omega^i) \cdot w_i = t$.*

Proof of Lemma 5. By lemma 3, we have the polynomial identity $\overrightarrow{\mathsf{B}}(x) \cdot (1 - \overrightarrow{\mathsf{B}}(x)) = \mathsf{T}(x) \cdot Q_2(x)$. At $x \in \mathbb{H}$, $\mathsf{T}(x) = 0$ since it is the vanishing polynomial on $\mathbb{H}$. Hence, $\overrightarrow{\mathsf{B}}(x)$ is either 0 or 1 on $x \in \mathbb{H}$. By lemma 3 we know that $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega) = 0$ and $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x \cdot \omega) - \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x) - (\overrightarrow{\mathsf{W}}(x) - t \cdot L_{N+1}(x)) \cdot \overrightarrow{\mathsf{B}}(x) = \mathsf{T}(x) \cdot Q_1(x)$ hold. Moreover, $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega \cdot x) - \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(x)$ is $b_i \cdot w_i$ everywhere (by plugging values in Eq. 2), except for $x = \omega^{N+1}$ where it is $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega^{N+2}) - \mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega^{N+1}) = 0 - t$ (from Eq. 1). By summing over for all $x \in \mathbb{H}$, we get $\mathsf{ParSum}_{\overrightarrow{\mathsf{B}}}(\omega^{N+1}) = \sum_{i=1}^{N} \overrightarrow{\mathsf{B}}(\omega^i) \cdot w_i = t$.

□

Lemma 6. *Let the set B^* and A be as defined in the unforgeability game, if the signature σ' verifies under aPK, $e(\mathsf{aPK}, \mathcal{H}(m)) = e([1]_1, \sigma')$, then, with $1 - \mathsf{negl}(\kappa)$ probability, the equation $\mathsf{aSK}(\{\mathsf{sk}_i\}_i) = \sum_{i \in B^*} \mathsf{sk}_i \cdot v_i + v_0$ holds for some v_i's.*

Proof of Lemma 6. We know that the signature verifies $e(\mathsf{aPK}, \mathcal{H}(m)) = e([1]_1, \sigma')$, hence, $\sigma' = \mathcal{H}(m)^{\mathsf{aSK}}$. Now, the group elements that the adversary sees and are related to $\mathcal{H}(m)$ are $\mathcal{H}(m)^{\mathsf{sk}_i}$ for $i \in B^*$ and $\mathcal{H}(m)^{\mathsf{sk}_i}$ (returned as output from the signing oracle). Hence, this lemma follows from lemma 2. This lemma states that aSK depends only on the honest parties' sk_i's that have signed m, and does not depend on τ or other honest parties' sk_i. □

Next, we prove the unforgeability of our scheme given lemmas 3, 4, 5, and 6. Suppose that the adversary $\mathcal{A}$ wins the unforgeability game. By definition, there exists a threshold $t > |(B^* \cup A) \cap \mathsf{Com}_e|$ such that $\mathsf{Verify}(m^*, \sigma^*, t, \mathrm{VK}, e) = 1$.

Since the adversary's signature verifies, lemma 3 states

$$\overrightarrow{\mathsf{SK}}(x) \cdot \overrightarrow{\mathsf{B}}(x) - \mathsf{aSK}(x, \{\mathsf{sk}_i\}_i) = Z(x, \{\mathsf{sk}_i\}_i) \cdot \mathsf{T}(x) + X(x, \{\mathsf{sk}_i\}_i) \cdot x,$$

And, $\mathsf{aSK} = \sum_{i \in B^*} \mathsf{sk}_i \cdot v_i + v_0$ by lemma 6.

Now, we extract the set $B' = \{i \in [N] : \overrightarrow{\mathsf{B}}(\omega^i) = 1\}$ from $\overrightarrow{\mathsf{B}}(x)$. Let $\mathsf{aSK}' = (\sum_{i \in B'} \mathsf{sk}_i)/|\mathbb{H}|$. There also exists an honestly sampled quotient polynomial $X'(x, \{\mathsf{sk}_i\}_i)$ and $Z'(x, \{\mathsf{sk}_i\}_i)$ (for aSK') satisfying

$$\overrightarrow{\mathsf{SK}}(x) \cdot \overrightarrow{\mathsf{B}}(x) - \mathsf{aSK}' = Z'(x, \{\mathsf{sk}_i\}_i) \cdot \mathsf{T}(x) + X'(x, \{\mathsf{sk}_i\}_i) \cdot x.$$

The adversary can efficiently compute Z' and X'. Subtracting the two equations we get

$$\mathsf{aSK}' - \mathsf{aSK} = (Z - Z') \cdot \mathsf{T}(x) + (X - X') \cdot x.$$

To forge a signature, the adversary computes two polynomials $\Delta_X(x, \{\mathsf{sk}_i\}_i)$ and $\Delta_Z(x, \{\mathsf{sk}_i\}_i)$ such that

$$\mathsf{aSK}' - \mathsf{aSK} = \Delta_Z(x, \{\mathsf{sk}_i\}_i) \cdot \mathsf{T}(x) + \Delta_X(x, \{\mathsf{sk}_i\}_i) \cdot x. \quad (14)$$

Since the total weight in B' is t (due to lemma 5), which satisfies $t > \sum_{i \in t > |(B^* \cup A) \cap \mathsf{Com}_e|} w_i$ (definition of forgery), there must exist some honest party with index j ($\in \mathsf{Com}_e$) such that $j \in B'$ and $j \notin B^*$, i.e. honest party j is in the committee and the adversary forged its partial signature. Therefore,

$$\begin{aligned}\mathsf{aSK}' - \mathsf{aSK} =& \Big(\sum_{i \in B'} \mathsf{sk}_i\Big)/|\mathbb{H}| - \Big(\sum_{i \in B^*} \mathsf{sk}_i \cdot v_i + v_0\Big) \\ =& \mathsf{sk}_j/|\mathbb{H}| + L(\{\mathsf{sk}_i\}_{i \neq j}).\end{aligned}$$

where $L(\cdot)$ is affine combination of $\{\mathsf{sk}_i\}_{i \neq j}$ whose coefficients depend on $(\{v_i\}_{i \in B^*}, v_0)$.

We argue that Eq. 14 implies a contradiction: By Lemma 4, the terms in $\Delta_X(x)$ that depends on sk_j have degree $\leqslant |\mathbb{H}| - 2$. Therefore, the terms in $\Delta_Z(x) \cdot \mathsf{T}(x) + \Delta_X(x) \cdot x$ that depends on sk_j can never be $c \cdot \mathsf{sk}_j$ for some constant c. In particular, it can never equal $\mathsf{sk}_j/|\mathbb{H}|$. Thus, the following equation will not hold

$$\mathsf{aSK}' - \mathsf{aSK} = \Delta_Z(x, \{\mathsf{sk}_i\}_i) \cdot \mathsf{T}(x) + \Delta_X(x, \{\mathsf{sk}_i\}_i) \cdot x$$

This concludes that the adversary wins the unforgeability game with negligible probability.

Appendix F. Comparison of Reconstruction Threshold vs Corruption Threshold

In this section, we analyse the relationship between the minimum reconstruction threshold and the corruption threshold for BLS multi-signatures, hinTS, and Dyna-hinTS. We repeat this experiment for varying number of total signers $N = 1000, 2000$ and 4000 in Fig. 8, 9 and 10 respectively. Assuming f among N signers are corrupt, the minimum reconstruction threshold t in Dyna-hinTS is computed via the following formula:

$$\text{The minimum value of } t \text{ s.t. } \frac{\binom{f}{t}}{\binom{N}{t}} \leqslant 2^{-40}.$$

Whereas, for BLS multi-signatures and hinTS, $t = f + 1$. As a result, t grows slowly in Dyna-hinTS yielding a massive boost in aggregation time when a larger fraction of of signers are corrupt.

Appendix G. Setup Cost

We evaluate the fixed costs of running the setup for each protocol. We assume that there is a network delay of $120ms$ when the signers send their public keys to the aggregator or post them on the blockchain. In BLS multisignature, each signer samples a public and secret key pair and broadcasts the public keys. These public keys are stored in a smart contract on the blockchain. Sampling the key pair is very fast since it takes less than $< 1ms$ and the signers broadcast it incurring a network delay of $120ms$.

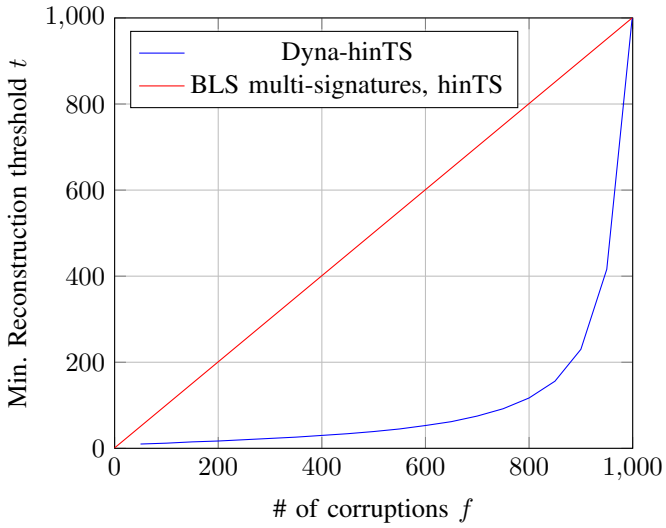

Figure 8: Comparison of (f vs t) for Dyna-hinTS, BLS multisignatures and hinTS when total # of signers N = 1000

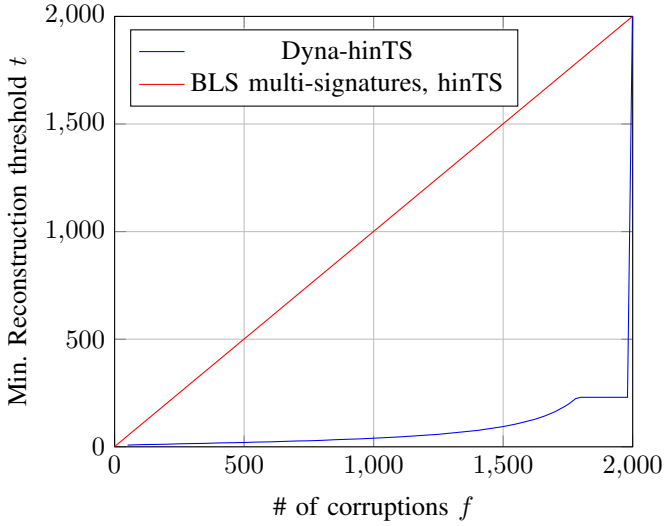

Figure 9: Comparison of (f vs t) for Dyna-hinTS, BLS multisignatures and hinTS when total # of signers N = 2000

hinTS and Dyna-hinTS both use a silent setup, where each signer together with their public keys broadcast a linear-sized auxiliary material (informally called the hints), whose computation takes $O(n)$ group multiplication and $O(n \log n)$ field operations. Then the Aggregator verifies all such hints sent by the signers in the universe. This step consists of two subprotocols. Firstly, the aggregator needs to compute KZG commitments to $O(n^2)$ Lagrange coefficient polynomials. Then using these polynomial commitments, the aggregator verifies the hints of each signer by performing random linear combinations using Pippenger's algorithm [18] and then performs a pairing check. We benchmark this extensively

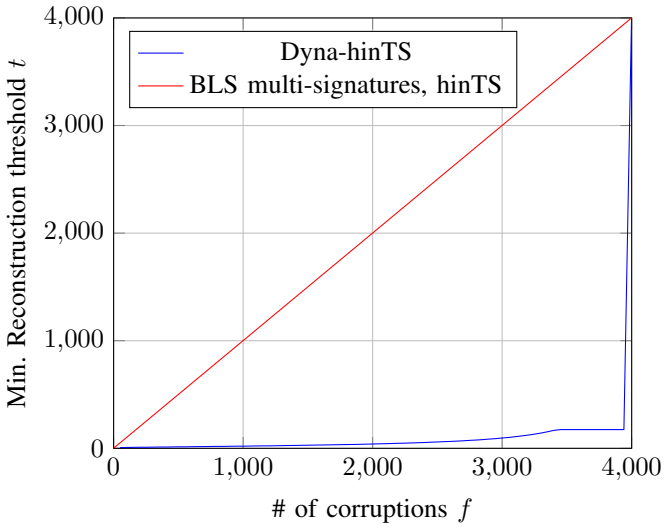

Figure 10: Comparison of (f vs t) for Dyna-hinTS, BLS multisignatures and hinTS when total # of signers N = 4000

and observed that it takes 10 sec for each index i on a single-threaded implementation. The Aggregator then performs one-time preprocessing using the public keys and hints to generate the aggregated public verification key VK and the additional aggregation key AK. The verification key VK consists of 3 group elements and it is posted on the blockchain, costing 390K gas. The setup need not be repeated for changes in the access structure and can be performed once considering networks of different powers of two sizes.

Optimization: We extensively benchmarked the setup cost for different values of N since it is the main bottleneck in the paradigm of silent threshold signatures and it affects Dyna-hinTS, hinTS [44], and [30]. We observe that for 1024 signers the key generation phase, the hints verification, and the generation of the public verification key are extremely fast and take around 30 seconds for all 1024 signers. However, this accounts for less than 1% of the cost. Whereas, 99% of the setup time is spent in computing the KZG commitments to the Lagrange coefficients. We propose how to delegate this computation to untrusted parties such that they can be verified very efficiently. In Fig. 3, the aggregator needs to compute KZG commitments to the Lagrange coefficients for $i \in [N]$.

$$\left[L_i(\tau)\right]_2, \left[\tfrac{L_i^2(\tau)-L_i(\tau)}{\mathsf{T}(\tau)}\right]_2, \left[\tfrac{L_i(\tau)-L_i(0)}{\tau}\right]_2, \left[(L_i(\tau)-L_i(0))\right]_2, \tag{15}$$

and the cross-term commitments

$$\left\{\left[\tfrac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\right]_2\right\}_{j\in[N]\setminus i}.$$

The computation of the first four types of commitments is very fast since it involves computing only 4 commitments for each i and requires around 16 ms for each signer. This is done by the aggregator. Next, the computation of the cross-term commitments requires around 737 ms for each

signer and accounts for 98% of the total time as it grows quadratically with the number of signers.

We reduce this cost by delegating its computation to untrusted parties. To do this, the aggregator locally computes $\left[\frac{L_1(\tau)}{\mathsf{T}(\tau)}\right]_1, \ldots \left[\frac{L_N(\tau)}{\mathsf{T}(\tau)}\right]_1$ and reuses $\left[L_1(\tau)\right]_2, \ldots \left[L_N(\tau)\right]_2$ from the above computation. The aggregator asks an untrusted party to compute the cross-term commitments for index i. When the untrusted party returns $\left[\frac{L_i(\tau)L_1(\tau)}{\mathsf{T}(\tau)}\right]_2, \ldots \left[\frac{L_i(\tau)L_N(\tau)}{\mathsf{T}(\tau)}\right]_2$, the aggregator samples a $\gamma \in \mathbb{F}$ and verifies it using random linear combination (via Pippinger's method) and a pairing check as

$$e\Big(\left[\tfrac{L_i(\tau)}{\mathsf{T}(\tau)}\right]_1, \Pi_{j\in[N]\setminus i}\left[L_j(\tau)\right]_2^{\gamma^j}\Big) \stackrel{?}{=} e\Big(\left[1\right]_1, \Pi_{j\in[N]\setminus i}\left[\tfrac{L_i(\tau)L_j(\tau)}{\mathsf{T}(\tau)}\right]_2^{\gamma^j}\Big) \quad (16)$$

Performing this verification is fast and takes around 15 ms for each party. In total, we reduce the cost to around 35 ms [7] from 737 ms, getting a 21× improvement.

Assuming there are M untrusted parties then the aggregator delegates the computation of the cross-terms of indices $i = 1 \ldots \frac{N}{M}$ to the first party, indices $i = \frac{N}{M} + 1 \ldots \frac{2N}{M}$ to the second party, and so on where the jth untrusted party computes the cross-term commitment for indices $i = \frac{(j-1)N}{M} + 1 \ldots \frac{jN}{M}$. Once the untrusted parties return the cross-term commitments, the aggregator verifies them using a random linear combination (via Pippinger's method) and a pairing check according to Eq. 16.

After performing the above optimizations, we obtain the total setup cost for $N = 1024$ signers as 66 seconds. We provide the detailed cost analysis of our optimized setup protocol in Fig. 11.

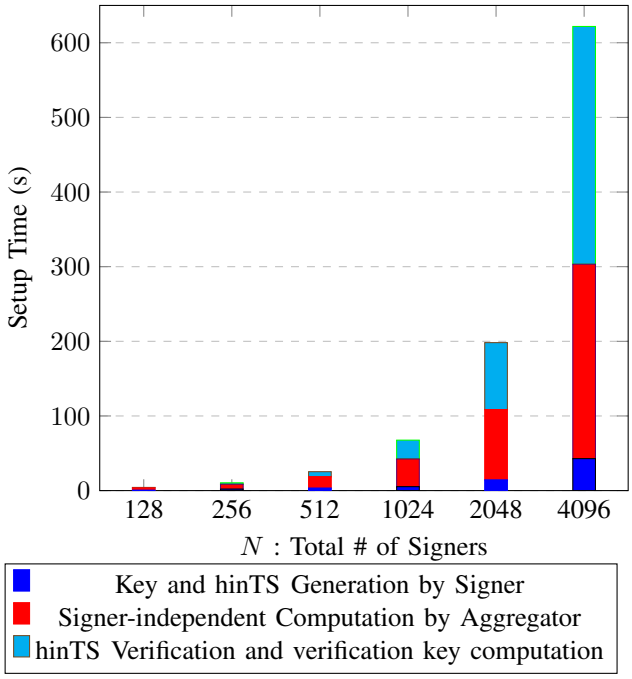

Figure 11: Cost Analysis of our Setup Protocol for varying number of signers.

Appendix H. Further Extensions

- **General Access Structure:** Dyna-hinTS supports general access structure (such as weighted). The SNARK proof in σ should convince the verifier that the set of signers are authorized according to the general access structure, and belong to the same committee. To perform the proof, the aggregator will use the SNARK proof to prove that the witness encoded in $\overrightarrow{\mathsf{B}}(x)$ satisfies the access structure Λ, where Λ is represented as a circuit $C : \{0,1\}^N \to \{0,1\}$, and the aggregator proves that $C\big(\overrightarrow{\mathsf{B}}(\omega), \ldots, \overrightarrow{\mathsf{B}}(\omega^N)\big) = 1$.
- **Dynamic Threshold:** Dyna-hinTS can support dynamic threshold more efficiently than the above general access structure approach. To accommodate this change, there is no change in the setup, the committee selection or the partial signature protocols. The only change that occurs is in the signature aggregation protocol, where the aggregator generates the SNARK proof w.r.t. the threshold t_j for each message m_j. This supports different thresholds even in the same committee.
- **Silent Dynamic Participation:** Another useful requirement could be silent dynamic participation, where each signer can dynamically join/leave the system of signers in Dyna-hinTS. In any STS scheme (including ours and hinTS) the set of signers can generate hints for a large value of N and send these to the aggregator. Suppose there are $N' < N$ signers. When a new signer joins, the aggregator assigns the index $N' + 1$ to that signer. The new signer generates the hints for value N and sends these to the aggregator. The aggregator updates the public verification key. The existing set of signers does not need to perform any computation/communication due to the silent setup. When a party leaves the system, the aggregator sets the index of that signer as null and includes it in the set E. The public verification key does not need to be updated. Later when a new signer joins, the aggregator assigns this empty index to that signer. The system can support up to N signers. When the number of signers exceeds N, the existing set of signers needs to compute additional hints for the new indices after N.
- **Multiverse Threshold Signatures:** Dyna-hinTS can be extended to construct an MTS scheme. Each committee is considered as an universe U where the universe U is chosen by the verifier and the reconstruction threshold is t_U. When the universe is chosen, the aggregator uses the crs to compute the KZG commitment $\left[\overrightarrow{\mathsf{B}}_U(\tau)\right]_2$ to universe U. This is the verification key for the universe U. It can be verified by the verifier by recomputing the KZG commitment. This is a non-interactive setup process for each universe, whereas the MTS scheme of

7. Roughly $20 \times N$ ms is spent on computing the $4N$ commitments in Eq. 15 and the N commitments used in Eq. 16.

[16] requires interaction between the signers and the aggregator. To sign a message m, each signer generates the signature as $\sigma_i = \mathcal{H}(m)^{\mathsf{sk}_i}$ and a proof $\pi_i^{\mathsf{eq\text{-}DL}}$ of correct signing. The aggregator verifies the partial signatures and aggregates t_U valid partial signatures from signers in universe U. Let $\left[\overrightarrow{\mathsf{B}}_{\mathsf{S}}(\tau)\right]_2$ denote the KZG commitment to the signing set in U. The aggregator computes a proof that t_U signers from universe U have signed, i.e. $\mathsf{S} \subseteq U$ by proving $\left[\overrightarrow{\mathsf{B}}_{\mathsf{S}}(\tau)\right]_2$ has Hamming weight t_U and $\frac{\left[\overrightarrow{\mathsf{B}}_{U}(\tau)\right]_2}{\left[\overrightarrow{\mathsf{B}}_{\mathsf{S}}(\tau)\right]_2}$ has weight $|U| - t_U$.