2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00057

hinTS: Threshold Signatures with Silent Setup

Sanjam Garg*‡, Abhishek Jain†‡, Pratyay Mukherjee§,
Rohit Sinha¶, Mingyuan Wang*, Yinuo Zhang*

* *University of California, Berkeley* † *Johns Hopkins University* ‡ *NTT Research* § *Supra* ¶ *Swirlds Labs*

***Abstract*—We propose hinTS — a new threshold signature scheme built on top of the widely used BLS signatures. Our scheme enjoys the following attractive features:**

- **A *silent setup* process where the joint public key of the parties is computed as a deterministic function of their locally computed public keys.**
- **Support for *dynamic* choice of thresholds and signers, after the silent setup, without further interaction.**
- **Support for *general* access policies; in particular, native support for *weighted* thresholds with zero additional overhead over standard threshold setting.**
- **Strong security guarantees, including proactive security and forward security.**

We prove the security of hinTS in the algebraic group model, and also provide an open-source implementation. Our scheme outperforms all prior proposals that avoid distributed key generation in terms of aggregation time, signature size, and verification time (as well as other qualitative measures). As an example, the aggregation time in hinTS for 1000 signers is under 0.5 seconds, while both signing and verification are constant time algorithms, taking 1 ms and 17.5 ms, respectively.

The key technical contribution of our work involves the design of special-purpose succinct proofs to *efficiently* prove the well-formedness of aggregated public keys. Our solution uses public "hints" released by the signers as part of their public keys (hence the name hinTS).

1. Introduction

Threshold signatures [32], [33] allow for distributing the signing process among a group of (say) n parties such that any subset of parties of size greater than a threshold (say) $t \leqslant n$ can create valid signatures on any message. Furthermore, any subset of parties of size at most t cannot forge signatures.

First introduced in the 1980s, threshold signatures [32], [33] have found broad appeal in recent years in decentralized systems such as blockchains, where they are used in state proofs [34], [29], oracle networks [37], crypto-wallets [9], distributed randomness services [8], cross-chain bridges [64], [62], [56], [58], and even in scaling byzantine consensus protocols [65]. This is now a mature research area, with lots of known constructions built on top of popular signature schemes such as BLS [21], Schnorr [60] (and its variant EdDSA [15]), and EC-DSA [28]. Moreover, attempts to standardize threshold signature have already begun [24].

By and large, all known designs of threshold signatures adopt the following schemata:

- First, a threshold t is chosen, and the parties run a distributed key generation (DKG) protocol that generates a public key pk and signing key sk and outputs a secret share sk_i of the signing key sk to party i.
- To sign a message, the parties run an (ideally, *one-round*) protocol to produce partial signatures.
- Finally, an aggregator "combines" the partial signatures into a "short" signature that can be verified with respect to pk.

While this model has found success over the years, in our experience, it is not sufficiently well-equipped to handle the demands of a growing, diverse set of scenarios where threshold signatures are now being considered, such as in blockchains. In a fully decentralized setting, limitations arise from the need for a DKG (which involves expensive multiparty protocols), the inability to choose the signing threshold or the signers after system setup (without a fresh DKG), and the lack of efficient extensions to more expressive access structures, such as weighted structures.

Our Model. In what follows, we expand upon our goals for threshold signatures. Some of these goals have recently been investigated elsewhere (see Section 1.3 for a discussion); our aim, however, is to achieve all of the goals simultaneously, while building *concretely efficient* solutions with a low deployment barrier.

I. Silent Setup. A major hurdle in deploying fully decentralized threshold signatures is the requirement of performing a DKG – a multiparty protocol that incurs high computation and communication costs and uses expensive primitives such as broadcast channels. Indeed, extensive literature is dedicated to optimizing DKG protocols along various axis (see Section 1.3). Despite many advances, DKG continues to be a major source of engineering and deployment challenges (see, e.g., [36], [30]) for threshold signatures. Though a DKG need only be performed just once and then used

DOI 10.1109/SP54263.2024.00057

repeatedly for signing, executing it even once becomes extremely challenging in some settings. For instance, in Ethereum 2.0, a state-proof is threshold-signed by two-thirds fraction of a committee of 512 validators (called the sync committee); it is both expensive and complicated to run an interactive DKG protocol for a new committee that is randomly selected every (approximately) 27 hours. Instead, Ethereum 2.0 resorts to a multi-signature scheme, that does not need a DKG, but requires the verifier (e.g. a smart contract running on a different ledger) to learn all 512 public keys[1], as multi-signatures only support n out of n access structures.

Therefore, our work starts with a natural question: *Can we construct a threshold signature without DKG, or with a silent setup?* In a silent setup, parties *do not exchange any message* with each other to generate keys. Instead, each party uses a local key-generation procedure to establish its key pair and the joint public key of a set of signers is computed as a deterministic function of their individual public keys.[2] This allows for an asynchronous setup, as parties *silently* join the system and publish their public keys without waiting for others. For the aforementioned Ethereum 2.0 application, it provides an attractive choice compared to the currently used multisignature, as silent threshold signatures offer the best of both worlds: like multisignatures, it allows for dynamic access structures and no DKG, and like threshold signatures, it has compact (and efficient) verification. More use cases are discussed in Section 1.2.

II. Dynamism. The setup phase (realized by a DKG protocol) in existing threshold systems *fixes* both the threshold and the set of signers. This rules out applications that require a *dynamic choice* of thresholds or signers (or both) at a later point (see Section 1.2).

This leads to our next question: *Can we build a threshold signature scheme that supports a dynamic choice of thresholds and signers?*[3] A näive solution is to simply run a fresh instance of a static threshold signature scheme for each choice. This, however, is not a scalable solution since the work and secret storage required of each signer increases with every new choice. We, instead, aim to achieve dynamism without increasing the work of the signers (relative to the static model).

1. The public keys of the sync committee can be derived from on-chain data, and hence, common solutions for state proofs often use SNARK-based mechanisms to prove the validity of all 512 public keys with respect to the signed headers.

2. This is conceptually similar to multisignatures [19], the n-out-of-n (i.e., full threshold) variant of threshold signatures, that do not require a DKG. However, multi-signatures do not achieve *liveness*, i.e., they do not support signature computation even if just one party is absent. See Section 1.3 for further discussion.

3. While DKG fixes the threshold during system setup in existing systems, getting rid of DKG does not automatically imply dynamism. It might be possible to achieve silent setup (i.e., no DKG) without dynamism. In this work, we aim to achieve *both* properties.

III. General Policies. Prior works primarily focus on threshold access structures in that all signers are treated as equals, and their cardinality determines an authorized set of signers. However, in stake-based blockchains [51] and oracle networks [37], all signers are not equal. Instead, each signer has an associated weight, and the access structure is defined by a weighted threshold. More generally, each signer could have different "attributes" and a circuit computed over the attributes might define an access structure.

We ask: *Can we build threshold signatures that support general access structures?* In principle, existing schemes that rely on linear secret-sharing [61] can be modified to support access structures for which linear secret-sharing schemes exist. This, however, does not (in general) result in efficient constructions, and limits the access structures that can be realized. We devise an alternative approach that is not limited by linear secret-sharing and incurs low performance overhead relative to the standard threshold setting. Motivated by the aforementioned applications, we put a special emphasis on *native* support for weighted thresholds such that our scheme suffers *no* performance overhead from the incorporation of high-precision (say 64-bit) weights.

IV. Enhanced Security. In addition to the standard unforgeability security of threshold signatures, we aim for two important and well-studied security properties — proactive security [53] and forward security [14]. Proactive security requires a periodic key refresh to ensure that even an adversary with unlimited corruption budget cannot compromise security as long as the total number of corruptions in any period does not exceed the threshold. On the other hand, forward security requires that a key compromise does not violate past periods' security. These properties are well-studied in the literature and are important for real-world deployments.

V. Concrete Efficiency. We aim for a *one-round* signing procedure and disallow any additional pre-processing rounds of interaction. Only threshold versions of the BLS scheme are known to satisfy this property. As such, we set threshold BLS with a DKG setup (such as [63]) as our baseline. We require that our signing, aggregation, and verification algorithms have comparable efficiency, concretely and asymptotically, to threshold BLS.

1.1. Our Contributions

1) Silent Threshold Signatures. We propose the notion of *silent threshold signatures* (STS) where the setup phase requires no communication between the parties. Each party runs a local key-generation and publishes its public key and the joint public key of the signers is deterministically derived from all of their public keys.

STS supports a dynamic choice of thresholds. Namely, for every message m, it is possible to choose

a different security threshold t without modifying the joint public key. Our main efficiency requirements are that the joint public key size, the (aggregated) signature size, and the verification time complexity should all be constant (i.e., independent of the threshold and the number of parties), while the aggregation time complexity is similar to that of standard threshold signatures.

2) hinTS — Our Construction of STS. We construct hinTS — a silent threshold signature scheme in the common reference string (CRS) model building on the BLS signature scheme and Plonk argument system [40]. The CRS in our scheme is the same as in the widely-used KZG commitment scheme [47].[4] Our base scheme natively supports weighted threshold access structures at no additional cost over the standard threshold case. We prove the unforgeability security of our scheme in the algebraic group model [39].

We show that our base scheme can be readily extended to achieve several attractive features discussed earlier: support for general access structures (described by circuits), proactive security, and forward security, as well as support for the recently introduced multiverse model [12]; without significant performance penalty.

3) Implementation and Evaluation. We provide an open-source implementation of hinTS, which we evaluate on networks with thousands of parties. Signing is exactly the same as BLS, and takes roughly 1 ms (regardless of the signer's weight). For 1000 signers, the aggregation time is under 0.5 seconds. Verification time is constant, which we measure to be 17.5 ms. When verifying signatures on-chain, we measure the EVM gas cost to be 395K – roughly 2.5x compared to threshold BLS. We find this moderate increase in gas cost to be an acceptable trade-off considering the other benefits offered by our scheme over threshold BLS. For instance, with weights of precision 10 bits, aggregation time in threshold BLS (using [63]) with 1000 signers takes 46 seconds, and simply becomes infeasible for higher precision weights.

1.2. Applications

Our wishlist of properties is inspired by the following set of applications in the blockchain ecosystem.

State Proofs. In a byzantine fault tolerant system for state machine replication, the parties participating in consensus periodically "checkpoint" their latest state as they reach consensus on new transactions. However, for an external party or processor – such as a smart contract running on another ledger – to safely act on these state transitions (a feature often referred to as interoperability), the consensus participants produce a *state proof*, which is implemented on some blockchains (e.g., Dfinity, Ethereum) as a threshold signature.[5]

Ethereum has over half a million validator nodes. It is infeasible to perform a DKG protocol across them all. Consequently, Ethereum 2.0 designed a protocol [1] that periodically selects a randomly-sampled subset of 512 validators (called the *sync committee*) who compute a threshold signature (with two-thirds threshold) for each block that reaches consensus; however, they avoid a DKG for even this subset, and instead use BLS multisig, with the obvious downside that verifying this multisig requires the verifier to learn all 512 public keys (see Section 1.3 for further discussion on the multisignature approach). Our proposed scheme hinTS targets this very problem, as it allows state proofs to be generated with a silent setup – which only uses the public key published by each validator when it joined the Ethereum network – and requires the verifier to only learn a compact verification key comprising a few group elements. Moreover, the threshold signature can be weighted, by assigning each node a weight based on its stake.

Oracle Networks. Oracles enable smart contracts to perform transactions based on off-chain data, such as issuing DeFi transactions based on the exchange rate of tokens. In Chainlink [37], [25], whenever the data feed's value (e.g., MKR/ETH exchange rate) fluctuates beyond a limit, the oracle nodes collectively agree on a new value to submit on-chain along with a threshold signature. As discussed in [12], smart contracts have the ability to configure the subset of oracle nodes they will accept the threshold signature from (based on reputation scores, for instance), and the signing threshold – for instance, security-critical applications such as automated collateral auditing (via proof-of-reserve) can set different thresholds compared to gaming contracts.

It is unreasonable for the oracle nodes to perform a fresh DKG for each downstream smart contract. It increases complexity – nodes would have to store a secret key from each DKG, and sign messages with each of those keys – and is prohibitively expensive in terms of bandwidth over a peer-to-peer network, as evaluated in [12]. hinTS addresses this shortcoming by only requiring the oracle nodes to publish their public keys once, yet allowing for any smart contract developer to later choose their own subset of oracle nodes that they trust (along with reputation-based weights and a signing threshold) and compute a verification key for the smart contract using those nodes' public keys; the oracle nodes are not involved at all, and they sign data feeds independent of the downstream smart contracts.

Off-chain DAO Voting. A decentralized autonomous organization (DAO) enables governance of assets based

4. Multiparty "ceremony" protocols for computing this reference string are well-known and widely used in practice [52].

5. For blockchains that do not implement state proofs, there are several intermediary networks, called bridges [64], [62], [56], [58], who forward checkpointed state (hashes) across chains – these bridges commonly compute threshold signatures amongst the bridge nodes.

on smart-contracts, where decisions are made by members voting on proposals or decisions. Voting systems for DAOs come in two flavors: 1) on-chain voting: DAO members cast votes on-chain; 2) off-chain voting: DAO members submit votes to one or more off-chain aggregator nodes, but the DAO's leadership team, who controls a multisig wallet, casts a single on-chain transaction with the decision. The latter option is clearly more efficient in terms of gas cost, but it comes at the cost of increased trust assumptions. To address this gap, there is a general interest in the blockchain ecosystem (e.g., [2], [7], [6]) to study SNARK-based aggregation of votes, which would be a best-of-both-worlds solution enabling off-chain voting without centralized trust.

Unfortunately, as proving statements about signature verification require expensive encoding of group operations within a SNARK circuit, off-the-shelf SNARKs simply do not scale (see [6], [2]), as the aggregation time for a few thousand votes is on the order of hours and requires a machine with 100s of GB of RAM. Looking beyond off-the-shelf SNARKs, it is unimaginable to expect thousands of DAO members to come online at the same time and perform a DKG ceremony; this is especially problematic because DAOs allow a rolling membership where parties join or leave at will. Not only does hinTS enable a silent setup, but it also allows the proposal to dynamically set the threshold after the voting period has commenced.[6] Most importantly, the voters get assigned a weight based on their stake or token allocation in the DAO, and more complex policies (based on leadership roles or other attributes, for example) can also be enforced by hinTS.

1.3. Related Work

DKG Protocols. There is a large literature on DKG protocols spanning four decades. Despite the recent advancements, DKG protocols have high computation and/or communication complexity. The Joint-Feldman style DKG protocols [42] require $O(n^2)$ computation per party when parties behave honestly, but balloons up to $O(n^3)$ computation in the event of complaints – Tomescu et al. [63] improve these costs to $O(n\log(n))$ and $O(n^2\log(n))$ respectively. Recent works – Abraham et al. [10], Kokoris et al. [48], and Das et al. [31] – have made DKG operate under asynchrony, but they nevertheless require $O(n^2)$ or $O(n^3)$ computation – more importantly, they only support corruption threshold of $t < n/3$, which is quite restrictive for many applications. These DKGs also require a broadcast channel, which needs several rounds of communication.[7]

A very recent line of work [45], [46], [43] has started investigating one-round DKG protocols where each party only sends one message to everyone else. This round of interaction requires an initial PKI setup and fixes the security threshold. In other words, in order to change the threshold, the parties must communicate again. In contrast, our scheme achieves a silent setup (in the CRS model) where the parties only publish their locally computed public key (i.e., they only setup a PKI), and the threshold can be decided dynamically without any interaction from the parties.

Prior approaches to avoiding DKG. We now discuss known approaches to build threshold signatures without DKG – all of them achieve silent setup and support dynamic thresholds. The first approach (abbrev. Multisig) relies on multisignatures [13], [18], a variant of threshold signatures that supports non-interactive key aggregation in the n-out-of-n (i.e., full threshold) setting. In order to support thresholds t strictly less than n, one can simply offload the task of public-key aggregation to the verifier. More specifically, the signature aggregator now outputs a signature that includes the list of all signers and the verifier aggregates the public keys of the signers in the list on-the-fly. While simple, this approach results in signature size and verification time *linear* in the number of signers. This can be prohibitively expensive in many applications, e.g., to setup a smart contract for 2000 signers, this approach requires around 60 million gas (see Table 5 in Section 7 for detailed comparison).

A natural idea to avoid the linear complexity of the previous approach is to offload the public-key aggregation work to the signature aggregator. That is, the signature aggregator now also computes the aggregated public key of the threshold number of signers who participated in the signing process, and outputs this key as part of the signature. But how do we know that the aggregated public key is honestly computed? To solve this dilemma, we require the aggregator to output a succinct non-interactive argument of knowledge (SNARK) [17], [16] that establishes the well-formedness of the aggregated public key. This approach yields an STS scheme, but in general, incurs prohibitively high concrete aggregation costs. For example, even using state-of-the-art SNARKs (such as PLONK [40]) and the most efficient signature schemes such as EdDSA [15] results in aggregation time in the order of a few minutes for modest choices of parameters – in comparison our aggregation requires less than a second (see Table 2 in Section 7). Moreover, even such performance numbers are only possible when we instantiate EdDSA with SNARK-friendly hash functions such as MiMc [11], whose security is not well-understood.

A recent work of Micali et. al. [50] (Comp-Cert for short) devises a customized instantiation of the above SNARK-based approach to achieve smaller aggregation times – they call it *compact certificates*. Similar to our

6. DAO proposals typically only get votes from a small fraction of the members, so it is typical for the DAOs to choose different thresholds for different proposals based on their importance.

7. The exception here is Abraham et al. [10], but their scheme produces a group element rather than a field element as the secret, making it incompatible with known threshold signature constructions.

Approach	Signature Size & Verifier Time	Policy
MultiSig	$O(n_s)$	Weighted
Comp-Cert [50]	$O(\mathsf{poly}(\log(n_s)))$	Weighted
hinTS	$O(1)$	**General**

TABLE 1: Comparison between our scheme and other two approaches that achieve silent setup and dynamism and has comparable efficiency (the general SNARK approach has prohibitively high aggregation time and hence is excluded). n_s denotes the number of signers. For a more detailed comparison and concrete efficiency, see Section 7.

work, they support the weighted-threshold setting.We highlight a few drawbacks of their scheme relative to ours: first, their scheme requires a larger signature size (logarithmic in the number of signers) and higher verification cost (also logarithmic), which is undesirable for blockchain applications (see Table 3 and Table 4 in Section 7 for comparison). Second, their scheme requires a gap between t_{sig} and t_{real}, where t_{real} is the total weight of signers who released partial signatures and t_{sig} is the weight guaranteed by the aggregated signature. The smaller the gap, the larger the size of their aggregated signature. Third, they do not provide an extension to general access structures.

We provide a comparison of our work with these three approaches in Table 1.

Multiverse Threshold Signatures. A very recent work of [12] proposes the notion of Multiverse Threshold Signature (MTS), where different verifiers may trust different (but potentially overlapping) subsets of signers. Each such subset of signers is referred to as a universe, and the collection of all universes is the multiverse. The key requirement is that each signer's secret state and the signing time should be independent of the number of universes. [12] constructs an MTS scheme in the generic group model where each universe can be set up using a one-round protocol. While our scheme also extends to the multiverse setting, we enjoy several additional advantages over [12]: first, our scheme does not require a per-universe setup (instead, the parties simply publish their public keys once and for all). Second, the partial signature size in our scheme is independent of the parties' weights; in contrast, [12] incurs a linear dependence. Finally, unlike [12], our scheme supports dynamic thresholds (even in the single universe setting) and general access structures.

2. Technical Overview

The starting point of our construction is the BLS multisignature scheme [19]. BLS multisignature enjoys the great benefit that parties can pick their own secret keys; in particular, no DKG is required. During the signing phase, parties simply sign the message using their own secret keys to produce a partial signature σ_i. The aggregated signature is $\sigma = \prod_{i=1}^{n} \sigma_i$, which should verify under the aggregated public key $\mathsf{aPK} = \prod_{i=1}^{n} \mathsf{pk}_i$.[8]

However, BLS multisignature is only an n-out-of-n threshold signature scheme. If one wants to use BLS multisignature as a threshold signature, the following changes are necessary. Suppose a subset of parties $B \subseteq [n]$ signed the message, the aggregated signature shall be $(B, \sigma = \prod_{i \in B} \sigma_i)$. For verification, the verifier needs to compute aPK as $\prod_{i \in B} \mathsf{pk}_i$ and verfies σ under aPK. This approach is highly undesirable: (1) The signature (B, σ) is asymptotically large as it needs to encode the information about B. (2) More crucially, the verification cost is high, e.g., the verifier needs to do t group operations to compute aPK. Additionally, the verifier needs to remember all the public keys, making the gas cost to store the verification key prohibitively high in the blockchain setting (refer to Section 7).

Alternatively, the aggregator can also compute aPK on behalf of the verifier, but then the validity of aPK must still be verified to prevent forgery attacks. Then, the key question we ask is:

Can the aggregator compute a succinct proof to convince the verifier that aPK *is computed correctly?*

If this is feasible, then the verifier can just accept aPK and verify σ under it.

Succinct Non-interactive Argument (SNARK) is by now a standard and concretely efficient technique for proving any NP statements. Can we use SNARKs to prove the validity of aPK? Unfortunately, SNARKs are not well-suited for proving statements involving group operations as they involve many expensive non-native arithmetics. For example, given some succinct encoding of all the public keys $\{\mathsf{pk}_i\}_{i=1}^{n}$ and certain set B, we currently do not have a concretely-efficient way of proving aPK is the product of pk_i for all $i \in B$. However, we can efficiently compute a succinct proof for the *secret statement* $\mathsf{aSK} = \sum_{i \in B} \mathsf{sk}_i$. Therefore, we ask the following question.

Can the aggregator provide a SNARK proof for the secret statement $\mathsf{aSK} = \sum_{i \in B} \mathsf{sk}_i$ *instead?*

The aggregator cannot directly do this as it involves the secret keys of the parties that the aggregator does not possess. Our key idea to enable this is as follows. At setup, each party publishes a "hint", dependent just on its secret key, which facilitates the aggregator to prove this secret statement without harming unforgeability.

In more detail, the aggregator proves well-formedness of aPK in the following manner. Let $\mathsf{SK}(x)$ be the polynomial that encodes all the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$. The verification key will simply be the

8. To prevent the rogue key attack, the multisignature scheme of [19] uses a random oracle to sample a random linear combination to aggregate the public keys. We note that one could also use a Proof-of-Possession [59] approach to prevent such attacks. We implicitly assume a PoP proof is given for the public keys.

polynomial commitment of $\mathsf{SK}(x)$, which can be computed given appropriate hints from each party. Now, the aggregator will use a polynomial commitment scheme to commit to a polynomial $B(x)$, which encodes the set B as the binary vector $(b_1, \ldots, b_n)$ (i.e., $b_i = 1$ if and only if $i \in B$).[9] The prover needs to prove two things: (1) the Hamming weight of $(b_1, \ldots, b_n)$ is $\geqslant T$; (2) the inner product between $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and $(b_1, \ldots, b_n)$ is aSK. The first statement is easier to prove as the aggregator knows the witness $(b_1, \ldots, b_n)$. Therefore, by using a standard SNARK (e.g., PLONK [40]), the aggregator can generate a proof for this.

To prove the second statement, we utilized the generalized sumcheck argument of [57] (refer to Lemma 2)

$$\mathsf{SK}(x) \cdot B(x) = \mathsf{aSK} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x)$$

as follows.[10] The verifier shall accept that aSK is a correctly aggregated secret key as long as the aggregator can provide two quotient polynomials Q_x and Q_Z that satisfy the polynomial identity. Note that, we shall verify this polynomial identity in the exponent using pairing. Therefore, the verifier could verify this statement using only aPK. Again, to compute Q_x and Q_Z honestly, the aggregator will need parties to broadcast appropriate hints.

This summarizes our scheme. The final signature consists of the aggregated public key aPK, aggregated signature σ, commitment to $B(x)$, and two proofs π_1 and π_2. The proof π_1 proves the Hamming weight of $B(x)$ is $\geqslant T$, and the proof π_2 consists of commitments to $Q_x(x)$ and $Q_Z(x)$, which proves the validity of aSK.

Efficiency. By appropriately designing the hints sent by each party, we achieve the following efficiency. Parties publish hints of size $O(n)$. The aggregator, taking all the hints as input, applies a one-time (deterministic) preprocessing algorithm to generate a linear-size (i.e., $O(n)$) aggregation key AK and a constant-size verification key vk. This one-time preprocessing step takes quadratic time. Later on, during the online phase, the aggregator, which takes AK and partial signatures as input, performs only $O(n)$ group operations to generate an aggregated signature. Concretely, our aggregation time is comparable to threshold BLS.

Security. We provide a high-level overview of why unforgeability holds. By invoking the security of standard SNARK, one can be convinced that if proof π_1 verifies, $B(x)$ must encode a vector $(b_1, \ldots, b_n)$ for a subset $B \subseteq [n]$ such that $|B| \geqslant T$. Our objective is to prove: the adversary cannot generate a valid signature if it does not get the partial signature from all parties in B. This naturally reduces to that the adversary cannot make the verifier accept an aSK such that it does not depend on sk_i for some $i \in B$.[11] Now, the verifier only accepts aSK if the aggregator can compute $Q_x(x)$ and $Q_Z(x)$ such that

$$\mathsf{SK}(x) \cdot B(x) = \mathsf{aSK} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x).$$

Note that, there is also the "honest" $Q'_x(x)$ and $Q'_Z(x)$, which testifies the "honest" $\mathsf{aSK}' = \sum_{i \in B} \mathsf{sk}_i$ as

$$\mathsf{SK}(x) \cdot B(x) = \mathsf{aSK}' + Q'_x(x) \cdot x + Q'_Z(x) \cdot Z(x).$$

Taking the difference gives us

$$\mathsf{aSK}' - \mathsf{aSK} = \Delta_x(x) \cdot x + \Delta_Z(x) \cdot Z(x).$$

Now, we shall argue that the adversary can never find $\Delta_x(x)$ and $\Delta_Z(x)$ such that the above polynomial identity holds. This is because all the group elements (including the hints) that the adversary sees (in the exponent) all satisfy $\mathsf{sk}_i \cdot f(x)$ for a sufficiently low degree polynomial $f(x)$. In such cases, the adversary cannot make, for instance,

$$\mathsf{sk}_i = \Delta_x(x) \cdot x + \Delta_Z(x) \cdot Z(x)$$

hold. However, we already established that, to launch a successful attack, it must be the case that $\mathsf{aSK}' - \mathsf{aSK}$ depend on some sk_i where $i \in B$. This essentially completes the security proof.

We stress that we never claim: aSK is the correctly aggregated secret key.[12] We simply argue that the forgery attack can never succeed. To summarize:

Even though the adversary may convince the verifier of an incorrect aPK*, these incorrect* aPK *do not help the adversary launch forgery attacks.*

Extensions. Finally, we highlight that our scheme enjoys several extensions: 1) general policy, 2) post-compromise security and forward security, 3) anonymity, and 4) multiverse threshold signature. We refer the readers to Section 6.2 for discussions on this.

3. Preliminaries

We use κ for the security parameter. We use $\mathsf{negl}(\kappa)$ for a negligible function, which means that for all polynomial $f(\kappa)$, $\mathsf{negl}(\kappa) < 1/f(\kappa)$ for large enough κ. We write $[n]$ for the set $\{1, 2, \ldots, n\}$.

We use $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T)$ for an asymmetric pairing group, where $\mathbb{F}$ is a prime field of order $p = \exp(\Omega(\kappa))$; $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of order p with a non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; $g_1 \in \mathbb{G}_2$ and $g_2 \in \mathbb{G}_2$ are uniformly sampled generators such that $g_T = e(g_1, g_2)$. For a field element $x \in \mathbb{F}$, we shall write $[x]_1$, $[x]_2$, and $[x]_T$ for g_1^x, g_2^x and g_T^x, respectively.

9. Throughout the paper, we abuse notations and use B for both the set $B \subseteq [n]$ and the polynomial $B(x)$.

10. More precisely, the lemma states that aSK is a constant factor of the sum $\sum_i b_i \cdot \mathsf{sk}_i$, which is also an acceptable aggregated key.

11. This reduction is essentially due to the security of multisignature.

12. In fact, we utilize this to allow the aggregator to add a random group element to aSK to achieve anonymity (See Section 6.2).

Algebraic Group Model. We work in the Algebraic Group Model (AGM), introduced by [39]. 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 [39], we also assume the following q-DLOG problem is hard for algebraic adversaries.

Definition 1 (q-DLOG Assumption)**.** *For any positive integer q and algebraic adversary $\mathcal{A}$, it holds that*

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

Assuming the q-DLOG assumption, the following lemma [39] simplifies the security analysis in AGM.

Lemma 1. *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 $1 - \mathsf{negl}(\kappa)$ 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.

Our construction of the threshold signature is based on the BLS signature [21] defined as follows.

Definition 2 (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* $(\mathsf{pk} = [\mathsf{sk}]_1, \mathsf{sk})$.
- $\mathsf{BLS.Sign}(\mathsf{msg})$*: It signs as* $\sigma = \mathcal{H}(\mathsf{msg})^{\mathsf{sk}}$.
- $\mathsf{BLS.Verify}(\mathsf{pk}, \mathsf{msg}, \sigma)$*: It verifies the validity of the signature by* $e(\mathsf{pk}, \mathcal{H}(\mathsf{msg})) \stackrel{?}{=} e([1]_1, \sigma)$.

Notations for Polynomials. Throughout the paper, we use the following notations for polynomials over finite fields. Let $\mathbb{H} \subset \mathbb{F}$ be a multiplicative subgroup of a finite field $\mathbb{F}$. Let ω be the generater of $\mathbb{H} = \{\omega, \omega^2, \ldots, \omega^{|\mathbb{H}|} = 1\}$. Let $L_1(x), L_2(x), \ldots, L_{|\mathbb{H}|}(x)$ be the Lagrange basis polynomial. That is, L_i is the unique degree $|\mathbb{H}| - 1$ polynomial defined by: $L_i(\omega^j)$ is 1 when $i = j$ and 0 when $i \neq j$. Let $Z(x) = \prod_{i=1}^{|\mathbb{H}|}(x - \omega^i)$ be the vanishing polynomial on $\mathbb{H}$. Since $\mathbb{H}$ is a multiplicative subgroup, $Z(x) = x^{|\mathbb{H}|} - 1$ and $L_i(x) = \frac{\omega^i}{|\mathbb{H}|} \cdot \frac{x^{|\mathbb{H}|}-1}{x-\omega^i}$. Note that $L_i(0) = |\mathbb{H}|^{-1}$. In our construction, we use $|\mathbb{H}| = n+1$, where n is the number of parties.

Polynomial Commitment. We use the KZG polynomial commitment [47]. 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 by $\prod_{i=0}^n [\tau^i]_1^{a_i}$. To open the polynomial at x^*, one computes the quotient polynomial $Q(x) = \frac{f(x)-x^*}{x-x^*}$. The opening proof is $\pi = [Q(\tau)]_1$. To verify the proof, one checks $e(\sigma, [1]_2) \stackrel{?}{=} e(\pi, [\tau - x^*]_2)$.

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 $e(\sigma_1^r \sigma_2^{r^2} \cdots \sigma_\ell^{r^\ell}, [1]_2) \stackrel{?}{=} e(\pi_1^r \pi_2^{r^2} \cdots \pi_\ell^{r^\ell}, [\tau - x^*]_2)$. 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 $\pi = \pi_1^r \pi_2^{r^2} \cdots \pi_\ell^{r^\ell}$ as the batched proof.

Generalized Sumcheck. Our construction relies on the following lemma known as generalized sumcheck. We refer the readers to Theorem 1 of [57] for proof.

Lemma 2 (Generalized Sumcheck [57])**.** *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*

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

where both Q_x and Q_Z are polynomials with degree $\leqslant |\mathbb{H}| - 2$ defined as

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

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

4. Definition of Silent Threshold Signature

This section formally defines the primitive silent threshold signature (STS). In an STS scheme, parties will publish some "hints" together with their public key in a *silent* manner. Given all the hints, a public algorithm will verify the validity of the hints. Furthermore, a *succinct* verification key will be deterministically computed from the hints. Formally, we have the following.

Definition 3 (STS)**.** *A Silent Threshold Signature consists of the following algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{HintGen}, \mathsf{Preprocess}, \mathsf{Sign}, \mathsf{SignAggr}, \mathsf{Verify})$*:*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$*: 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* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{hint} \leftarrow \mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}, n)$*: On input the* crs*, the secret key* sk*, and the number of parties n, the* $\mathsf{HintGen}$ *algorithm outputs a hint* hint.

- $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]})$*: On input the* crs*, all pairs* $\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]}$*, the* Preprocess *algorithm computes an aggregation key* AK *and a (succinct) verification key* vk.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: *On input some secret key* sk*, and some message* msg*, the* Sign *algorithm outputs a partial signature* σ.
- $1/0 \leftarrow \mathsf{PartialVerify}(\mathsf{msg}, \sigma, \mathsf{pk})$: *On input a message* msg*, a partial signature* σ*, and a public key* pk*, it returns* 1 *if and only if the partial signature is verified correctly.*
- $\sigma \leftarrow \mathsf{SignAggr}(\mathsf{crs}, \mathsf{AK}, \{\sigma_i\}_{i\in B})$: *On input the* crs*, an aggregation key* AK*, and a set of signatures* $\{\sigma_i\}_{i\in B}$*, the* SignAggr *algorithm outputs a (succinct) signature* σ.
- $b \leftarrow \mathsf{Verify}(\mathsf{msg}, \sigma, T, \mathsf{vk})$: *On input a message* msg*, a signature* σ*, a threshold* T*, and the verification key* vk*, it verifies the signature.*

Moreover, STS 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.*

Remark 1 (Extended Public Key)**.** We note that HintGen does not take other parties' pk_i's as input. It solely depends on the CRS and parties can publish $(\mathsf{pk}_i, \mathsf{hint}_i)$ in *one shot*. In other words, $(\mathsf{pk}_i, \mathsf{hint}_i)$ can be viewed as the (extended) public key of party i.

Correctness and unforgeability are defined below. We highlight that correctness and unforgeability are defined in the *malicious* setting. That is, we consider 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. In particular, this subsumes correctness and security in the semi-honest setting.

Definition 4 (Correctness)**.** *The STS scheme* Σ *satisfies correctness if, for any adversary* $\mathcal{A}$*, the output of the* Correctness − Game *defined in Figure 1 is* 1 *with probability* $\geqslant 1 - \mathsf{negl}(\kappa)$.

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* A *of corrupt parties such that* $A \subseteq [n]$.
3) *For all* $i \in [n] \setminus A$*, the public key and hint are sampled honestly* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(1^\kappa)$ *and* $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}_i, n)$.
4) *For all* $i \in A$*, the adversary returns a public key* pk_i *and a hint* hint_i *to the challenger.*
5) *The public pre-processing is invoked by the challenger as* $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]})$ *and the output are given to* $\mathcal{A}$.
6) *The adversary picks a message* msg *and prepare the partial signatures* $\{\sigma_i\}_{i\in B_1}$ *for some subset of malicious parties* $B_1 \subseteq A$*. Let* $B_1' \subseteq B_1$ *be the subset of maliciously generated signatures that verifies under* PartialVerify.
7) *The adversary may also request a subset of honest parties* $B_2 \subseteq [n] \setminus A$ *for partial signatures, which are returned by computing* $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{msg})$ *on a given message for all* $i \in B_2$*. This process is run as many times as desired by the adversary.*
8) *The challenger computes the aggregated signature as* $\sigma \leftarrow \mathsf{SignAggr}(\mathsf{crs}, \mathsf{AK}, \{\sigma_i\}_{i\in B})$*, where* $B = B_1' \cup B_2$ *is the set of all partial signatures that verifies.*
9) *The output of this game is 1 if,* for all $T \leqslant |B|$*, we have* $\mathsf{Verify}(\mathsf{msg}, \sigma, T, \mathsf{vk}) = 1$.[13]

Figure 1: Correctness − Game

Definition 5 (Unforgeability)**.** *The STS scheme* Σ *satisfies unforgeability if, for any adversary* $\mathcal{A}$*, the output of the game in Figure 2 is* 1 *with probability* $\leqslant \mathsf{negl}(\kappa)$.

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 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* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(1^\kappa)$ *and* $\mathsf{hint}_i = \mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}_i, n)$.
4) *For all* $i \in A$*, the adversary picks a public key* pk_i *and the corresponding hint* hint_i.
5) *The challenger preprocess* $\{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]}$ *as* $(\mathsf{AK}, \mathsf{vk}) \leftarrow \mathsf{Preprocess}(\mathsf{crs}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]})$ *which are given to* $\mathcal{A}$.
6) *The adversary may make a partial signature query with a message* msg *and an honest party* $i \in [n] \setminus A$ *– the query is returned by computing* $\sigma_i = \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$*. This is run as many times as desired by* $\mathcal{A}$.
7) *Finally, the adversary shall output a challenge message* msg^* *and an aggregated signature* σ^**. Let* B^* *be the subset of honest parties queried by the adversary to sign* msg^*.
8) *The adversary wins the forgery game if there exists a threshold* $T > |B^* \cup A|$ *such that* $\mathsf{Verify}(\mathsf{msg}^*, \sigma^*, T, \mathsf{vk}) = 1$*, in which case, the output of the game is 1.*

Figure 2: Forgery − Game

Silent Weighted Threshold Signature. Weighted access structure is similar to threshold structure except that there is a predefined weight vector $(w_1, w_2, \ldots, w_n)$ associated with all the parties. For every message msg 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 is essentially analogous to the definition for the threshold case except for the last step of the correctness and forgery game.

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

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

5. Construction of hinTS

In this section, we present a construction for Definition 3 based on asymmetric pairings. We start by describing the construction of the (unweighted) threshold access structure. Next, we describe the minor modifications needed for the weighted setting.

Construction. Each algorithm is specified as follows.

- $\mathsf{Setup}(1^\kappa)$: it samples a random $\tau \leftarrow \mathbb{F}$, a hash function $\mathcal{H} : \{0,1\}^* \to G_2$ and set
$$\mathsf{crs} = (\mathcal{H}, [\tau]_1, [\tau^2]_1, \ldots, [\tau^M]_1, [\tau]_2, \ldots, [\tau^M]_2).$$
Here, M is an upper bound on the maximum universe size. In particular, it holds that $M \geqslant n+1$.
- $\mathsf{KGen}(1^\kappa)$: This is the same as BLS.Gen. That is, it samples $\mathsf{sk} \leftarrow \mathbb{F}$ and output $(\mathsf{pk} = [\mathsf{sk}]_1, \mathsf{sk})$.
- $\mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}_i, n)$: Party P_i computes the following things as hint_i.
$$[\mathsf{sk}_i \cdot L_i(\tau)]_1,$$
$$\left[\mathsf{sk}_i \cdot \frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}\right]_1, \left\{\left[\mathsf{sk}_i \cdot \frac{L_i(\tau)\cdot L_j(\tau)}{Z(\tau)}\right]_1\right\}_{j\neq i},$$
$$\left[\mathsf{sk}_i \cdot \frac{L_i(\tau) - L_i(0)}{\tau}\right]_1, \left[\mathsf{sk}_i \cdot (L_i(\tau) - L_i(0))\right]_1.$$
- $\mathsf{Preprocess}(\mathsf{crs}, \{\mathsf{hint}_i, \mathsf{pk}_i\}_{i\in[n]})$: It first verifies the validity of the hints through pairings. For instance, the $[\mathsf{sk}_i \cdot L_i(\tau)]_1$ term should satisfy
$$e([\mathsf{sk}_i \cdot L_i(\tau)]_1, [1]_2) = e([\mathsf{sk}_i]_1, [L_i(\tau)]_2).$$
The rest of the terms in hint_i could be verified similarly.[14] Let $\mathcal{E}$ be the set of parties whose hints do not verify. It proceeds to compute AK as follows.
  - It set $\mathsf{pk}_i = [0]_1$ for all $i \in \mathcal{E}$. (Note that this effectively sets $\mathsf{sk}_i = 0$.)
  - It set $w_i = 0$ if $i \in \mathcal{E}$ and $w_i = 1$ if $i \in [n] \setminus \mathcal{E}$.

  The aggregation key AK consists of the following.
$$\mathcal{E},\ \{w_i, \mathsf{pk}_i\}_i,$$
$$\left\{\left[\mathsf{sk}_i \cdot \frac{L_i^2(\tau) - L_i(\tau)}{Z(\tau)}\right]_1\right\}_i, \left\{\left[\sum_{j\neq i} \mathsf{sk}_j \cdot \frac{L_i(\tau)\cdot L_j(\tau)}{Z(\tau)}\right]_1\right\}_i,$$
$$\left\{\left[\mathsf{sk}_i \cdot \frac{L_i(\tau) - L_i(0)}{\tau}\right]_1\right\}_i, \left\{\left[\mathsf{sk}_i \cdot (L_i(\tau) - L_i(0))\right]_1\right\}_i.$$

14. We note that all verifications on party P_i can be done by first taking a random linear combination over hint_i and one pairing check.

Note that the size of the AK is $O(n)$. For the verification key vk, compute the following.
$$\mathsf{SK}(\tau) = \sum_{i\in[n]} \mathsf{sk}_i \cdot L_i(\tau), \quad W(\tau) = \sum_{i\in[n]} w_i \cdot L_i(\tau),$$
$$Z(\tau) = \prod_{i\in[n+1]} (\tau - \omega^i).$$
The verification key is
$$\mathsf{vk} = \Big([\mathsf{SK}(\tau)]_1, [W(\tau)]_1, [Z(\tau)]_2\Big).$$

- $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: This is the same as BLS.Sign. That is, $\sigma = \mathcal{H}(\mathsf{msg})^{\mathsf{sk}}$.
- $\mathsf{PartialVerify}(\mathsf{msg}, \sigma, \mathsf{pk})$: Same as BLS.Verify, it outputs the result of $e(\mathsf{pk}, \mathcal{H}(\mathsf{msg})) \stackrel{?}{=} e(g_1, \sigma)$.
- $\mathsf{SignAggr}(\mathsf{crs}, \mathsf{AK}, \{\sigma_i\}_{i\in B})$: It first set $B = B \setminus \mathcal{E}$. That is, no party whose hint is erroneous will be considered. Next, it computes the aggregated public key and signature as
$$\mathsf{aPK} = \Big(\prod_{i\in B} \mathsf{pk}_i\Big)^{|\mathbb{H}|^{-1}}, \qquad \sigma' = \Big(\prod_{i\in B} \sigma_i\Big)^{|\mathbb{H}|^{-1}}.$$
The final signature is
$$\sigma = (\mathsf{aPK}, \sigma', \pi),$$
where π is a succinct proof proving the correct aggregation of aPK. For all i, let $b_i = 1$ iff $i \in B$. The proof π consists of the following.
  1) The claimed threshold $w = |S|$.
  2) A commitment to the vector of signers
$$[B(\tau)]_2 = \left[\sum_{i\in B} L_i(\tau)\right]_2.$$
  3) The commitments to the quotient polynomial $Q_x(x)$ and $Q_Z(x)$ satisfying
$$\mathsf{SK}(x)\cdot B(x) - \mathsf{aSK} = Q_Z(x)\cdot Z(x) + Q_x(x)\cdot x, \quad (1)$$
where $\mathsf{aSK} = (\sum_{i\in B} \mathsf{sk}_i)/|\mathbb{H}|$. Due to Lemma 2, they could be computed as
$$Q_Z(\tau) = \sum_{i\in[n]} b_i \cdot \left(\mathsf{sk}_i \cdot \frac{L_i(\tau)\cdot L_i(\tau) - L_i(\tau)}{Z(\tau)}\right) + \sum_{i\in[n]} b_i \cdot \left(\sum_{j\neq i} \mathsf{sk}_j \cdot \frac{L_i(\tau)\cdot L_j(\tau)}{Z(\tau)}\right)$$
and
$$Q_x(\tau) = \sum_{i\in[n]} b_i \cdot \left(\mathsf{sk}_i \cdot \frac{L_i(\tau) - L_i(0)}{\tau}\right).$$
Note that the quantities in the bracket are precomputed as AK. Therefore, computing $[Q_Z(\tau)]_1$ and $[Q_x(\tau)]_1$ takes only $O(n)$ group operations.
  4) Compute
$$[Q_x(\tau)\cdot\tau]_1 = \prod_{i\in B} [\mathsf{sk}_i \cdot (L_i(\tau) - L_i(0))]_1,$$

which will be used for degree check on $Q_x(\tau)$.

5) Commit to the following partial sum polynomial

$$\mathsf{ParSum}(x) = \sum_{i\in[n+1]} \left(\sum_{j=1}^{i-1} b_j \cdot w_j\right) \cdot L_i(x).$$

That is, one computes $[\mathsf{ParSum}(\tau)]_1$.

6) Compute quotient polynomials $Q_1(x)$ and $Q_2(x)$ for the following polynomial identities

$$\mathsf{ParSum}(x\cdot\omega) - \mathsf{ParSum}(x) - (W(x) - w\cdot L_{n+1}(x))\cdot B(x) = Z(x)\cdot Q_1(x) \quad (2)$$

and

$$B(x)\cdot(1-B(x)) = Z(x)\cdot Q_2(x). \quad (3)$$

Intuitively, the first identity checks the well-formedness of ParSum.[15] The second identity checks that $B(x)$ is 0 or 1 at $x \in \mathbb{H}$. It commits to them by computing $[Q_1(\tau)]_1, [Q_2(\tau)]_1$.

7) By Fiat-Shamir heuristic, a random challenge r is generated using a random oracle $\mathcal{O} : \{0,1\}^* \to \mathbb{F}$.

$$r = \mathcal{O}\Big(w, [B(\tau)]_2, [Q_Z(\tau)]_1, [Q_x(\tau)]_1, [Q_x(\tau)\cdot\tau]_1, [\mathsf{ParSum}(\tau)]_1, [Q_1(\tau)]_1, [Q_2(\tau)]_1\Big).$$

It computes the following opening and batch opening proof of the committed polynomials.

- Open and prove $\mathsf{ParSum}(\omega) = 0$.
- Open and prove $B(\omega^{n+1}) = 1$. (Refer to Footnote 15.)
- Open and prove the following at $x = r$.

$$\left\{\begin{matrix}\mathsf{ParSum}(x), W(x), B(x),\\ Q_1(x), Q_2(x)\end{matrix}\right\}.$$

- Open and prove $\mathsf{ParSum}(x)$ at $x = r\cdot\omega$.

We note that Step (5)-(7) is exactly a Plonk-style SNARK for proving $B(x)$ has sufficient weight.

- $\mathsf{Verify}(\mathsf{msg}, \sigma, T, \mathsf{vk})$: Parse σ as above. The verifier checks the following things.
  - **Verify the proof π.**
    1) Verify all opening proof of the polynomials.
    2) Check that Equation 2 and 3 hold at the evaluation point r.[16]
    3) Check that Equation 1 holds using pairing

$$e([\mathsf{SK}(\tau)]_1, [B(\tau)]_2)\cdot e(\mathsf{aPK}, [1]_1)^{-1} = e([Q_Z(\tau)_1, [Z(\tau)]_2)\cdot e([Q_x(\tau)]_1, [\tau]_2).$$

15. Note that $\mathsf{ParSum}(\omega\cdot x) - \mathsf{ParSum}(x)$ is $b_i\cdot w_i$ at everywhere except for $x = \omega^{n+1}$, in which case, $\mathsf{ParSum}(\omega\cdot x) - \mathsf{ParSum}(x) = 0 - w$. Therefore, we add $-w\cdot L_{n+1}(x)$ to $W(x)$ and further require that $B(\omega^{n+1}) = 1$.

16. Here, we assume that the verifier will evaluate $Z(x) = x^{|\mathbb{H}|} - 1$ at $x = r$, which takes $\log n$ field operations. The concrete efficiency of this is insignificant compared to the other constant number of pairings and group operations that the verifier performs. However, if one wishes to strictly enforce that the verifier is constant-time (i.e., also performs a constant number of field operations), one can also ask the aggregator to compute $Z(r)$ along with a KZG opening proof.

    4) Run the degree check on $Q_x(\tau)$ as

$$e([Q_x(\tau)]_1, [\tau]_2) \stackrel{?}{=} e([Q_x(\tau)\cdot\tau]_1, [1]_2).$$

  - **Verify the aggregated signature σ'.** This is the same as $\mathsf{BLS.Verify}$. Check if $e(\mathsf{aPK}, \mathcal{H}(\mathsf{msg})) \stackrel{?}{=} e([1]_1, \sigma')$
  - **Check the claimed threshold.** Finally, it checks if $T \leqslant w$, i.e., the (proven) total number w of parties signed is no less than the claimed threshold T.

Construction for Weighted STS. We highlight that the construction of the weighted silent threshold signature is almost identical to our construction above. The only difference is that, instead of using $w = (1, 1, \ldots, 1)$ as the weight vector, we use the predefined weights $w = (w_1, w_2, \ldots, w_n)$.[17] The other changes follow naturally. For instance, the aggregated signature claims $w = \sum_{i\in B} w_i$ as the total weight instead of $w = |B|$.

Now, we directly prove the correctness and unforgeability of our construction for the weighted STS according to definitions in Section 4.

Proof of Correctness. Observe that any hints and partial signatures sent by the malicious parties must be correct if they pass verification. These are summarized as the following two claims. These claims are easy to see as the honest generated hint_i and σ_i are the *unique group elements* that may pass the pairing check.

Claim 1. *For all corrupted parties $i \in A$, let $(\mathsf{hint}_i, \mathsf{pk}_i)$ be the public key and corresponding hint provided by the adversary $\mathcal{A}$. If $\mathsf{hint}_i \neq \mathsf{HintGen}(\mathsf{crs}, \mathsf{sk}_i, n)$, it must hold that $i \in \mathcal{E}$.*

Claim 2. *For a message* msg *and the partial signatures that verify (i.e., $\mathsf{PartialVerify}(\mathsf{msg}, \sigma_i, \mathsf{pk}_i) = 1$), σ_i must be the correctly generated partial signature.*

Note that the hints and the partial signature are the only messages that the adversary sent that may be incorrect. As the claims state that these must be correct, the correctness in the malicious setting reduces to the correctness in the semi-honest setting, which holds trivially.

Proof of Unforgeability. The unforgeability proof is deferred to Appendix A. The proof essentially works by utilizing Lemma 1 for AGM and carefully arguing that the adversary cannot compute group elements (i.e., the forgery) that satisfies all the polynomial identities, checked by the pairing equations of the verification procedure.

Unforgeability for Adaptive Adversary. Our definition considers a static adversary that selects the corrupted parties at the beginning of the game. One may consider an adaptive adversary that adaptively selects the parties to corrupt as the game proceeds. We note

17. Malicious parties who send erroneous hints are still set to have $w_i = 0$ during preprocessing.

that *our scheme is adaptively secure in AGM*. One may prove this by standard tricks. In particular, for adaptive security, the simulator will honestly sample parties' secret keys, except for one party whose public key is the embedded challenge. The simulator can answer all the corruption queries since, with a non-negligible probability, the adversary will never corrupt the honest party with the embedded challenge. Conditioned on the adversary never corrupts the embedded party, the rest of the proof is similar to the proofs in the static setting. We omit the formal definitions and proofs due to space constraints.

6. Extensions and Optimizations

6.1. Optimizations

Signature Size. For ease of presentation, our construction omits a few optimizations that one could use to reduce the signature size. This optimization is similar to the optimization in Plonk [40].

1) Instead of verifying $\mathsf{ParSum}(\omega) = 0$, one could verify $L_1(x) \cdot \mathsf{ParSum}(x) = Q_3(x) \cdot Z(x)$ at random location $x = r$.
2) Similarly, instead of verifying $B(\omega^{n+1}) = 1$, one verifies $L_{n+1}(x) \cdot (1 - B(x)) = Q_4(x) \cdot Z(x)$ at random location $x = r$.
3) We can now batch $Q_1(x), Q_2(x), Q_3(x), Q_4(x)$ into just one quotient polynomials. In particular, a random challenge v is sampled by the random oracle, the prover only sends
$$Q(x) = Q_1(x) + v \cdot Q_2(x) + v^2 \cdot Q_3(x) + v^3 \cdot Q_4(x).$$
4) As we mentioned, all the KZG opening proofs at the same location $x = r$ can also be batched into a single group element open_r.

The final signature consists of the following

$$\left\{ \begin{array}{c} \mathsf{aPK}, \sigma', w, \\ [B(\tau)]_2, [Q_x(\tau)]_1, [Q_Z(\tau)]_1, [\mathsf{ParSum}(\tau)]_1, [Q(\tau)]_1, \\ B(r), \mathsf{ParSum}(r), \mathsf{ParSum}(r \cdot \omega), W(r), Q(r), \\ \mathsf{open}_r,\ \mathsf{open}_{r \cdot \omega}. \end{array} \right\}.$$

Except for the claim threshold $w \in \mathbb{N}$, the rest of the signature consists of 9 group and 5 field elements.

HintGen without size n and index i. In our basic construction, $\mathsf{HintGen}$ takes as input n and implicitly the index i. In practice, parties may not know the size of the universe or its index. This can be fixed as follows.

Parties could easily generate their hints without n by assuming the $n = M$, where M is the maximum universe size. However, this is not an efficient solution as the efficiency of the aggregation now depends on M instead of n. What parties could do is to publish a set of hints for every $n \in \{2, 2^2, \ldots, 2^{\log M}\}$. Later, whenever one wants to set up a universe with size $2^t \leqslant n < 2^{t+1}$, the aggregator can use the corresponding hints from every party. The total size of the published hints is $O(M)$.

The above approach has a subtle security issue as our security relies on the adversary not being able to compute $\mathsf{sk}_i \cdot f(\tau)$ for a high degree polynomial $f(\tau)$. But if the same τ is used for universes with various sizes, the adversary might be able to break this guarantee. One way to fix this is to use a different τ for universes with different sizes. Note that this only increases the CRS size by a factor of 2.

We present two ways for parties to generate hints without knowing i. First, parties could simply publish $\mathsf{sk}_i \cdot \tau, \mathsf{sk}_i \cdot \tau^2, \ldots, \mathsf{sk}_i \cdot \tau^n$. This would allow the aggregator to compute any degree n polynomial for sk_i; in particular, it can compute the hints for this party with respect to any index j. The aggregator, however, will need to do more work in the preprocessing phase.

The second solution is to use hashing. In particular, parties could use some hash function $\mathsf{hash}(\mathsf{pk})$ to find a set $S = \{i_1, i_2, \ldots\}$ of candidate indices. Then, parties generate hints with respect to every index in S. Later, to set up a universe, parties are assigned an index inside their respective candidate set such that there is no collision among all parties. The efficiency depends on how large the set S needs to be to avoid the collision. Cuckoo hashing [54], for instance, only requires a constant-size candidate set to avoid the collision.

6.2. Extensions

General Policy Aggregatable Signature. Our basic scheme already gives a construction for the weighted access structure. One could further consider a general access structure $\Lambda \subseteq 2^{[n]}$, which defines all authorized subsets. The objective here is the following: if an authorized subset of parties signed msg, the aggregator should be able to aggregate these partial signatures into σ. The validity of σ will convince the verifier that some authorized subset of parties all signed the message.

Our basic scheme can be easily adapted to support a general policy as follows. The aggregator still computes the polynomial $B(x)$, which encodes the subset of signed parties. It still sends aPK, $Q_x(x)$, and $Q_Z(x)$ to convince the verifier that aPK is a correct aggregation of the public keys from parties encoded in $B(x)$. This is identical to the basic scheme.

Now, the different part is: the aggregator will use a Plonk-style Snark to generically prove that the witness encoded in $B(x)$ satisfies the access structure Λ. For instance, if Λ can be described by a circuit $C : \{0,1\}^n \to \{0,1\}$, one can prove that $C\big(B(\omega), \ldots, B(\omega^n)\big) = 1$ using Plonk. In order to facilitate verification of the proof, the preprocessing algorithm needs to compute a Snark verification key for the circuit C. The efficiency for generating this proof depends linearly on the size of the circuit $|C|$.

Forward Security. Forward security [14] is another highly desirable feature, which aims to prevent adversaries from forging signatures in the past. To achieve this security notion, parties would regularly update their secret key (while retaining the same public key). This notion is inspired, in particular, by the so-called long-range attack [27] in the proof-of-stake blockchain applications.

Recently, Drijvers et al. [35] proposed a forward-secure multisignature scheme based on BLS multisignature. Their scheme maintains the original BLS public key structure and only modifies the secret keys. In particular, the aggregated signature is still verified under the aggregated public key aPK, which is a linear combination of each party's BLS public keys. Our construction enables the verifier to succinctly verify whether an aggregated public key aPK is a correct aggregation of sufficient many public keys. Thus, our construction can be directly combined with the method proposed by [35] to create a forward-secure threshold signature scheme.

Proactive Security. Another desirable security requirement for threshold signature is proactive security [53] (a.k.a., post-compromise security). It deals with scenarios where a party's secret key is leaked to an adversary, and the party wishes to send an update to refresh the secret keys to restore security. Crucially, the update should not alter the public key to ensure continuity.

In our scheme, this can be achieved as follows. If a party i wants to update his signing key, he simply samples a new polynomial $\Delta\mathsf{SK}(x)$ and informs the j^{th} party to shift his secret key by $\Delta\mathsf{SK}(\omega^j)$. He also shifts his own secret key by $\Delta\mathsf{SK}(\omega^i)$. He will also send the updated aggregation key to the aggregator. This effectively changes the encoding of the secret key polynomial to $\mathsf{SK}(x) + \Delta\mathsf{SK}(x)$. However, we do not need to update the verification key from $[\mathsf{SK}(\tau)]_1$ to $[\mathsf{SK}(\tau) + \Delta\mathsf{SK}(\tau)]_1$. Instead, the signature itself could contain $[\Delta\mathsf{SK}(\tau)]_1$ to facilitate the verification process. As there are more and more updates $[\Delta\mathsf{SK}_1(\tau)]_1, [\Delta\mathsf{SK}_2(\tau)]_1, \ldots$, the aggregator may simply add them up and append it to the signature as $[\Delta\mathsf{SK}_1(\tau) + \Delta\mathsf{SK}_2(\tau) + \cdots]_1$. Thus, the signature size will not grow in terms of the number of updates.

There is still a security issue. An adversary may intentionally choose a polynomial $\Delta\mathsf{SK}(x)$ that he does not know the evaluation to launch attacks. For instance, the adversary may choose $\Delta\mathsf{SK}(x) = -\mathsf{SK}(x)$. In order to avoid this, the party who is updating the secret key polynomial must prove that he knows polynomial $\Delta\mathsf{SK}(x)$ entirely. This can be easily fixed as follows. In the CRS, we embed a hidden challenge $[a]_2$ and publish $[a\cdot\tau]_1, [a\cdot\tau^2]_1, \ldots$. Whenever the party publish $[\Delta\mathsf{SK}(\tau)]_1$, we also require it to publish $[a \cdot \Delta\mathsf{SK}(\tau)]_1$, which should pass the pairing check using $[a]_2$.

Anonymizing the Signers. Threshold signature is anonymous as the signature hides the set of signers. We stress that one could also make our aggregated signature anonymous. First, the zero-knowledge property can be added to Plonk-style proof easily. All the polynomials involved in the Plonk proof are padded with a random multiple of the vanishing polynomial, e.g., $f(x) \cdot Z(x)$. Particular to our scheme, we need to mask $B(x)$ and $\mathsf{ParSum}(x)$. The degree of $f(x)$ used in the mask depends on how many points one needs to open the polynomial at. For instance, we are only going to open $B(x)$ at one location; hence, we only need to mask $B(x)$ by $c \cdot Z(x)$ for a random constant c. On the other hand, we open $\mathsf{ParSum}(x)$ at two locations; hence, we need to mask it by $(c_1 \cdot x + c_0) \cdot Z(x)$ for random c_1, c_0.

This shows how we can make $B(x)$ and the proof of it zero-knowledge. However, the aggregated public key aPK still leaks information. Therefore, we need to sample a random mask c and pad aPK with g^c. To fix the generalized sumcheck polynomial identity, we need to calculate the quotient polynomials $\Delta Q'_x(x)$ and $\Delta Q'_Z(x)$ such that

$$c = \Delta Q'_x(x) \cdot x + \Delta Q'_Z(x) \cdot Z(x).$$

The aggregator can compute this as $Z(x)$ and x are coprime.[18] Now, the aggregator can convince the verifier that $\mathsf{aPK} \cdot g^c$ is the aggregated public key. Clearly, $\mathsf{aPK} \cdot g^c$ hides the identities of the signer.

Multiverse Threshold Signature (MTS). Recently, Baird et al. [12] proposed the notion of multiverse threshold signature. An MTS scheme enables a group of parties to create multiple universes, each allowing a specific subset of parties to perform threshold signature based on a universe-specific threshold. As a crucial efficiency requirement, a party's secret state should remain succinct; in particular, it should not grow linearly in the number of universes it belongs to. Thus, a naïve solution of running a separate DKG for each universe is not a viable solution. Our construction directly gives a solution for MTS. Furthermore, it significantly improves the construction of Baird et al. [12]. In our solution, parties could publish the hints once and for all. For any subset of parties who wants to form a universe with any threshold, the aggregation key and the verification key can be computed in a transparent manner from the published hints.[19] In comparison, the solution of Baird et al. [12] still requires involving parties to engage in a one-round setup protocol for each universe.

7. Implementation and Evaluation

We implement our hinTS construction in Rust and release it open-source at https://github.com/hintsrepo/hints. We use the BLS12-381 pairing-based curve [20], and the hashing to elliptic curve method defined in [38] – we rely on the arkworks libraries for their

18. Note that our degree-check only ensure that the part of $Q_x(x)$ related to sk_i has degree $\leqslant |\mathbb{H}| - 2$. The part of $Q_x(x)$ that is independent of the secret keys could potentially have a higher degree.

19. In particular, our earlier discussion on generating hints without knowing n and i is highly relevant to this setting.

implementation. For efficiency, we implement multi-exponentiation of group elements (within the aggregator) using Pippenger's method [55], [22], which, for n group elements, requires $O(n/\log(n))$ running time as opposed to $O(n)$.

For a fair comparison of all schemes, we only implement the single-threaded version of the algorithms, though there are obvious opportunities for parallelism. All experiments are run on a Macbook Pro with M1 Pro chip and 32 GB RAM. We also report EVM gas costs[20] for publishing and verifying signatures on-chain.

We now compare our hinTS construction to some alternative threshold signature schemes. These include: 1) generic SNARK approach; 2) compact certificates in Micali et. al. [50]; 3) threshold BLS signatures [18]; and, 4) multisignature based on BLS [49], [19]. These schemes are described below.

Aggregation using SNARKs Assuming a public key infrastructure (PKI), an aggregator, on receiving signatures from a set of parties, can produce a SNARK proof that convinces a verifier that the prover knows (as witness) a set of valid signatures, each verifiable under a distinct public key in the PKI – the prover also proves that the number or the aggregate weight of the signers exceeds a threshold. To set up the experiment, we use the gnark library [3] to create a circuit composing multiple instances of the signature verification circuit.[21] For what seems like a fair comparison, we choose the most SNARK-friendly signature scheme available in the gnark library, which is EdDSA signatures – with the gnark frontend, a single EdDSA verification produces roughly 6.2K constraints in the Groth16 system [44] and 13.1K constraints in the PLONK system [40]. Furthermore, we will assume that the verifier stores the table that maps nodes to their public keys; alternatively, the proof can be constructed with respect to a commitment to this table, but that only adds to the prover (aggregator) running time that we report. For hashing, which is used in the signature scheme as a random oracle and must take place inside the SNARK circuit, we use the MiMc [11] hash function. As we show later, the aggregator's running time is prohibitively expensive for hundreds of nodes, even if its efficiency is asymptotically identical to hinTS.

Compact Certificates Micali et. al. [50] introduce compact certificates, based on non-interactive proofs of knowledge in the random oracle model. The certificate proves that signers have a sufficient total weight, while only including a logarithmic number of individual signatures. As we show later, the certificate size is an order of magnitude larger than ours, incurring a heavy gas cost to not only submit the certificate on chain but also for the smart contract to verify a logarithmic number of partial signatures and Merkle paths (each of log size).

Threshold BLS The (standard) threshold BLS [18] system uses a BLS scheme that is set up by a distributed key generation (DKG) protocol, such as Gennaro et al. [42] or with the recent improvements in [63]. The only known technique for dealing with weights in this scheme is virtualization, where each party is given as many shares as its weight. Not surprisingly, as we show later, signing is linear in the weight, and aggregation is linear in the total weight of all parties (specifically, the threshold weight) – this becomes prohibitively expensive for modest precision (e.g., 32-bit) weights.

BLS Multisignature We also study BLS multisignatures [49], [19], where rogue key attacks are addressed via proofs-of-possession in a setup phase (i.e., each party will include the proof when publishing the public key). This scheme has the following properties: 1) the aggregated signature contains the identity of each signer (in a bitmap of size n bits); and 2) the aggregated public key is computed by the verifier (e.g., smart contract) by aggregating the public keys of all signers. As we show later, this scheme has expensive (on-chain) costs, both in setup cost (storing linear public keys) and the verification cost (adding linear group elements).

7.1. Aggregation Time

TABLE 2: Aggregation Time (secs)

Parties	SNARK Groth16/PLONK	Thr. BLS $W = \{1, 2^{10}\}$	Mul. BLS	**hinTS**
128	11.2 / 6.9	0.011, 2.87	0.00029	0.06
256	24.5 / 14.6	0.019, 5.72	0.00057	0.13
512	71.14 / 39.2	0.057, 11.4	0.0015	0.23
1024	186.3 / 126.3	0.126, 24.16	0.0029	0.47
2048	484.3 / 302.9	0.282, 46.26	0.0058	1.02

In Table 2, we compare the aggregation time for various schemes, and measure how the runtime grows with the number of parties. The aggregation time is independent of the signers' weights in all these schemes, with the exception of threshold BLS for which we report the running time for 10-bit weights (higher precision, such as 64-bits is simply infeasible to even evaluate with a handful of parties) – as it is the state-of-the-art, we use the implementation from [63] to report the running time for threshold BLS. Clearly, threshold BLS is not a scalable solution, even with a low precision of 10-bit weights – running time is exponential in the precision.

As proving statements about group operations is expensive in generic SNARKs, we find that the SNARK-based approach – we experiment with both the Groth16 and PLONK provers – is untenable for networks beyond a few hundred parties. Multisig BLS has the

20. Our calculation uses the pre-compiled gas costs for the BLS12-381 curve as defined in EIP-2537 [4]: ECADD costs 600, ECMUL costs 12000, and k pairings cost $115000 + k \cdot 23000$. The gas cost for each 32-byte storage slot is 20000.

21. Alternatively, we could have produced k independent Groth16 proofs, and aggregated them using Bunz et. al. [26] (implemented in [41]), that results in $O(\log(k))$ sized proofs. Recursive composition techniques also exist [23], but they are relatively inefficient.

most efficient aggregation (comprising at most n group additions). That said, for the various applications of threshold signatures that we discussed, we find that hinTS is efficient for networks of practical sizes.

7.2. Signature Size

TABLE 3: Size of Partial and Aggregated Signatures

Scheme	Partial Size	Aggregate Size
SNARK (Groth16)	1 G_1, 1 F	2 G_1, 1 G_2 (192 B)
SNARK (PLONK)	1 G_1, 1 F	9 G_1, 6 F (624 B)
Compact Cert.	1 G_1, 1 F	$>$ 200 G_1 and F ($>$ 15 KB)
Threshold BLS	W G_1	1 G_1 (48 B)
Multisig BLS	1 G_1	1 G_2, n bits (176 B, $n = 2^{10}$)
hinTS	1 G_1	9 G_1, 5 F (592 B)

We report the signature sizes in Table 3, where n denotes the number of parties and W is the weight per party (e.g., $0 < W < 2^{32}$ for 32-bit weights). Depending on the scheme, a signature has several elements: G_1 and G_2 denote group elements (of size 48 and 96 bytes, respectively) from the source groups of the pairings curve and F denotes field elements (of size 32 bytes).

The size of the partial signature is important to study not only because it affects the amount of computation and bandwidth required of each signer, but also the inbound bandwidth required of the aggregator. Except for the threshold BLS scheme, where weights are handled via virtualization, all schemes require each party to output a constant-size signature (one or two group elements) that is verifiable under that party's public key.

Aggregated Signatures are constant-size in hinTS, Threshold BLS, and SNARK-based schemes. Among these, hinTS is the largest with 896 bytes (of which 848 bytes are for the proof); that said, we opine that the improvement in aggregation time compared to the other two schemes is an acceptable tradeoff in practice.

Compact certificates use logarithmic size proofs; for 128-bit security, soundness requires them to output a certificate of size 7.5-12 KB for 100 parties, and roughly 40-250 KB for 10K parties – Table 3 includes a data point for the threshold that is 80% of total weight, and the signature is even larger for lower thresholds.[22] In fact, for a few hundred nodes, the certificate is larger than simply outputting all signatures, due to the overheads of the Merkle paths – their approach is targeted toward networks with millions of nodes. Compact certificates are impractical for our use case, especially since the aggregated signature is published and verified on-chain.

BLS multisig produces a linear size aggregated signature (1 G_1 element and n bits), as the verifier must know which parties have signed. Though asymptotically worse, it fares better in practice compared to hinTS; for reasonable values of n, say 1024, we get a 176-byte signature. Despite the smaller size, multisigs impose a high compute cost on the verifier (discussed below), so we opine that hinTS has an acceptable tradeoff here too.

22. Irrespective of the number of nodes and weights, the certificate contains the following number of signatures (in addition to the Merkle path hashes) for 128-bit security: 1343 for T = 0.55W, 702 for T = 0.6 W, 380 for T = 0.7 W, 272 for T = 0.8W, and 217 for T = 0.9W.

7.3. Verification

TABLE 4: Verification Time and Gas Cost

Scheme	Verifier Ops	CPU Time / EVM Gas
SNARK (Groth16)	3 P, 1 G_1 mul	13 ms / 196K
SNARK (PLONK)	2 P, 18 G_1 mul	7 ms / 377K
Compact Cert.	816 G_1 add 544 G_1 mul 272 H	190 ms / $>$ 672K
Threshold BLS	2 P	3.51 ms / 160K
Multisig BLS	n G_1 add, 2 P	2.5 ms / 774K ($n = 2^{10}$)
hinTS	10 P, 1 G_1 mul	17.5 ms / 395K

Table 4 reports the verification complexity of all schemes, in terms of algebraic operations, CPU time, and EVM gas cost (computed using [4]) – here, we only report the incremental gas cost for verifying each signature, and the fixed cost for setting up the smart contract is deferred to section 7.4. Algebraic operations are of several types: H denotes hash functions, P denotes pairing operations, while group operations in G_1 are either additions or multiplications.

Again, with the exception of compact certificates and BLS multisig, all schemes have constant time. Due to the linear number of group additions, BLS multisig has the highest gas cost for $n > 2^{10}$. The number of partial signatures in a compact certificate is logarithmic asymptotically, but for values of $n \approx 2^{10}$ includes several hundred signatures (depending on the ratio between the threshold and total weight of all parties) – each EdDSA (over Curve25519) signature costs 2000 gas [5].

In terms of gas cost, hinTS is not as competitive as threshold BLS or the Groth16 SNARK (although PLONK has a similar gas cost as hinTS). We contend that applications that are prioritizing gas cost can use the common recursive approach for compressing proofs, wherein the hinTS proof is checked within a SNARK circuit of a Groth16 scheme – as opposed to checking n signatures, the Groth16 prover in this recursive approach would be evaluating 10 P and 1 G_1 operation, which we measure to take approximately 30 seconds.

7.4. Setup Cost

We measure the fixed costs associated with each scheme in Table 5. The setup protocol column measures the cost of running any protocol amongst the parties prior to signing; this includes interactive protocols or silent setup phases for generating the verification key,

TABLE 5: Setup Protocol Complexity and Gas Cost

Scheme	Setup Protocol	EVM Gas
SNARK (Groth16) SNARK (PLONK)	circuit-specific setup powers-of-tau ceremony	300K
Compact Cert.	none	20K
Threshold BLS (AMT DKG [63])	$O(n \log(n))$ DKG ($O(n^2 \log(n))$ complaints)	60K
Multisig BLS	none	60M ($n = 2^{10}$)
hinTS	$O(n \log(n))$ silent setup 46.3s for $n = 2^{10}$	390K

and optionally, a protocol for generating the signing keys (e.g., DKG for generating shares in threshold BLS). The gas cost column measures the cost of storing the verification key material on a smart contract that will later be verifying the threshold signatures.

In the case of SNARK-based schemes, we need a setup phase that only depends on the (max) size of the network, but need not be repeated for each subset of parties up to that max size – this is either a circuit-specific setup in Groth16 (where the circuit verifies up to max size number of EdDSA signatures, or a powers-of-tau ceremony for the KZG commitment scheme in Plonk. The verification key output by this setup phase must be stored on-chain, incurring a modest gas cost.

Compact certificates do not need a trusted setup (beyond the broadcasting of public keys), and only store commitments to the public keys and weights on-chain.

Threshold BLS uses a DKG protocol, for which several schemes have been developed in recent years. Protocols in the vein of Joint-Feldman DKG, such as Tomescu et al. [63], require each party to perform $O(n \log(n))$ work ($O(n^2 \log(n))$ in the event of complaints). On the other hand, the non-interactive DKG protocols [45], [46], [43] require parties to compute expensive NIZKs (but they do not require a complaint phase). In either case, the setup is expensive and must be repeated whenever the access structure changes: parties joining or leaving, changes to weights, or the threshold. On the plus side, threshold BLS only requires the verifier to store a single group element that is the public key, so the gas cost for setup is low.

Multisig BLS does not have any setup phase beyond broadcasting of the public keys, but requires the verifier to store the public key of each party. For any reasonable n, this becomes expensive – for instance, for $n = 2^{10}$, we need 60 million gas ($1450 at the time of writing).

hinTS uses a silent setup (based on some maximum number of signers n), where each party broadcasts a public key that contains auxiliary material of linear size, whose computation requires n group multiplications and $O(n \log(n))$ field operations (for the inverse FFT) – as an example, for $n = 2^{10}$, the computation takes 46.3 seconds. 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. hinTS has a modest gas cost, representing the cost for storing the verification key, some group elements from the CRS, and some pre-processed polynomial commitments (e.g., vanishing polynomial) on the smart contract.

8. Conclusion

We propose hinTS, a novel threshold signature scheme with silent setup. Our scheme avoids a DKG, allows dynamic thresholds and signers, and supports general access structures. Empirical evaluation shows that our scheme is efficient and has applications in state proofs, oracle networks, and off-chain DAO voting.

Acknowledgements. The first author was supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. The second author was supported in part by NSF CNS-1814919, NSF CAREER 1942789, Johns Hopkins University Catalyst award, JP Morgan Faculty Award, and research gifts from Ethereum, Stellar and Cisco. Any opinions, findings and conclusions, or recommendations in this material are those of the authors and do not necessarily reflect the views of the United States Government or DARPA.

References

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

[2] Experiment in gassless voting using snarks. https://ethresear.ch/t/experiment-in-gassless-voting-using-snarks/10927.

[3] `https://docs.gnark.consensys.net/`.

[4] `https://eips.ethereum.org/EIPS/eip-2537`.

[5] `https://eips.ethereum.org/EIPS/eip-665`.

[6] Isokratia. https://nibnalin.me/dust-nib/isokratia.html.

[7] The Time NounsDAO Got Private Voting. https://medium.com/aztec-protocol/the-time-nounsdao-got-private-voting-4336fe4a2c29.

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

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

[10] 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.

[11] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 191–219, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[12] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. In *44nd IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA*, 2023.

[13] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Francois Garillot, Jonas Lindstrom, Ben Riva, Arnab Roy, Alberto Sonnino, Pun Waiwitlikhit, and Joy Wang. Subset-optimized bls multisignature with key aggregation. Cryptology ePrint Archive, Paper 2023/498, 2023. https://eprint.iacr.org/2023/498.

[14] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.

[15] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012. https://link.springer.com/article/10.1007/s13389-012-0027-1.

[16] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[18] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC'03*.

[19] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Heidelberg, December 2018.

[20] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. BLS Signatures. Internet-Draft draft-irtf-cfrg-bls-signature-05, Internet Engineering Task Force, June 2022. Work in Progress.

[21] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[22] Bootle, Jonathan. Efficient Multi-Exponentiation. https://jbootle.github.io/Misc/pippenger.pdf.

[23] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: enabling decentralized private computation. In *IEEE Symposium on Security and Privacy*, pages 947–964. IEEE, 2020.

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

[25] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chainlink Labs*, 2021.

[26] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Heidelberg, December 2021.

[27] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. 2014. https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work.

[28] Lily Chen, Dustin Moody, Andrew Regenscheid, and Angela Robinson. Digital signature standard (dss). 2023. https://www.nist.gov/publications/digital-signature-standard-dss-3.

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

[30] Oana Ciobotaru, Fatemeh Shirazi, Alistair Stewart, and Sergey Vasilyev. Accountable light client systems for pos blockchains. Cryptology ePrint Archive, Paper 2022/1205, 2022.

[31] 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.

[32] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988.

[33] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

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

[35] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2093–2110. USENIX Association, August 2020.

[36] Elizabeth Crites, Chelsea Komlo, Tim Ruffing. From Theory to Practice to Theory: Lessons Learned in Multi-Party Schnorr Signatures. *Real World Crypto – Contributed Talk*, 2023. https://iacr.org/submit/files/slides/2023/rwc/rwc2023/36/slides.pdf.

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

[38] Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-12, Internet Engineering Task Force. Work in Progress.

[39] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[40] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[41] Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical snark aggregation. Cryptology ePrint Archive, Paper 2021/529, 2021. https://eprint.iacr.org/2021/529.

[42] 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.

[43] 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 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 458–487. Springer, 2022.

[44] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

[46] 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.

[47] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[48] 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.

[49] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001.

[50] 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.

[51] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019.

[52] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. Cryptology ePrint Archive, Paper 2022/1592, 2022.

[53] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59, 1991.

[54] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[55] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.

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

[57] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg.

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

[59] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Heidelberg, May 2007.

[60] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

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

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

[63] 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.

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

[65] 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. Proofs of Unforgeability

We first prove some lemmas and then prove the unforgeability.

Lemma 3. *Suppose that at the end of the* $\mathsf{Forgery}-\mathsf{Game}$ *defined in Figure 2, the adversary* $\mathcal{A}$ *outputs a message and signature pair* $(\mathsf{msg}^*, \sigma^*)$ *such that* $\mathsf{Verify}(\mathsf{msg}^*, \sigma^*, T, \mathsf{vk}) = 1$. *Then, with* $1 - \mathsf{negl}(\kappa)$ *probability, we can extract (multivariate) polynomials* $\mathsf{ParSum}(x)$, $B(x)$, $Q_1(x)$, $Q_2(x)$, $Q_x(x, \{\mathsf{sk}_i\}_i)$, $Q_x^*(x, \{\mathsf{sk}_i\}_i)$, $Q_Z(x, \{\mathsf{sk}_i\}_i)$, *and* $\mathsf{aSK}(x, \{\mathsf{sk}_i\}_i)$ *from* $\mathcal{A}$ *such that the following identities hold.*[23]

$$\begin{aligned}
\mathsf{ParSum}(\omega) &= 0,\\
B(\omega^{n+1}) &= 1,\\
\mathsf{ParSum}(x\cdot\omega) - \mathsf{ParSum}(x) - &\\
(W(x) - w\cdot L_{n+1}(x))\cdot B(x) &= Z(x)\cdot Q_1(x),\\
B(x)\cdot(1-B(x)) &= Z(x)\cdot Q_2(x),\\
\mathsf{SK}(x)\cdot B(x) - \mathsf{aSK}(x,\{\mathsf{sk}_i\}_i) &= Q_Z(x,\{\mathsf{sk}_i\}_i)\cdot Z(x)\\
&\quad + Q_x(x,\{\mathsf{sk}_i\}_i)\cdot x,\\
Q_x(x,\{\mathsf{sk}_i\}_i)\cdot x &= Q_x^*(x,\{\mathsf{sk}_i\}_i).
\end{aligned}$$

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

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

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

Proof of Lemma 3. Parse $\sigma^* = (\mathsf{aPK}, \sigma', \pi)$. Let $(\mathsf{ParSum}, B, Q_1, Q_2, Q_x, Q_x^*, Q_Z, \mathsf{aSK})$ group elements contained in the proof π. Since $\mathcal{A}$ is an algebraic adversary, these commitments must be a linear combination of the group elements that $\mathcal{A}$ takes as input. Therefore, the output of $\mathcal{A}$ are multivariate polynomials depending on $\tau, \{\mathsf{sk}_i\}_{i\in S^*}$, and $\{\mathcal{H}(\mathsf{msg}_i)\}_i$.[24] We first argue that these polynomial identities must hold with respect to τ. Next, we argue why ParSum, B, Q_1, and Q_2 must be single variate polynomial depending only on τ and why Q_x, Q_x^*, and Q_Z only depends on τ and $\{\mathsf{sk}_i\}_i$.

First, since the opening proof for the polynomial commitment verifies, by Lemma 1, we know that, with $1 - \mathsf{negl}(\kappa)$ probability, $\mathcal{A}$ must know a set of polynomials such that (1) $\mathsf{ParSum}(\omega) = 0$; (2) $B(\omega^{n+1}) = 1$; and (3) for a random r,

$$\begin{aligned}
\mathsf{ParSum}(r\cdot\omega) - \mathsf{ParSum}(r)&\\
-(W(r) - w\cdot L_{n+1}(r))\cdot B(r) &= Z(r)\cdot Q_1(r),\\
B(r)\cdot(1-B(r)) &= Z(r)\cdot Q_2(r).
\end{aligned}$$

23. Note that only sk_i from the honest parties are treated as variables. Secret keys from malicious parties are not treated as variables as the adversary knows them in the clear.

24. To be precise, we mean the discrete log of $\mathcal{H}(\mathsf{msg}_i)$. For ease of presentation, we abuse notation here.

By Schwarz-Zippel, with all but $\mathsf{poly}(\kappa)/|\mathbb{F}| = \mathsf{negl}(\kappa)$ probability, the following polynomial identity holds.[25]

$$\begin{aligned}\mathsf{ParSum}(x\cdot\omega) - \mathsf{ParSum}(x)\\ -(W(x) - w\cdot L_{n+1}(x))\cdot B(x)\end{aligned} = Z(x)\cdot Q_1(x),$$
$$B(x)\cdot(1-B(x)) = Z(x)\cdot Q_2(x).$$

Furthermore, the following pairing equations hold.

$$e([\mathsf{SK}(\tau)]_1,[B(\tau)]_2)\cdot e(\mathsf{aPK},[1]_1)^{-1} = e([Q_Z(\tau)_1,[Z(\tau)]_2)\cdot e([Q_x(\tau)]_1,[\tau]_2)$$

and

$$e([Q_x(\tau)]_1,[\tau]_2) = e([Q_x(\tau)\cdot\tau]_1,[1]_2).$$

By Lemma 1, the adversary $\mathcal{A}$ must know polynomials $B(x)$, $Q_x(x)$, $Q_x^*(x)$, and $Q_Z(x)$, which satisfy the polynomial identities

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

We now argue that why ParSum, B, Q_1, and Q_2 must only depend on τ. In particular, they are independent of sk_i and $\{\mathcal{H}(\mathsf{msg}_i)\}_i$. If otherwise, suppose B depends on sk_1. Then $B\cdot(1-B)$ will have a quadratic dependence on sk_1. However, the group elements that $\mathcal{A}$ takes as input are all linearly dependent on sk_1. Therefore, the adversary $B(1-B) = Z(x)\cdot Q_2$ as a multivariate polynomial identity will never hold. Hence, with $1-\mathsf{negl}(\kappa)$ probability, both $B(x)$ and $Q_2(x)$ are single variate polynomials depending only on τ. Similarly, if Q_1 depends on, for instance, sk_1, then $Z(x)\cdot Q_1$ will contains a term $\mathsf{sk}_1\cdot Z(x)$. However, all inputs of $\mathcal{A}$ that depend on sk_1 only have degree $\leqslant \mathbb{H}-1$ in terms of τ. Consequently, the polynomial identity of the well-formedness check of ParSum will never satisfy. Therefore, with $1-\mathsf{negl}(\kappa)$ probability, both $\mathsf{ParSum}(x)$ and $Q_1(x)$ are single variate polynomials depending only on τ. Finally, Q_Z and Q_x will not depend on $\{\mathcal{H}(\mathsf{msg}_i)\}_i$ because the adversary never sees $\mathcal{H}(\mathsf{msg}_i)^\tau$. Hence, Q_Z, Q_x will only depends on τ and sk_i, which implies the same for $Q_x^*(x)$. □

Proof of Lemma 4. By Lemma 3, we have $Q_x(x)\cdot x = Q_x^*(x)$. Note that all the input to the adversary that depends on sk_i have degree $\leqslant \mathbb{H}-1$ in terms of τ (for instance, $\mathsf{sk}_i\cdot L_i(\tau)$). In particular, the terms in $Q_x^*(x)$ that depends on sk_i have degree $\leqslant \mathbb{H}-1$ in terms of τ. Therefore, the terms in $Q_x(x,\{\mathsf{sk}_i\}_i)$ that depends on sk_i will have degree $\leqslant |\mathbb{H}|-2$ in terms of τ. □

Proof of Lemma 5. Due to lemma 3, we have the polynomial identity $B(x)\cdot(1-B(x)) = Z(x)\cdot Q_2(x)$. Thus, $B(x)$ must be either 0 or 1 on $\mathbb{H}$. Furthermore, since $\mathsf{ParSum}(\omega) = 0$ and $\mathsf{ParSum}(x\cdot\omega) - \mathsf{ParSum}(x) - (W(x) - w\cdot L_{n+1}(x))\cdot B(x) = Z(x)\cdot Q_1(x)$ hold, it must be that $\mathsf{ParSum}(\omega^{n+1}) = \sum_{i=1}^n B(\omega^i)\cdot w_i = w$. □

25. A polynomial adversary will query the random oracle $\mathsf{poly}(\kappa)$ times. Each query gives a random challenge r. The adversary succeeds as long as one of the challenges is bad. By union bound, the bad event happens with $\mathsf{poly}(\kappa)/|\mathbb{F}|$.

Proof of Lemma 6. Intuitively, this lemma states that aSK will only depend on the sk_i's that have signed msg. In particular, it cannot depend on τ or other honest parties' sk_i.

Observe that $e(\mathsf{aPK},\mathcal{H}(\mathsf{msg}))=e([1]_1,\sigma')$, hence, $\sigma' = \mathcal{H}(\mathsf{msg})^{\mathsf{aSK}}$. Now, the group elements that the adversary sees and are related to $\mathcal{H}(\mathsf{msg})$ are $\mathcal{H}(\mathsf{msg})^{\mathsf{sk}_i}$ for $i\in B^*$ and $\mathcal{H}(\mathsf{msg})^{\mathsf{sk}_i}$. This lemma is, thus, a simple consequence of Lemma 1. □

Given Lemma 3, Lemma 4, Lemma 5, and Lemma 6, we prove unforgeability. Suppose that the adversary $\mathcal{A}$ wins the unforgeability game. By definition, there exists a threshold $T > |B^*\cup A|$ such that $\mathsf{Verify}(\mathsf{msg}^*,\sigma^*,T,\mathsf{vk}) = 1$. Since the adversary's signature verifies, Lemma 3 states

$$\mathsf{SK}(x)\cdot B(x) - \mathsf{aSK} = Q_Z(x,\{\mathsf{sk}_i\}_i)\cdot Z(x) + Q_x(x,\{\mathsf{sk}_i\}_i)\cdot x.$$

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

Now, we extract the set $S' = \{i\in[n] : B(\omega^i)=1\}$ from $B(x)$. Let $\mathsf{aSK}' = (\sum_{i\in S'}\mathsf{sk}_i)/|\mathbb{H}|$. There should also exist an honestly sampled quotient polynomial $Q'_x(x,\{\mathsf{sk}_i\}_i)$ and $Q'_Z(x,\{\mathsf{sk}_i\}_i)$ (for aSK') satisfying

$$\mathsf{SK}(x)\cdot B(x) - \mathsf{aSK}' = Q'_Z(x,\{\mathsf{sk}_i\}_i)\cdot Z(x) + Q'_x(x,\{\mathsf{sk}_i\}_i)\cdot x.$$

Note that Q'_Z and Q'_x can be computed efficiently by the adversary. Taking the difference gives

$$\mathsf{aSK}' - \mathsf{aSK} = (Q_Z - Q'_Z)\cdot Z(x) + (Q_x - Q'_x)\cdot x.$$

Therefore, in order to forge a signature, the adversary successfully computes two polynomials $\Delta_x(x,\{\mathsf{sk}_i\}_i)$ and $\Delta_Z(x,\{\mathsf{sk}_i\}_i)$ such that

$$\mathsf{aSK}' - \mathsf{aSK} = \Delta_Z(x,\{\mathsf{sk}_i\}_i)\cdot Z(x) + \Delta_x(x,\{\mathsf{sk}_i\}_i)\cdot x. \quad (4)$$

Since the total weight in B' is w (Lemma 5), which satisfies $w > \sum_{i\in B^*\cup A} w_i$ (definition of forgery), there must exist some party with index j such that $j\in B'$ and $j\notin B^*$. 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 now argue that Equation 4 implies a contradiction as follows: By Lemma 4, we observe that the terms in $\Delta_x(x)$ that depends on sk_j has degree $\leqslant |\mathbb{H}|-2$. Therefore, the terms in $\Delta_Z(x)\cdot Z(x) + \Delta_x(x)\cdot x$ 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 polynomial identity

$$\mathsf{aSK}' - \mathsf{aSK} = \Delta_Z(x,\{\mathsf{sk}_i\}_i)\cdot Z(x) + \Delta_x(x,\{\mathsf{sk}_i\}_i)\cdot x$$

will not hold. This concludes that the adversary wins the forgery game with probability $\leqslant \mathsf{negl}(\kappa)$.

Appendix B.
Meta-Review

B.1. Summary

This paper provides a threshold signature scheme with silent setup which is also concretely efficient. Silent setup means that signers do not have to interact to setup the public key.

B.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field
- Addresses a Long-Known Issue
- Creates a New Tool to Enable Future Science

B.3. Reasons for Acceptance

1) Provides a Valuable Step Forward in an Established Field: Threshold signatures have been extensively studied; this paper provides a new property, silent setup, and techniques to maintain efficiency and other properties.
2) Addresses a Long-Known Issue: the need for interaction can be problematic in large systems where parties change frequently. Providing silent setup while maintaining efficiency, this paper makes progress towards
3) Creates a New Tool to Enable Future Science: this paper applies techniques from the vector commitments literature to the space of threshold signatures, and could inspire new constructions based on this principle.