Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold

Sourav Das
University of Illinois at
Urbana-Champaign
souravd2@illinois.edu

Philippe Camacho
Espresso Systems
philippe@espressosys.com

Zhuolun Xiang
Aptos Labs
xiangzhuolun@gmail.com

Javier Nieto
University of Illinois at
Urbana-Champaign
jmnieto2@illinois.edu

Benedikt Bünz
Espresso Systems
benedikt@espressosys.com

Ling Ren
University of Illinois at
Urbana-Champaign
renling@illinois.edu

ABSTRACT

Threshold signatures protect the signing key by sharing it among a group of signers so that an adversary must corrupt a threshold number of signers to be able to forge signatures. Existing threshold signatures with succinct signatures and constant verification times do not work if signers have different weights. Such weighted settings are seeing increasing importance in decentralized systems, especially in the Proof-of-Stake blockchains. This paper presents a new paradigm for threshold signatures for pairing and discrete logarithm-based cryptosystems. Our scheme has a compact verification key consisting of only 7 group elements, and a signature consisting of 8 group elements. Verifying the signature requires 8 exponentiations and 8 bilinear pairings. Our scheme supports arbitrary weight distributions among signers and arbitrary thresholds. It requires non-interactive preprocessing after a universal powers-of-tau setup. We prove the security of our scheme in the Algebraic Group Model and implement it using golang. Our evaluation shows that our scheme achieves a comparable signature size and verification time to a standard (unweighted) threshold signature. Compared to existing multisignature schemes, our scheme has a much smaller public verification key.

1 INTRODUCTION

The increasing demand for decentralized Byzantine Fault Tolerant (BFT) applications has resulted in a large scale adoption of threshold signature schemes. Many state-of-the-art BFT protocols utilize threshold signatures to lower communication costs [2, 41, 44, 52, 56, 66]. Furthermore, efforts to standardize threshold cryptosystems are already underway [58]. A threshold signature scheme [11, 43] enables distributing a secret signing key among multiple signers such that each can generate a partial signature over any message using its key share. Given sufficiently many partial signatures, any untrusted aggregator can aggregate the partial signatures into a threshold signature.

Traditionally, threshold signatures have been studied in the *unweighted* setting where each signer has equal weight; in other words, the threshold is measured by the number of signers who signed. However, this is not suitable for many applications. For instance, in Proof-of-Stake (PoS) [36, 44] blockchains and Decentralized Autonomous Organizations (DAO) [33] the *weight* of each signer is determined by the amount of stake they own in the system, and the threshold is measured by the combined stake among those who signed. Another application that calls for the weighted setting is off-chain voting where weighted votes are aggregated offline and only the final aggregated vote is posted to the blockchain.

Another limitation of existing threshold signature schemes is that they only support a single threshold, and this threshold needs to be fixed a priori. This makes them unsuitable for applications that require fine-grained thresholds [32]. For example, Ethereum's Proof-of-Stake consensus protocol Gasper [21] requires a threshold of two-third of the total stake but it is accumulated over multiple blocks [21]. Using a threshold signature with a single fixed threshold will make the protocol lose a lot of flexibility. If the threshold is too small, it may under-utilize some signers and require a larger number of blocks to accumulate to the desired stake threshold. On the other hand, a very high threshold may lead to insufficient signers to cross the threshold. In addition, Byzantine quorum systems [23, 54], or variants of BFT [53, 65] use quorums of different thresholds, and may benefit from multi-threshold signature scheme. Another application that benefits from multi-threshold signatures is oracle networks [32]. In particular, smart contracts that accept data from oracle networks can choose their own set of oracle nodes (with reputation scores) and signing thresholds [40].

Existing approaches and their limitations. Existing (unweighted) threshold signature schemes [11, 49] with n signers and threshold t use a (n, t) Shamir secret sharing [61] so that each signer has one share of the signing key. These schemes have constant signature size, verification key size, and verification time.

Here is a straightforward folklore approach to extend these schemes to support arbitrary weight distributions. The signing key is secret shared using a $(\|\boldsymbol{w}\|_1, t)$ Shamir secret sharing scheme, where $\|\boldsymbol{w}\|_1$ is the total weight of all signers. A signer with weight w then receives w signing keys and plays the roles of w *virtual* signers. (Hence, this approach is also called *virtualization.*) With this approach, a signer's signing cost and partial signature size are proportional to its weight, and the aggregator's cost is proportional to the total number of virtual signers or total weight $\|\boldsymbol{w}\|_1$. These costs can be very expensive in many target applications. For example, in Ethereum PoS, there are more than 500,000 validators (akin to virtual signers in our context) and the count is still increasing.

Alternatively, multisignatures schemes [13] naturally supports arbitrary weight distributions and only requires one signing key per signer, regardless of its weight. However, its main downside is

Table 1: **Comparison of threshold signature schemes. We measure the computation cost in units of group exponentiations.**

Scheme or Approaches	Signing key size	Signing cost per signer	Signature size	Verification key size	Verification cost	Aggregation key size	Aggregation cost	Multiple threshold	Setup
Virtualization	$w_i\ \mathbb{F}$	$O(w_i)$	$1\ \mathbb{G}$	$1\ \mathbb{G}$	$O(1)$	$O(\|\boldsymbol{w}\|_1)$	$O(\|\boldsymbol{w}\|_1)^{\#}$	✗	DKG
Multisignature	$1\ \mathbb{F}$	$O(1)$	n bits + $1\ \mathbb{G}$	$n\ \mathbb{G}$	$O(n)$	$O(n)$	$O(n)$	✓	PKI
SNARK [45]	$1\ \mathbb{F}$	$O(1)$	$3\ \mathbb{G}$	$7\ \mathbb{G}$	$O(1)$	Large	High	✓	MPC
CCoK [55]	2κ	$O(1)$	$O(\kappa\lambda_s \log n)^*$	2κ	$O(\lambda_s \log n)$	$O(n)$	$O(n + \lambda_s \log n)^{\dagger}$	✓	PKI
This work	$1\ \mathbb{F}$	$O(1)$	$8\ \mathbb{G} + 1\ \mathbb{Z}$	$7\ \mathbb{G}$	$O(1)$	$O(n)$	$O(n)^{\#}$	✓	PKI, q-SDH

† The comptuation cost are hashing.

* The λ_s is an soundness parameter.

$^{\#}$ The aggregator in the virtualization approach and our scheme additionally performs $O(\|\boldsymbol{w}\|_1 \log^2 \|\boldsymbol{w}\|_1)$ and $O(n \log n)$ field operations, respectively.

that the verification key size and verification time increase linearly in the number of signers.

Yet another approach to weighted threshold signature is to use generic succinct non-interactive argument of knowledge (SNARK). Here, each signer uses its signing key to compute a partial signature and sends it to an aggregator. The aggregator then generates a SNARK proof that it has seen valid partial signatures from signers with a combined weight of at least t. However, in spite of recent progress, the SNARK proof generation at the aggregator is still prohibitively expensive (§6).

Micali et al. [55] proposed a weighted threshold signature with sublinear signature size and verification time. However, the concrete signature size of their scheme is large. Another drawback of their scheme is that the aggregator needs to collect signatures with combined weights significantly higher than the required threshold.

Our Results. In this paper, we present a new succinct threshold signature paradigm that supports arbitrary weight distribution among signers and supports all possible thresholds simultaneously. We summarize these properties and compare them with existing approaches in Table 1. Crucially, the signature size and the verification time of our scheme are independent of the number of signers n, their weight distributions, and the threshold t. More precisely, our scheme has a small signature size of only 8 elliptic curve group elements, and efficient signature verification involving only 8 group exponentiations and 8 bilinear pairings. Each signer's signing key is a single field element, and the signing cost for each signer is constant (independent of its weight).

Another nice property of our scheme is that, assuming a Public Key Infrastructure (PKI), the setup phase of our scheme (after a universal powers-of-tau setup [27, 57]) is non-interactive, whereas standard threshold schemes need an interactive distributed key generation (DKG) protocol [28, 43].

A key component of our construction is a new efficient inner-product argument (IPA) that proves the inner product between the vector of public keys and a vector indicating the signers who have signed the message. Our IPA uses bilinear pairing, is non-interactive in the algebraic group model, and has a constant proof size and verification time. Looking ahead, our construction can be viewed as a specialized SNARK that utilizes multisignature schemes in a non-black-box manner. However, as we describe in §2, we need to address several challenges to achieve the desirable efficiency.

We also discuss several extensions of our scheme (cf. Appendix B). First, similar to multisignature, our signature scheme can be made accountable with minimal overhead. Second, an aggregator can generate an efficient proof that a certain signer is included in aggregated signature. Third, our scheme enables a new multiverse threshold signature [5] with comparable efficiency and weaker setup assumptions. We believe these results might be of independent interest.

Evaluation. We have implemented (code avaiable at https://github.com/sourav1547/wts) our threshold signature scheme in `golang` using BLS signatures as the underlying signature scheme. We measure the time costs for signing, aggregation and verification, and compares them BLS threshold signature, multisignature, generic SNARK, and the scheme of [55]. Our evaluation confirms the concrete efficiency of our scheme. Using BLS12381 as the underlying elliptic curve, our signature sizes are only 536 bytes, independent of the number of signers. The verification time is also only 8.21 milliseconds. Also, with 4096 signers, the aggregator requires only 71 milliseconds to compute the aggregate signature.

As we envision our signature scheme to be used in blockchain applications, we have implemented an Ethereum smart contract that verifies signatures generated by our scheme. Our evaluation shows that our signature verification takes only 772k gas, while the multisignature scheme with 4096 signers takes more than 23M gas.

Paper organization. The rest of the paper is organized as follows. We present an overview of our signature scheme in §2. We define threshold signature schemes and give the required preliminaries in §3. We describe our scheme in detail in §4, and analyze its security and performance in §5. We present details of our implementation and evaluation results in §6. We discuss related work in §7 and conclude with a discussion in §8.

2 TECHNICAL OVERVIEW

Let $\mathbb{G}$ be an elliptic curve group with $\mathbb{F}$ as its scalar field. Let $g \in \mathbb{G}$ be a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the standard bilinear pairing operation. Our starting point will be the aforementioned weighted multisignature scheme. To be concrete, throughout this paper, we will use the pairing-based BLS multisignature [13], which roughly works as follows.

Each signer samples its signing key independently at random. Let $\boldsymbol{s} = [s_1, s_2, \ldots, s_n] \in \mathbb{F}^n$ be the vector of signing keys. Also, let $\boldsymbol{pk} = [g^{s_1}, g^{s_2}, \ldots, g^{s_n}] \in \mathbb{G}^n$ and $\boldsymbol{w} = [w_1, w_2, \ldots, w_n] \in \mathbb{F}^n$ be the vectors of public keys and weights, respectively. To compute a multisignature on a message m, each signer i uses its signing key to compute the partial signature $\sigma_i = \mathsf{H}(m)^{s_i} \in \mathbb{G}$, and sends it to the aggregator $\mathcal{P}$. Here, $\mathsf{H}(\cdot)$ is a random oracle.

$\mathcal{P}$ validates the partial signatures it receives. Let $I \subseteq [n]$ be the subset of signers from whom the aggregator receives valid partial signatures. Let $\boldsymbol{b} = [b_1, b_2, \ldots, b_n] \in \{0,1\}^n$ be a bit vector where $b_i = 1$ for each $i \in I$ and 0 otherwise. The multisignature on m is then the tuple $(\boldsymbol{b}, \sigma)$, where $\sigma = \prod_{i \in I} \sigma_i$. The threshold of the multisignature is $t = \sum_{i \in I} w_i$.

Upon receiving the signature $(\boldsymbol{b}, \sigma)$ on a message m, the verifier $\mathcal{V}$ computes the aggregated public key $g_\mu = \prod_{i \in I} g^{s_i}$. $\mathcal{V}$ then checks that σ is a valid signature with respect to the g_μ, i.e., $e(g_\mu, \mathsf{H}(m)) = e(g, \sigma)$. If the check is successful, $\mathcal{V}$ accepts the signature as a weighted threshold signature with threshold $t = \sum_{i \in I} w_i$.

2.1 Multisignature to Inner Product Argument

As a stepping stone to our scheme, we formulate the aggregation and verification of the above multisignature scheme as the relation $\mathcal{R}_{\mathsf{TS}}$ as follows. Let $c_{\boldsymbol{pk}}$ and $c_{\boldsymbol{w}}$ be the succinct commitments to the vector $\boldsymbol{pk}$ and $\boldsymbol{w}$, respectively. For now, we assume that the commitments to the $c_{\boldsymbol{pk}}$ and $c_{\boldsymbol{w}}$ are computed honestly and are known to the verifier. Moreover, we assume that the total weight $\|\boldsymbol{w}\|_1 = \sum_{i \in [n]} w_i < |\mathbb{F}|$.

For any message m, $\mathcal{P}$ computes the commitment $c_{\boldsymbol{b}}$ to the bit vector $\boldsymbol{b}$, the aggregated public key g_μ, the threshold t, and the aggregated signature σ. $\mathcal{P}$ then sends the tuple $(m, c_{\boldsymbol{b}}, g_\mu, t, \sigma)$ to $\mathcal{V}$ along with a proof π that these values are computed correctly. $\mathcal{V}$ upon receiving the tuple and the proof validates their correctness with respect to $m, c_{\boldsymbol{pk}}, c_{\boldsymbol{w}}$. We formalize these ideas in the relation $\mathcal{R}_{\mathsf{TS}}$ below. Here we use com to denote a function that takes a vector as input and outputs its succinct commitment.

$$\mathcal{R}_{\mathsf{TS}} : \left\{ \begin{array}{c|r} & \boldsymbol{pk} \in \mathbb{G}^n; c_{\boldsymbol{pk}} = \mathsf{com}(\boldsymbol{pk}) \\ \{c_{\boldsymbol{b}}, g_\mu, \sigma\} \in \mathbb{G}^3 & \boldsymbol{w} \in \mathbb{F}^n, \|\boldsymbol{w}\|_1 < |\mathbb{F}|; c_{\boldsymbol{w}} = \mathsf{com}(\boldsymbol{w}) \\ \wedge & \boldsymbol{b} \in \{0,1\}^n; c_{\boldsymbol{b}} = \mathsf{com}(\boldsymbol{b}) \\ e(g_\mu, \mathsf{H}(m)) = e(g, \sigma) & \boldsymbol{\sigma} \in \mathbb{G}^n; \langle \boldsymbol{\sigma}, \boldsymbol{b} \rangle = \sigma \\ & \langle \boldsymbol{w}, \boldsymbol{b} \rangle \geq t; \langle \boldsymbol{pk}, \boldsymbol{b} \rangle = g_\mu \end{array} \right\}$$

Here, $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ is the vector of partial signatures where we use $\sigma_i = 1_{\mathbb{G}}$ as default for each i with $b_i = 0$.

Note that a secure protocol for $\mathcal{R}_{\mathsf{TS}}$ implies a secure weighted threshold signature scheme. Also, the signature scheme will inherit the efficiency properties of the protocol for $\mathcal{R}_{\mathsf{TS}}$. Hence, we can now focus on designing an efficient protocol for $\mathcal{R}_{\mathsf{TS}}$.

$\mathcal{R}_{\mathsf{TS}}$ as an inner product argument. Our next key idea is to formulate $\mathcal{R}_{\mathsf{TS}}$ as an inner product argument (IPA) between $\mathcal{P}$ and $\mathcal{V}$. The constraints $t = \langle \boldsymbol{w}, \boldsymbol{b} \rangle$ and $g_\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ are naturally inner product constraints. There have also been recent works that use IPA to prove that a committed vector is binary [18, 20]. However, to achieve efficiency comparable to existing threshold signatures, we need to address many challenges, both for $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ and proving that $\boldsymbol{b}$ is a bit vector. We next discuss these challenges in §2.2 and describe our solutions in §2.3.

2.2 Challenges with using existing IPA protocol

To get an efficient threshold signature scheme, the protocol for $\mathcal{R}_{\mathsf{TS}}$ must be succinct, i.e., with sublinear proof size and sublinear verification time. For the inner product $\langle \boldsymbol{w}, \boldsymbol{b} \rangle$, both vectors consist of field elements. We can then use the existing IPA protocol from [25], which has an $O(1)$ proof size and verification time.

The main challenge is the inner product $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle$. This is an inner product between a vector of group elements $\boldsymbol{pk}$ and a vector of field elements $\boldsymbol{b}$. The only known IPA schemes for group elements are the structured key generalized inner product argument (GIPA) from [20] and its transparent setup variant [50]. In GIPA, $\mathcal{P}$ commits to the group element using the commitment schemes from [1]. Then, $\mathcal{P}$ and $\mathcal{V}$ run a interactive protocol similar to the Bulletproofs [18] over the target group. This approach has logarithmic proof size and logarithmic verification time, a moderate cost asymptotically. Its concrete efficiency is much worse. In particular, the proof consists of elements in the target group, which are much larger than the source group elements; similarly, signature verification involves operations in the target group, which are more expensive. Moreover, the prover time is also concretely inefficient as the prover needs to perform $2n$ pairing operations.

The second challenge is that existing IPA schemes for proving a vector binary require $\mathcal{V}$ to compute commitment to a random vector [18, 20]. Computing this commitment requires $\mathcal{V}$ to perform $O(\log n)$ group operations. In §4.2, we will describe an approach that obviates the need for additional random vectors and achieves a $O(1)$ verification cost.

For now, we focus on the main challenge of proving $g_\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$.

2.3 Our Approach

Note that the inner product $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ is nothing but $g^{\langle \boldsymbol{s}, \boldsymbol{b} \rangle}$. So we want $\mathcal{P}$ to give IPA for $\langle \boldsymbol{s}, \boldsymbol{b} \rangle$ in the exponent. The challenge is that $\mathcal{P}$ does not know the secret key vector $\boldsymbol{s}$. Fortunately, signers collectively know $\boldsymbol{s}$, and will assist $\mathcal{P}$ in producing the IPA.

In the rest of this overview, we will first describe the high-level idea of the IPA protocol for $\langle \boldsymbol{s}, \boldsymbol{b} \rangle$ assuming $\mathcal{P}$ knows $\boldsymbol{s}$. Concretely, we will use the IPA protocol of [25]. We then describe how $\mathcal{P}$, with assistance from the signers and without knowing $\boldsymbol{s}$, can efficiently compute the IPA for the inner product $\langle \boldsymbol{s}, \boldsymbol{b} \rangle$ in the exponent. We note that we only describe part of the IPA protocol below to demonstrate our main idea. We refer readers to Appendix A and [25, §5.2] for the complete protocol.

The IPA for field elements. The IPA protocol uses a powers-of-tau of degree n, i.e., $[g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^n}]$, as the common reference string (CRS). Let μ be the claimed inner product, i.e., $\mathcal{P}$ wants to convince $\mathcal{V}$ that $\mu = \langle \boldsymbol{s}, \boldsymbol{b} \rangle$. Let $s(\cdot)$ and $b(\cdot)$ be the two polynomials of degree $n-1$ with $s(\omega^i) = \boldsymbol{s}[i]$, and $b(\omega^i) = \boldsymbol{b}[i]$, respectively. Here, $\omega \in \mathbb{F}$ is a n-th root of unity. Then, let $c_{\boldsymbol{s}} = g^{s(\tau)}$ and $c_{\boldsymbol{b}} = g^{b(\tau)}$ be the commitments of the vectors $\boldsymbol{s}$ and $\boldsymbol{b}$, respectively. We assume that $\mathcal{V}$ has access to the commitments $c_{\boldsymbol{s}}$ and $c_{\boldsymbol{b}}$. The IPA uses the following polynomial identity from [8] which has been used extensively to design efficient SNARKs.

$$s(x)b(x) = q(x) \cdot z_H(x) + x \cdot r(x) + \langle \boldsymbol{s}, \boldsymbol{b} \rangle \cdot n^{-1}.$$

Here, $z_H(x)$ is the degree n polynomial that evaluates to zero at all points ω^i for all $i \in [n]$. Also, $q(x)$ and $r(x)$ are the unique quotient and remainder polynomials each of degree $n-2$ (cf. §3.4).

The IPA for $\langle \boldsymbol{s}, \boldsymbol{b} \rangle$ is the tuple $(g_q, g_r) = (g^{q(\tau)}, g^{r(\tau)})$. $\mathcal{V}$ upon receiving (g_q, g_r) accepts μ as the inner product if the following check pass,

$$e(c_{\boldsymbol{s}}, c_{\boldsymbol{b}}) = e\left(g_q, g^{z_H(\tau)}\right) \cdot e\left(g_r, g^\tau\right) \cdot e\left(g^\mu, g^{1/n}\right) \qquad (1)$$

The IPA protocol is non-interactive and has a constant proof size and verification time. Also, $\mathcal{P}$ incurs a computation cost of $O(n \log n)$ field operations and $O(n)$ group exponentiations.

With this approach, $\mathcal{P}$ needs to compute the tuple $(c_{\boldsymbol{s}}, c_{\boldsymbol{b}}, g^{q(\tau)}, g^{r(\tau)})$. Computing $c_{\boldsymbol{b}}$ is easy as $\mathcal{P}$ knows $\boldsymbol{b}$. Computing the other three would also have been easy had $\mathcal{P}$ known $\boldsymbol{s}$. But in reality, $\mathcal{P}$ needs to compute them only with access to the public keys and the powers-of-tau CRS. We next describe how $\mathcal{P}$ can do so with one-time assistance from all the signers.

Computing the commitment to $\boldsymbol{s}$. In our scheme, each signer i, besides publishing g^{s_i}, also publishes $g^{s_i \mathcal{L}_i(\tau)}$. Here, $\mathcal{L}_i(x)$ is the i-th Lagrange polynomial defined over the set H (cf. §3.4). Using these additional helper values, $\mathcal{P}$ computes $c_{\boldsymbol{s}}$ as:

$$c_{\boldsymbol{s}} = g^{s(\tau)} = \prod_{i \in [n]} g^{s_i \mathcal{L}_i(\tau)}$$

Here, we are assuming a canonical ordering between the signers.

Note that $\mathcal{L}_i(x)$ for each $i \in [n]$ are polynomials of degree $n-1$, and hence can be computed from powers-of-tau CRS using only public operations. Also, given g^{s_i}, the term $g^{s_i \mathcal{L}_i(\tau)}$ is publicly verifiable using a non-interactive zero-knowledge (NIZK) protocol for equality of discrete logarithm.

Computing the IPA proofs efficiently. Even with assistance from signers, computing the IPA proof $(g^{q(\tau)}, g^{r(\tau)})$ seems to require $\mathcal{P}$ to perform $O(n^2)$ group exponentiations and store $O(n^2)$-sized aggregation keys. Both of these quickly become prohibitive for a moderate number of signers. We give a method where $\mathcal{P}$ performs a one-time preprocessing that requires $O(n^2)$ group exponentiations. After that, $\mathcal{P}$ stores a linear-sized aggregation key, and each signature aggregation involves only $O(n)$ group exponentiations.

Non-interactive and transferable preprocessing. Assuming a PKI, our preprocessing step is non-interactive. Each signer samples its signing key independently and publishes the corresponding public key along with necessary helper values, referred to as the *partial aggregation keys* using the PKI. Any aggregator then uses the partial aggregation keys from the signers to compute the linear-sized aggregation key and the constant-sized verification key in the preprocessing step. We also remark that although our preprocessing step costs $O(n^2)$ group exponentiation, its output (aggregation key) is publicly verifiable using only n group exponentiations and 3 pairings. This makes the aggregation key *transferable*, i.e., it is sufficient if one aggregator performs the preprocessing and sends the provable results to other potential aggregators in the system. We will present more details in §4.5.

We want to note that for any given n, the partial aggregation key of each signer is linear in n. If the linear size per signer partial aggregation key is too large to put on a PKI, each signer can directly send them to the aggregator. Note this approach will require special care to handle malicious behavior. For example, if any signer does not send its partial aggregation key to the aggregator, the aggregator will use a $0 \in \mathbb{F}$ as its signing key.

3 SYSTEM MODEL AND PRELIMINARIES

Notations. We use κ to denote the security parameter. We also use κ to denote the size of a group element and the output size of cryptographic objects, for example, the length of the random oracle output. These objects may slightly differ in size in practice, but they are roughly on the same order. Alternatively, one can interpret κ as the largest among them. For any integer a, we use $[a]$ to denote the ordered set $\{1, 2, \dots, a\}$. For two integers a and b where $a < b$, we use $[a, b]$ to denote the ordered set $\{a, a+1, \dots, b\}$. A machine is probabilistic polynomial time (PPT) if it is a probabilistic algorithm that runs in poly(κ) time. We summarize the notations in Table 2.

Table 2: Notations used in the paper

Notation	Description
κ	Security parameter
n, t	Total number of signers and signature threshold
$[n]$	The set $\{1, 2, 3, \dots, n\}$
$\mathbb{G}, \mathbb{F}$	Elliptic curve group with scalar field $\mathbb{F}$.
g, h, v	Random and independent generators of $\mathbb{G}$
w_i, s_i, g^{s_i}	Weight, signing key and public key of signer i
$\boldsymbol{s}$	Vector $[s_1, s_2, \dots, s_n]$ of signing keys.
$\boldsymbol{w}, \lVert \boldsymbol{w} \rVert_1$	Vector of weights of all signers and total weight
ak, vk	Public aggregation key and public verification key
m	Message to be signed
σ_i, σ	Partial signature of signer i and the aggregate signature
$\boldsymbol{b}$	Bit vector indicating the set of valid partial signatures
H	Multiplicative subgroup $\{\omega, \omega^2, \dots, \omega^n\} \subseteq \mathbb{F}$ of order n.
L	Subgroup of order $\geq n-1$ with $H \cap L = \phi$
$\mathcal{L}_{i,H}(x)$	The Lagrange polynomial $\mathcal{L}_{\omega^i,H}(x)$
$\mathsf{H}, \mathsf{H}_{\mathsf{FS}}, \mathsf{H}_{\mathsf{pop}}$	Random oracles
τ	The q-SDH trapdoor
g_i, h_i	$g_i = g^{\mathcal{L}_{i,H}(\tau)}$ and $h_i = h^{\mathcal{L}_{i,H}(\tau)}$

3.1 Threshold Signature

Let there be n signers, denoted with $1, 2, \dots, n$ where the i-th signer has weight w_i. Let $\boldsymbol{w} = [w_1, w_2, \dots, w_n]$ be the vector consisting of weights of all the signers, with total weight $\lVert \boldsymbol{w} \rVert_1 < |\mathbb{F}|$. The constraint $\lVert \boldsymbol{w} \rVert_1 < |\mathbb{F}|$ guarantees that there is no wrap-around while computing the signature threshold. The signers wish to sign a message m and produce a aggregate signature σ, such that σ convinces a client that signers with a combined weight of at least t have signed the message m. We also assume that the client has access to the public verification key of the signature scheme.

A (weighted) threshold signature scheme roughly works as follows. A key generation algorithm takes as input the number of signers n, and a vector of weights $\boldsymbol{w}$. The key generation algorithm generates the public verification key vk and n signing keys $\boldsymbol{s} = [s_1, s_2, \dots, s_n]$, one for each signer. The key generation algorithm additionally outputs a public aggregation key ak. For any given message m, the signers use their signing keys to create partial signatures and send them to an aggregator denoted as $\mathcal{P}$. $\mathcal{P}$, using the aggregation key ak, aggregates valid partial signatures corresponding to a total weight of t, and computes the aggregate signature σ. Any verifier $\mathcal{V}$ with access to vk uses the signature verification algorithm to verify that σ is a valid aggregate signature on message m with respect to the public verification key vk, and is signed by signers of a total weight of at least t.

Definition 3.1 (Weighted Threshold Signature). Let $\{1, 2, 3, \dots, n\}$ be a set of n signers. Let $\boldsymbol{w} = [w_1, w_2, \dots, w_n]$ be the set of weights where w_i represents the weight of signer i. Let $\lVert \boldsymbol{w} \rVert_1 = \sum_{i \in [n]} w_i$. Each signer i has a signing and public key s_i and pk_i, respectively.

Let ak and vk be the public aggregation key and verification key, respectively. With this setup, a weighted threshold signature scheme has the following interfaces.

- $\mathsf{Setup}(1^\kappa) \rightarrow pp$. The setup algorithm Setup takes the security parameter as input and outputs the public parameters pp of the signature scheme.
- $\mathsf{KeyGen}(pp, n, \boldsymbol{w}) \rightarrow vk, ak, [s_1, \ldots, s_n], [pk_1, \ldots, pk_n]$. The key generation algorithm KeyGen takes as input the public parameters pp, the total number of nodes n and a vector of weights $\boldsymbol{w}$. The algorithm outputs the global verification key vk, aggregation key ak, and per signer signing and public key (s_i, pk_i).
- $\mathsf{PSign}(s_i, m) \rightarrow \sigma_i$: Signer i uses the PSign algorithm with its signing key s_i to generate a partial signature σ_i.
- $\mathsf{PVerify}(m, \sigma_i, pk_i) \rightarrow 0/1$: The verify algorithm takes a message m, public key pk_i, and a potential signature σ_i checks whether σ_i is generated using the signing key s_i.
- $\mathsf{Combine}(\{\sigma_i\}, t, ak) \rightarrow \sigma$: On input a set of valid partial signatures of sum total weight of at least $t \leq \|\boldsymbol{w}\|_1$, and the public aggregation key ak, the Combine algorithm generates a aggregate signature σ.
- $\mathsf{Verify}(m, \sigma, vk, t) \rightarrow 0/1$: Outputs 1 only if the message m is signed by signers with total weight of at least t.

The WTS scheme should satisfy the following correctness, security and efficiency properties.

Correctness. For any n, weights $\boldsymbol{w}$ with $\|\boldsymbol{w}\|_1 < q$, and threshold $t \leq \|\boldsymbol{w}\|_1$, an honestly generated partial signature should always pass the partial verification, and an honestly generated aggregate signature should always pass the final verification. Formally,

$$\Pr[\mathsf{PVerify}(m, \mathsf{PSign}(m, s_i), pk_i) = 1] = 1,$$
$$\Pr[\mathsf{Verify}(m, \mathsf{Combine}(\{\sigma_i\}, t, ak), vk, t') = 1] = 1$$

where $t' < t$, and s_i, pk_i for all $i \in [n]$, vk and ak are generated from the Setup and KeyGen algorithms.

Unforgeability. We define the unforgeability game in the presence of an adaptive and rushing adversary. Note that an adaptive adversary can corrupt signers at arbitrary time during the protocol. Also, a rushing adversary can choose its messages depending upon the messages of honest signers.

Let $\mathcal{A}$ be an adaptive adversary which initially corrupts a subset $F_0 \subset [n]$ of signers. $\mathcal{A}$ interacts with the challenger C during the KeyGen protocol. Next, $\mathcal{A}$ interacts with C to receive arbitrarily many partial signatures on messages of its choice. During its interaction with C, $\mathcal{A}$ can corrupt additional signers. Let $F \subseteq [n]$ be the subset of signers $\mathcal{A}$ eventually corrupts. Also, let $w_F = \sum_{i \in F} w_i$ be the total weight of the corrupt signers. Then, $\mathcal{A}$ outputs a message signature pair (σ, m^*, t).

The forgery is considered non-trivial if $\mathsf{Verify}(\sigma, m^*, t, vk) = 1$, and $\mathcal{A}$ has queried partial signatures of weight less than $t - w_F$ on the message m^*. We describe the unforgeability game in Figure 1.

$pp \leftarrow \mathsf{Setup}(1^\kappa)$

$(n, \boldsymbol{w}, F_0) \leftarrow \mathcal{A}_0(pp)$ // *F_0 is the initial set of corrupt nodes*

$(vk, \{pk_i\}, \{s_i\}, ak) \leftarrow \mathsf{KeyGen}(pp, n, \boldsymbol{w})$

$(m^*, \sigma, t) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{sign}}(\cdot,\cdot), \mathcal{O}_{\text{cur}}(\cdot)}(vk, \{pk_i\}_{i \in [n]}, ak)$

$\mathcal{O}_{\text{sign}}(S', m')$ returns partial signature on message m' from each signer in S' and $\mathcal{O}_{\text{cur}}(\cdot)$ lets $\mathcal{A}$ corrupt additional signers.

Winning condition: Output 1 if $\mathsf{Verify}(pp, m^*, \sigma, vk, t) = 1$ and $\mathcal{A}$ has queried $\mathcal{O}_{\text{sign}}(S, m^*)$ and $w_S < t - w_F$. Here, F is the set of signers $\mathcal{A}$ eventually corrupts. Also, w_S and w_F is the sum total of weights of signers in S and F, respectively.

Figure 1: Unforgeability game of our threshold signature

3.2 Pairing based Multisignature

Let $\boldsymbol{pk} = [g^{s_1}, g^{s_2}, \ldots, g^{s_n}]$ be the vector of public keys of signers and let $\boldsymbol{w}$ be the corresponding weight vector. The pairing based multisignature on a message m with claimed weight t, is the tuple $(\sigma, \boldsymbol{b}) \in \mathbb{G} \times \{0,1\}^n$ that satisfy the following:

$$\langle \boldsymbol{w}, \boldsymbol{b} \rangle \geq t \ \text{ and } e(g_\mu, \mathsf{H}(m)) = e(g, \sigma); \ \text{ where } g_\mu = \prod_{i \in [n]} \left(g^{s_i}\right)^{\boldsymbol{b}[i]}$$

Here $\mathsf{H}(\cdot)$ is the random oracle and σ is the aggregated signature defined as:

$$\sigma = \prod_{i \in [n]; b_i = 1} \sigma_i; \quad \text{where } \sigma_i = \mathsf{H}(m)^{s_i} \tag{2}$$

The signing algorithm in the multisignature works as follows. For any given message m, each signer i computes its partial signature $\sigma_i = \mathsf{H}(m)^{s_i}$ and sends it to the aggregator $\mathcal{P}$. $\mathcal{P}$ upon receiving validates them by checking that $e(g^{s_i}, \mathsf{H}(m)) = e(g, \sigma_i)$. Upon receiving valid signatures from signers for total weight t, $\mathcal{P}$ computes the bit vector $\boldsymbol{b} \in \{0,1\}^n$, where $\boldsymbol{b}[i] = 1$ whenever σ_i is valid, otherwise $\boldsymbol{b}[i] = 0$. $\mathcal{P}$ then computes the aggregate signature σ as in equation (2).

3.3 Inner Product Argument

An inner product argument (IPA) is a protocol between a PPT prover $\mathcal{P}$ and an efficient verifier $\mathcal{V}$. Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, an IPA enables the $\mathcal{P}$ to convince $\mathcal{V}$ that $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \mu$, where the verifier only have access to the commitment $c_{\boldsymbol{a}}$ and $c_{\boldsymbol{b}}$ of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively.

IPA has been studied extensively in the recent years [8, 16, 18–20, 25, 50, 51] and has been used repeatedly to design more efficient argument systems, especially SNARKs. Most of these IPA schemes focus on the case where both $\boldsymbol{a}$ and $\boldsymbol{b}$ consists of field elements, i.e., $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$. The most efficient IPA when both $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ has a constant proof size and constant verification time assuming the powers-of-tau as the underlying common reference string (CRS) [25, 51].

As we describe in §2, we require an IPA scheme with succinct proof size and verification time that supports inner product between a vector group and field elements. Known constructions of IPA schemes for inner-product between vector of group and field elements, also known as generalized inner-product arguments (GIPA), are concretely inefficient [20, 50]. In particular, the proof consists of $2 \log n$ $\mathbb{G}_T$ elements, and verifying the signature requires $2 \log n$ $\mathbb{G}_T$ and 7 pairing operations. Moreover, prover time is also concretely inefficient as the prover needs to perform $2n$ pairing operations.

3.4 Polynomial Identities

For any given set $M \subseteq \mathbb{F}$, we define the Lagrange polynomial with respect to M as:

$$\mathcal{L}_{a,M}(x) = \frac{\prod_{b \in M; b \neq a}(x-b)}{\prod_{b \in M; b \neq a}(a-b)} \tag{3}$$

When $|M|= d$, each $\mathcal{L}_{a,M}(\cdot)$ is a degree $d-1$ polynomial. Also, we can write any polynomial $p(x)$ of degree at most $d-1$ as

$$p(x) = \sum_{a \in M} \mathcal{L}_{a,M}(x)p(a) \tag{4}$$

Our threshold signature scheme uses the following two identities about univariate polynomials.

Lemma 3.2 (Polynomial Remainder Lemma). *For any given polynomial $p(\cdot) \in \mathbb{F}[x]$ of degree d, there exists an unique quotient polynomial $q(x) \in \mathbb{F}[x]$ of degree $d-1$ such that for any $a \in \mathbb{F}$*

$$p(x) = q(x)(x-a) + p(a) \tag{5}$$

Lemma 3.3 (Univariate Sumcheck [8]). *Let $H = \{\omega, \omega^2, \ldots, \omega^d\}$ be a multiplicative subgroup of $\mathbb{F}$ of order d. Given two polynomials $a(\cdot), b(\cdot) \in \mathbb{F}[x]$ of degree $d-1$ each, then there exists unique polynomials $q(\cdot)$ and $r(\cdot)$ such that*

$$a(x)b(x) = q(x)z_H(x) + r(x)x + n^{-1} \cdot \sum_{i \in [d]} a(\omega^i)b(\omega^i) \tag{6}$$

here $z_H(x)$ is the vanishing polynomial over the set H, i.e., $z_H(\omega^i) = 0$ for each $i \in [d]$. Also, we can write $z_H(x)$ as

$$z_H(x) = \prod_{i \in [d]} (x - \omega^i) \tag{7}$$

4 THRESHOLD SIGNATURE USING IPA

As we describe in the overview (§2), our approach is to formulate the threshold signature scheme as the relation $\mathcal{R}_{\text{TS}}$ and then present an efficient protocol for $\mathcal{R}_{\text{TS}}$ using an inner product argument. Formally, the $\mathcal{R}_{\text{TS}}$ relation is given as below and we refer the reader to §2 for the intuitive explanation.

$$\mathcal{R}_{\text{TS}} : \left\{ \begin{array}{r|r} & \boldsymbol{pk} \in \mathbb{G}^n; c_{\boldsymbol{pk}} = \mathsf{com}(\boldsymbol{pk}) \\ \{c_{\boldsymbol{b}}, g_\mu, \sigma\} \in \mathbb{G}^3 & \boldsymbol{w} \in \mathbb{F}^n, \|\boldsymbol{w}\|_1 < |\mathbb{F}|; c_{\boldsymbol{w}} = \mathsf{com}(\boldsymbol{w}) \\ \wedge & \boldsymbol{b} \in \{0,1\}^n; c_{\boldsymbol{b}} = \mathsf{com}(\boldsymbol{b}) \\ e(g_\mu, \mathsf{H}(m)) = e(g, \sigma) & \boldsymbol{\sigma} \in \mathbb{G}^n; \langle \boldsymbol{\sigma}, \boldsymbol{b} \rangle = \sigma \\ & \langle \boldsymbol{w}, \boldsymbol{b} \rangle \geq t; \langle \boldsymbol{pk}, \boldsymbol{b} \rangle = g_\mu \end{array} \right\}$$

Handling rogue-key attacks. Although, our signature scheme is agnostic to the specifics of a rogue key attack handling mechanism, for concreteness, we consider the approach used in Boneh et al. [13, §6]. Briefly for any claimed public key g^s, the signer computes the Proof-of-Possession (PoP) as $\pi = \mathsf{H}_{\text{pop}}(g^s)^s$. Here $\mathsf{H}_{\text{pop}} : \{0,1\}^* \rightarrow \mathbb{G}$ is a hash function modeled as a random oracle, that could be constructed from $\mathsf{H}(\cdot)$ using domain separation. The PoP verification procedure accepts if $e(g^s, \mathsf{H}_{\text{pop}}(g^s)) = e(g, \pi)$.

We organize the rest of the section as follows. We begin by describing different parts of the protocol for the relation $\mathcal{R}_{\text{TS}}$. Finally, in §4.8, we combine all these building blocks and present our full threshold signature scheme.

4.1 Setup and Public Parameters

CRS. Let $\mathbb{G}$ be a elliptic curve group with $\mathbb{F}$ as its scalar field. For any given number of signers n, the CRS consists of

$$\left\{ \left[g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^n} \right]; \left[h, h^\tau, h^{\tau^2}, \ldots, h^{\tau^{n-1}} \right]; v \right\} \tag{8}$$

for uniformly random generators $g, h, v \in \mathbb{G}$, and for uniformly random field element $\tau \in \mathbb{F}$.

The CRS also consists of descriptions of two subgroups $H, L \subseteq \mathbb{F}$. Here H is a multiplicative subgroup of order n, and L is a subgroup of order $n-1$ such that $H \cap L = \phi$, i.e., their intersection is empty. Throughout this paper we will work with $H = \{\omega, \omega^2, \ldots, \omega^n\}$, where $\omega \in \mathbb{F}$ is a n-th root of unity. Also, for L, we use a coset of H.

CRS preprocessing. Our scheme uses a CRS that can be computed from the CRS in equation (8) using public operations. More precisely, the preprocessed CRS is:

$$\left\{ \left[g^{\mathcal{L}_1(\tau)}, \ldots, g^{\mathcal{L}_n(\tau)} \right]; \left[h^{\mathcal{L}_1(\tau)}, \ldots, h^{\mathcal{L}_n(\tau)} \right]; v; g^\eta \right\} \tag{9}$$

here we use $\mathcal{L}_i(x)$ to denote the Lagrange polynomial $\mathcal{L}_{\omega^i,H}$ as per equation (3). Also, let $\eta = \sum_{i \in [n]} \mathcal{L}_i(\tau)/\omega^i$.

Note that the CRS in equation (9) is publicly computable from the CRS in equation (8). The computation requires $O(n \log n)$ group exponentiations using number theoretic transform in the exponent. The CRS also consists of

$$\boldsymbol{u} = [u_\ell]_{\ell \in L}; \text{ where } u_\ell = g^{\mathcal{L}_{\ell,L}(\tau)}, \ \forall \ell \in L \tag{10}$$

Here on we will use the following notations.

$$\forall i \in [n] \ g_i = g^{\mathcal{L}_i(\tau)} \ \text{ and } \ h_i = h^{\mathcal{L}_i(\tau)}$$

Verification and aggregation keys. Each signers i samples its signing key $s_i \in \mathbb{F}$, uniformly at random. Let $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$ and $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]$ be the vector of signing keys and weights, respectively. Let $s(\cdot)$ and $w(\cdot)$ be the two polynomials of degree $n-1$ each where $s(\omega^i) = s_i$, and $w(\omega^i) = w_i$, respectively. Then the public verification key vk is the tuple:

$$vk = \left\{ g, h, v, g^{s(\tau)}, g^{w(\tau)}, g^\tau, g^{z_H(\tau)} \right\}$$

The aggregation key ak is:

$$ak = \left\{ \left[g_i^{s_i}\right]_{i \in [n]}; \left[h_i^{s_i}\right]_{i \in [n]}; \left[g^{q_i(\tau)} \right]_{i \in [n]}; \left[g^{\eta s_i}\right]_{i \in [n]} \right\}$$

where $q_i(x)$ is the polynomial of degree $n-1$ defined as:

$$s(x)L_i(x) = q_i(x)z_H(x) + s_i\mathcal{L}_i(x)$$

here $z_H(x)$ is the degree n polynomial that evaluates to zero at all points in H. In particular,

$$z_H(x) = \prod_{i \in [n]} (x - \omega^i) = x^n - 1$$

To assist $\mathcal{P}$ in computing ak, each signer i sends ak_i to $\mathcal{P}$, where

$$ak_i = \left\{ g^{s_i}, g_i^{s_i}, h_i^{s_i}, v^{s_i}, g^{\eta s_i}, \left[u_k^{s_i} \right]_{k \in [n]} \right\} \tag{11}$$

Note that only $\mathcal{P}$ needs to read the linear size ak_i from each signer i. Also, ak_i for each signer i is publicly verifiable using the

CRS and g^{s_i}. Finally, to compute vk, only the element $g_i^{s_i}$ for each signer i is sufficient.

$$g^{s(\tau)} = \prod_{i \in [n]} g_i^{s_i}$$

Similarly, $\mathcal{P}$ computes $g^{w(\tau)} = \prod_{i \in [n]} g_i^{w_i}$. We will describe in §4.5 on how $\mathcal{P}$ computes the terms $[g^{q_i(\tau)}]_{i \in [n]}$.

Remark. As we discuss in §2, assuming a PKI where each signer publishes its partial aggregation key, the setup phase of our signature scheme is non-interactive.

4.2 Proving that the committed vector is binary

Let $b(x)$ be the polynomial of degree $n-1$ such that $b(\omega^i) = \boldsymbol{b}[i]$. Then, if indeed $\boldsymbol{b}$ is binary then the polynomial $b(x)(1-b(x))$ evaluates to 0 for every $x \in H$. Thus, using the polynomial remainder lemma, we get that $b(x)(1-b(x)) = q_b(x) \cdot z_H(x)$.

Given $g^{b(\tau)}$, the commitment to $\boldsymbol{b}$, $\mathcal{P}$ proves that $\boldsymbol{b} \in \{0,1\}^n$, by sending $\pi_b = g^{q_b(\tau)}$ to $\mathcal{V}$. $\mathcal{V}$ upon receiving the proof π_b, accepts the proof if the following checks pass.

$$e\left(g^{b(\tau)}, g^{1-b(\tau)}\right) = e\left(\pi_b, g^{z_H(\tau)}\right)$$

Remark. Note that our approach to proving $\boldsymbol{b}$ a bit vector is not an IPA. Nevertheless, since it shares similarities with the IPA we use, we sometimes refer to it as an IPA for ease of exposition.

Analysis. Completeness is clear. We will prove its soundness in §5. The proof is a single group element and verification requires one group operation and two pairings. $\mathcal{P}$ performs $O(n \log n)$ field operations to compute $q(x)$ using Number Theoretic Transform (NTT). $\mathcal{P}$ then computes $g^{q_b(\tau)}$ using $O(n)$ group exponentiations.

4.3 IPA between public keys and bit vector

Let $\mu = \langle \boldsymbol{s}, \boldsymbol{b} \rangle$ and let $\boldsymbol{pk} = [g^{s_1}, g^{s_2}, \ldots, g^{s_n}]$. We use the following polynomial identity for the inner product $g^{\mu} = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$.

$$s(x)b(x) = q(x)z_H(x) + r(x)x + \mu \cdot n^{-1}. \tag{12}$$

Let $p(x) = r(x)x + \mu \cdot n^{-1}$. Then for each $i \in [n]$, $p(\omega^i) = s_i b_i$. Let $c_{\boldsymbol{b}} = g^{b(\tau)}$ be the commitment to the vector $\boldsymbol{b}$. Then, the proof π for $g^{\mu} = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ is the tuple:

$$\pi = \left(g^{q(\tau)}, g^{r(\tau)}, h^{p(\tau)}, v^{\mu}\right) \tag{13}$$

$\mathcal{V}$ accepts the proof $\pi = (g_q, g_r, h_p, v_\mu)$ and the corresponding inner product g^{μ}, if the following checks pass

$$e\left(g^{s(\tau)}, c_{\boldsymbol{b}}\right) = e\left(g_q, g^{z_H(\tau)}\right) \cdot e\left(g_r, g^{\tau}\right) \cdot e\left(g^{\mu}, g^{1/n}\right) \tag{14}$$

$$e(h_p, g) = e(g_r, h^{\tau}) \cdot e\left(g^{\mu}, h^{1/n}\right) \tag{15}$$

$$e\left(v_\mu, g\right) = e\left(g^{\mu}, v\right) \tag{16}$$

Intuitively, equation (14) checks that the polynomial identity specified in (12) holds with respect to the proof π at τ. The equation (15) and equation (16), checks that g_r and g^{μ} are commitment to a polynomial of degree $n-2$ and a constant, respectively. We elaborate on these checks in Lemma 5.4 and 5.3, respectively.

4.4 Computing the IPA proof

In this section we will describe how $\mathcal{P}$ computes the IPA proof π (in equation (13)) using $O(n)$ group exponentiations. Recall from §2, the difficulty arises because $\mathcal{P}$ needs to compute π having only access to the aggregation public key. We note that to compute π, $\mathcal{P}$ does one-time preprocessing that requires $O(n^2)$ computation costs. We elaborate on the preprocessing cost in §4.5.

Computing $g^{q(\tau)}$. We use the following polynomial identity from [31]. For completeness, we derive it in Appendix D.

$$q(x) = \sum_{i \in [n]} b_i \cdot q_i(x) \;\Rightarrow\; q(\tau) = \sum_{i \in [n]} b_i \cdot q_i(\tau) \tag{17}$$

where the polynomials $q_i(x)$ are defined as:

$$\mathcal{L}_{i,H}(x)s(x) = s_i \cdot \mathcal{L}_{i,H}(x) + z_H(x)q_i(x) \tag{18}$$

The important observation in equation (18) is that the polynomial $q_i(x)$ depends on H and the set of all signers and not on the set of signers who signed a message. This is unlike b_i, whose values gets decided only during the signature aggregation. Thus, $\mathcal{P}$ can precompute $g^{q_i(\tau)}$ for each $i \in [n]$. Then, during signature aggregation, $\mathcal{P}$ computes $g^{q(\tau)}$ using $O(n)$ group operations as:

$$g^{q(\tau)} = \prod_{i \in [n]} \left(g^{q_i(\tau)}\right)^{b_i} \tag{19}$$

Computing $h^{p(\tau)}$. Similarly, we use the following identity from [31]:

$$p(x) = x \cdot r(x) + p(0) = \sum_{i \in [n]} b_i s_i \mathcal{L}_i(x) \tag{20}$$

Equation (20) immediately implies $h^{p(\tau)} = \prod_{i \in [n]} \left(h_i^{s_i}\right)^{b_i}$.

Computing $g^{r(\tau)}$. Using equation (4), we can write $r(x)$ as:

$$r(x) = \sum_{i \in [n]} r(\omega^i)\mathcal{L}_i(x)$$

Since $p(\omega^i) = b_i s_i \; \forall i \in [n]$, we can write $r(\omega^i)$ as:

$$r(\omega^i) = \frac{p(\omega^i) - p(0)}{\omega^i} = \frac{b_i s_i - p(0)}{\omega^i}$$

$$\Rightarrow \sum_{i \in [n]} r(\omega^i)\mathcal{L}_i(\tau) = \sum_{i \in [n]} \frac{b_i s_i \mathcal{L}_i(\tau)}{\omega^i} - \sum_{i \in [n]} \frac{p(0)\mathcal{L}_i(\tau)}{\omega^i} \tag{21}$$

Let r_1 be the first term in the RHS of equation (21). Then,

$$r_1 = \sum_{i \in [n]} s_i \mathcal{L}_i(\tau) \cdot \frac{b_i}{\omega^i} \quad \Rightarrow \quad g^{r_1} = \prod_{i \in [n]} \left(g_i^{s_i}\right)^{b_i/\omega^i}$$

Let r_2 be the second term in (21), and let $\eta = \sum_{i \in [n]} \frac{\mathcal{L}_i(\tau)}{\omega^i}$. Then,

$$r_2 = p(0) \cdot \eta = \eta \cdot \sum_{k \in [n]} p(\omega^k)\mathcal{L}_k(0) = \eta \cdot \sum_{k \in [n]} s_k b_k \mathcal{L}_k(0)$$

$$\Rightarrow g^{r_2} = \prod_{k \in [n]} \left(g^{\eta s_k}\right)^{b_k \mathcal{L}_k(0)}$$

Finally, combining the above, we get that $g^{r(\tau)} = g^{r_1}/g^{r_2}$.

4.5 Computing the preprocessed elements

In this section, we will describe how the $\mathcal{P}$ precomputes $g^{q_i(\tau)}$ for each $i \in [n]$ where $q_i(\cdot)$ is defined as:

$$\mathcal{L}_{i,H}(x)s(x) = s_i \cdot \mathcal{L}_{i,H}(x) + z_H(x)q_i(x) \tag{22}$$

Recall $L \subseteq \mathbb{F}$ with $|L|= n-1$ and $L \cap H = \phi$. This implies that for each $\ell \in L$, $z_H(\ell) \neq 0$. Then, we can write $q_i(x)$ as:

$$q_i(x) = \sum_{\ell \in L} q_i(\ell)\mathcal{L}_{\ell,L}(x) = \sum_{\ell \in L} \left(\frac{\mathcal{L}_{i,H}(\ell)s(\ell) - s_i\mathcal{L}_{i,H}(\ell)}{z_H(\ell)} \right) \mathcal{L}_{\ell,L}(x)$$

$$\Rightarrow q_i(\tau) = \sum_{\ell \in L} \left(\frac{\mathcal{L}_{i,H}(\ell)s(\ell) - s_i\mathcal{L}_{i,H}(\ell)}{z_H(\ell)} \right) \mathcal{L}_{\ell,L}(\tau) \tag{23}$$

Recall from §4.1, the CRS also includes $u_\ell = g^{\mathcal{L}_{\ell,L}(\tau)}$ for each $\ell \in L$. Each signer i also publishes $[u_\ell^{s_i}]$ for each $\ell \in L$. $\mathcal{P}$ uses them to compute $g^{q_i(\tau)}$ for each $i \in [n]$ as follows. Rewrite $q_i(\tau)$ as

$$q_i(\tau) = \sum_{\ell \in L} \left(\frac{\mathcal{L}_{i,H}(\ell)s(\ell)\mathcal{L}_{\ell,L}(\tau)}{z_H(\ell)} \right) - \sum_{\ell \in L} \left(\frac{s_i\mathcal{L}_{i,H}(\ell)\mathcal{L}_{\ell,L}(\tau)}{z_H(\ell)} \right) \tag{24}$$

Let $q_{i,2}$ be the second term of equation (24). Then, $\mathcal{P}$ computes $g^{q_{i,2}}$ using n group exponentiations as:

$$g^{q_{i,2}} = \prod_{\ell \in L} \left(u_\ell^{s_i} \right)^{\mathcal{L}_{i,H}(\ell)/z_H(\ell)}$$

Let $q_{i,1}$ be the first term of equation (24). Let δ_ℓ be such that

$$\delta_\ell = \frac{s(\ell)\mathcal{L}_{\ell,L}(\tau)}{z_H(\ell)} \Rightarrow g^{q_{i,1}} = \prod_{\ell \in L} \left(g^{\delta_\ell} \right)^{\mathcal{L}_{i,H}(\ell)}$$

Note that given g^{δ_ℓ}, computing $g^{q_{i,2}}$ requires $O(n)$ group exponentiations. Next, $\mathcal{P}$ computes g^{δ_ℓ} using the following equations.

$$\delta_\ell = \frac{\mathcal{L}_{\ell,L}(\tau)}{z_H(\ell)} \cdot \sum_{k \in [n]} s_k \mathcal{L}_{k,H}(\ell)$$

$$\Rightarrow g^{\delta_\ell} = \prod_{k \in [n]} \left(u_\ell^{s_k} \right)^{\mathcal{L}_{k,H}(\ell)/z_H(\ell)}$$

Finally, $g^{q_i(\tau)} = g^{q_{i,1}} / g^{q_{i,2}}$.

Verifying $g^{q_i(\tau)}$. We now describe how any external entity can efficiently verify the correctness of $g^{q_i(\tau)}$ for each i. Our idea is to use the standard approach of random linear combination. Let $[g_{q,i}]_{i \in [n]}$ be the claimed values. For an uniformly random $\gamma \in \mathbb{F}$, let $\boldsymbol{\gamma} = [1, \gamma, \gamma^2, \ldots, \gamma^{n-1}]$. Then, the entity computes

$$g_\gamma = \langle [g_i]_{i \in [n]}, \boldsymbol{\gamma} \rangle; \; g_{q,\gamma} = \langle [g_{q,i}]_{i \in [n]}, \boldsymbol{\gamma} \rangle; \; g_{s,\gamma} = \langle [g_i^{s_i}]_{i \in [n]}, \boldsymbol{\gamma} \rangle$$

and checks that the following check holds.

$$e\left(g^{s(\tau)}, g_\gamma\right) = e\left(g^{z_H(\tau)}, g_{q,\gamma}\right) \cdot e\left(g_{s,\gamma}, g\right) \tag{25}$$

Intuitively, equation (25) batch checks the polynomial identity in equation (18) at τ using the standard random linear combinations. Hence, its soundness follows from the Schwartz-Zippel lemma.

4.6 Proving correctness of the threshold

$\mathcal{P}$ uses an IPA to convince $\mathcal{V}$ that $t = \langle \boldsymbol{w}, \boldsymbol{b} \rangle$, precisely the IPA scheme from Appendix A. For completeness, we summarize it next. Recall $c_{\boldsymbol{b}}$ is the commitment to the bit vector $\boldsymbol{b}$.

Proof generation. Let $q_w(x)$ and $r_w(x)$ the polynomials such that:

$$w(x)b(x) = q_w(x)z_H(x) + xr_w(x) + t \cdot n^{-1} \tag{26}$$

Also, let $p_w(x) = xr_w(x) + t \cdot n^{-1}$. Then, the IPA is the tuple

$$\pi = \left\{ g^{q_w(\tau)}, g^{r_w(\tau)}, h^{p_w(\tau)} \right\} \tag{27}$$

Proof verification. $\mathcal{V}$ upon receiving the proof $\pi = (g_{q_w}, g_{r_w}, h_{p_w})$ accepts t as the correct threshold if the following checks pass.

$$e\left(g^{w(\tau)}, c_{\boldsymbol{b}}\right) = e\left(g_{q_w}, g^{z_H(\tau)}\right) \cdot e\left(g_{r_w}, g^\tau\right) \cdot e\left(g^t, g^{1/n}\right); \text{ and}$$

$$e\left(h_{p_w}, g\right) = e\left(g_{r_w}, h^\tau\right) \cdot e\left(g^t, h^{1/n}\right)$$

Note that these verification checks are analogous to the verification checks in equation (14) and (15) for the IPA for $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle$. We omit the check in equation (16), as Lemma 5.3 holds trivially for t.

4.7 Merging IPA proofs

In our scheme so far, $\mathcal{P}$ produces two separate IPA proofs, one for each inner product $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ and $\langle \boldsymbol{w}, \boldsymbol{b} \rangle$. Since, both of these proofs have the same structure, we merge them by taking their random linear combination using the Fiat-Shamir heuristic [35].

$\mathcal{P}$ computes $\xi = \mathsf{H}_{\mathsf{FS}}(g^{s(\tau)}, g^{w(\tau)}, g^{b(\tau)}, g^\mu, t)$, where H_{FS} is a random oracle derived from $\mathsf{H}(\cdot)$ using domain separation. Let $o(x)$ be the polynomial defined as $o(x) = s(x) + \xi w(x)$. This implies,

$$o(x)b(x) = (q_s(x) + \xi q_w(x))z_H(x) + (r_s(x) + \xi r_w(x))x + (\mu + \xi t)n^{-1}$$

here the polynomials $q_w(x), r_w(x)$ are as defined in §4.6, and $q_s(x)$ and $r_s(x)$ defined as below.

$$s(x)b(x) = q_s(x)z_H(x) + r_s(x) + x + \mu \cdot n^{-1} \tag{28}$$

Let $q_o(x), r_o(x)$ and $p_o(x)$ be the polynomials defined as:

$$q_o(x) = q_s(x) + \xi q_w(x)$$

$$r_o(x) = r_s(x) + \xi r_w(x)$$

$$p_o(x) = r_o(x)x + (\mu + \xi t) \cdot n^{-1}$$

$\mathcal{P}$ then sends the tuple $(g^{b(\tau)}, g^\mu, t)$ along with the IPA proof

$$\pi = \left\{ g^{q_o(\tau)}, g^{r_o(\tau)}, h^{p_o(\tau)}, v^\mu \right\} \tag{29}$$

$\mathcal{V}$ upon receiving (g_b, g_μ, t) and the proof (g_q, g_r, h_p, v_μ), first computes $\xi = \mathsf{H}_{\mathsf{FS}}(g_s, g_w, g_b, g_\mu, t)$ and then checks that the following equation holds:

$$e\left(g_s \cdot g_w^\xi, g_b\right) = e\left(g_q, g^{z_H(\tau)}\right) \cdot e\left(g_r, g^\tau\right) \cdot e\left(g_\mu \cdot g^{\xi t}, g^{1/n}\right) \tag{30}$$

Finally, $\mathcal{V}$ accepts it as a valid signature with threshold t if the following additional checks pass.

$$e\left(h_p, g\right) = e\left(g_r, h^\tau\right) \cdot e\left(g_\mu \cdot g^{\xi t}, h^{1/n}\right) \text{ and } e\left(v_\mu, g\right) = e\left(g_\mu, v\right)$$

Analysis. The completeness is clear and we prove its soundness in §5. This brings down the combined proof size of both the IPA to four group elements from seven group elements.

```
$\mathsf{Setup}(1^\kappa, n)$:
On input $1^\kappa$ for the security parameter $\kappa$ produces first the public parameters for the BLS scheme $pp_{\mathrm{BLS}} = \{\mathbb{F}, \mathbb{G}, \mathbb{G}_T, (g, g_T), e(\cdot,\cdot), \mathsf{H}(\cdot)\}$.
Here $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is the bilinear pairing operation, and $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{G}$ is the random oracle. The setup algorithm additionally outputs the following CRS
  - Let $h, v \in \mathbb{G}$ be additional uniform random generators of $\mathbb{G}$
  - Sample $\tau \in \mathbb{F}$;
  - Compute $\boldsymbol{g} := [g, g^\tau, \cdots, g^{\tau^n}]$ and $\boldsymbol{h} := [h, h^\tau, \cdots, h^{\tau^{n-1}}]$
  - From $\boldsymbol{g}$ and $\boldsymbol{h}$ compute
    – $\vec{g_{\mathcal{L}}} := [g^{\mathcal{L}_{1,H}(\tau)}, g^{\mathcal{L}_{2,H}(\tau)}, \ldots, g^{\mathcal{L}_{n,H}(\tau)}]$
    – $\vec{h_{\mathcal{L}}} := [h^{\mathcal{L}_{1,H}(\tau)}, h^{\mathcal{L}_{2,H}(\tau)}, \ldots, h^{\mathcal{L}_{n,H}(\tau)}]$
    – $\vec{u} := [g^{\mathcal{L}_{1,L}(\tau)}, g^{\mathcal{L}_{2,L}(\tau)}, \ldots, g^{\mathcal{L}_{n,L}(\tau)}]$
  - Compute $g^\eta$ where $\eta = \sum_{i\in[n]} \mathcal{L}_i(\tau)/\omega^i$ using $\vec{g_{\mathcal{L}}}$.
Output $pp := (pp_{\mathrm{BLS}}, \vec{g_{\mathcal{L}}}, \vec{h_{\mathcal{L}}}, \vec{u}, h, v, g^\eta)$

$\mathsf{KeyGen}(pp, n, \boldsymbol{w})$:
  - Each signer $i$ samples its signing key $s_i \leftarrow \mathbb{F}$ uniformly at random.
  - Let $\boldsymbol{pk} := [g^{s_1}, \ldots, g^{s_n}]$ be the vector of public keys.
  - Compute $vk := \{g, h, v, g^{s(\tau)}, g^{w(\tau)}, g^\tau, h^\tau, g^{z_H(\tau)}\}$
  - Compute $ak := \left\{[g_i^{s_i}]_{i\in[n]} ; [h_i^{s_i}]_{i\in[n]} ; [g^{q_i(\tau)}]_{i\in[n]} ; [g^{\eta s_i}]_{i\in[n]}\right\}$
    with help from the signers (see §4.1 and §4.5)
Output $(vk, ak)$

$\mathsf{PSign}(m, s_i)$: Output $\sigma_i = \mathsf{H}(m)^{s_i}$

$\mathsf{PVerify}(m, \sigma_i, g^{s_i})$: Output 1 if $e(g^{s_i}, \mathsf{H}(m)) = e(g, \sigma_i)$, otherwise 0.

$\mathsf{Combine}(\{\sigma_i\}, t', ak)$:
  - Let $I$ be the indices corresponding to valid partial signatures.
  - Compute the bit vector $\boldsymbol{b}$ where $\boldsymbol{b}[i] = 1,\ \forall i \in I$ and 0 otherwise.
  - Compute commitment to $\boldsymbol{b}$, $g_b := g^{b(\tau)}$ and the proof $g_{q_b} := g^{q_b(\tau)}$.
  - Compute the aggregated public key $g_\mu = g^\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$.
  - Compute $\sigma_{\mathrm{BLS}} = \prod_{i\in I} \sigma_i$.
  - Compute the IPA proof $\pi = \{g^{q_o(\tau)}, g^{r_o(\tau)}, h^{p_o(\tau)}, v^\mu\}$ (cf. §4.7)
Output $\sigma := (g_\mu, g_b, g_{q_b}, \sigma_{\mathrm{BLS}}, \pi, t')$

$\mathsf{Verify}(m, \sigma, vk, t)$:
  - Parse $\sigma$ as $(g_\mu, g_b, g_{q_b}, \sigma_{\mathrm{BLS}}, \pi, t')$.
  - Check correctness of bit vector as
        $e(g_b, g/g_b) = e(g_{q_b}, g^{z_H(\tau)})$    (31)
  - Verify the IPA proof $\pi$
    – Compute $\xi = \mathsf{H}_{\mathrm{FS}}(g_s, g_w, g_b, g_\mu, t')$ where $g_s, g_w$ are part of $vk$.
    – Check the three equations below hold:
        $e\left(g_s \cdot g_w^\xi, g_b\right) = e\left(g_q, g^{z_H(\tau)}\right) \cdot e\left(g_r, g^\tau\right) \cdot e\left(g_\mu \cdot g^{\xi t'}, g^{1/n}\right)$    (32)
        $e\left(h_p, g\right) = e\left(g_r, h^\tau\right) \cdot e\left(g_\mu \cdot g^{\xi t'}, h^{1/n}\right)$    (33)
        $e(v_\mu, g) = e(g_\mu, v)$    (34)
  - Check the BLS signature
        $e(g_\mu, \mathsf{H}(m)) = e(g, \sigma_{\mathrm{BLS}})$    (35)
  - Finally check that $t' \geq t$.
```

Figure 2: Our signature scheme.

4.8 Threshold signature design

Combining all the above, we get the following threshold signature scheme. We summarize the construction in Figure 2.

Setup. The algorithm Setup produces the parameters for the BLS signature scheme $pp_{\mathrm{BLS}} = \{\mathbb{F}, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot), \mathsf{H}(\cdot)\}$ and a CRS of size linear in n, the number of signers. More precisely, the algorithm Setup samples a uniformly random generators $h, v \in \mathbb{G}$ and $\tau \in \mathbb{F}$ and computes $\boldsymbol{g} := [g, g^\tau, \ldots, g^{\tau^n}]$ and $\boldsymbol{h} := [h, h^\tau, \cdots, h^{\tau^{n-1}}]$.

Then, as described in §4.1, using the Lagrange polynomials defined over the multiplicative subgroups H, L, and the vectors $\boldsymbol{g}, \boldsymbol{h}$, the Setup algorithm computes the following:

- $\vec{g_{\mathcal{L}}} := [g_1, g_2, \cdots, g_n] = [g^{\mathcal{L}_{1,H}(\tau)}, g^{\mathcal{L}_{2,H}(\tau)}, \ldots, g^{\mathcal{L}_{n,H}(\tau)}]$
- $\vec{h_{\mathcal{L}}} := [h_1, h_2, \cdots, h_n] = [h^{\mathcal{L}_{1,H}(\tau)}, h^{\mathcal{L}_{2,H}(\tau)}, \ldots, h^{\mathcal{L}_{n,H}(\tau)}]$
- $\vec{u} := [u_1, u_2, \cdots, u_n] = [g^{\mathcal{L}_{1,L}(\tau)}, g^{\mathcal{L}_{2,L}(\tau)}, \ldots, g^{\mathcal{L}_{n,L}(\tau)}]$

Finally it computes g^η for $\eta = \sum_{i\in[n]} \mathcal{L}_i(\tau)/\omega^i$ using $\vec{g_{\mathcal{L}}}$, and outputs $pp := (pp_{\mathrm{BLS}}, \vec{g_{\mathcal{L}}}, \vec{h_{\mathcal{L}}}, \vec{u}, h, v, g^\eta)$ as the CRS.

Key generation. Each signer i samples its signing key s_i, and publishes the corresponding public key $pk_i = g^{s_i}$ the and the proofs-of-possession. Concretely, for proof-of-possession, we use the approach from [13, §6].

As we describe in §4.1, in order to assist the aggregator $\mathcal{P}$, each signer i additionally computes its partial aggregation key ak_i and sends it to $\mathcal{P}$ where

$$ak_i := \left\{g^{s_i}, g_i^{s_i}, h_i^{s_i}, v^{s_i}, g^{\eta s_i}, \left[u_k^{s_i}\right]_{k\in[n]}\right\} \tag{36}$$

Note that given pk_i, $\mathcal{P}$ can check validity of ak_i using pairings or NIZK proofs for discrete logarithm equality.

As we describe in §4.5, $\mathcal{P}$ uses ak_i to compute the aggregation key ak defined as

$$ak := \left\{[g_i^{s_i}]_{i\in[n]} ; [h_i^{s_i}]_{i\in[n]} ; \left[g^{q_i(\tau)}\right]_{i\in[n]} ; [g^{\eta s_i}]_{i\in[n]}\right\} \tag{37}$$

Note that similar to ak_i, the aggregation key ak is also publicly verifiable given pk_i and the CRS (cf. §4.5). Next, $\mathcal{P}$ computes the $(g_s, g_w) = (g^{s(\tau)}, g^{w(\tau)})$, the commitment to the public keys the weights using the weight vector $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]$, $g_i^{s_i}$ for each i, and the CRS. The public verification key vk of our scheme is

$$vk = \left\{g, h, v, g_s, g_w, g^\tau, h^\tau, g^{z_H(\tau)}\right\}$$

Computing partial signature. As in standard BLS signature, for any message m, signer i computes its partial σ_i as $\mathsf{H}(m)^{s_i}$.

Verifying partial signature. As in standard BLS signature, a partial signature σ_i from signer i is valid if $e(g^{s_i}, \mathsf{H}(m)) = e(g, \sigma_i)$.

Combining partial signatures. Upon receiving valid partial signatures σ_i, $\mathcal{P}$ first checks that the total weight of these partial signatures is greater than the required threshold t.

(1) Let $\boldsymbol{b} \in \{0,1\}^n$ be the vector that indicates the set of valid signers. As in §4.2, $\mathcal{P}$ computes the polynomial commitment $g^{b(\tau)}$, along with the proof $g_{q_b} = g^{q_b(\tau)}$ for $\boldsymbol{b}$ being a bit vector. Also, $t' = \langle \boldsymbol{w}, \boldsymbol{b} \rangle$ is the signature threshold.
(2) $\mathcal{P}$ computes the aggregated public key $g_\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ and the BLS aggregated signature $\sigma_{\mathrm{BLS}} := \prod_{i:b[i]=1} \sigma_i$.
(3) Finally, $\mathcal{P}$ computes the IPA proof π as per §4.7 to convince $\mathcal{V}$ that $g_\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$ and $t' = \langle \boldsymbol{w}, \boldsymbol{b} \rangle$.

Verifying the aggregate signature. The verifier $\mathcal{V}$ upon receiving the aggregate signature $\sigma = (g_\mu, g_b, g_{q_b}, \sigma_{\mathrm{BLS}}, \pi, t')$ validates it by checking that: (i) g_{q_b} is a correct proof of g_b being a commitment to a bit vector; (ii) π is a valid IPA for the aggregated public key g_μ and the threshold t'; and, (iii) σ_{BLS} is a valid BLS signature on the message m with respect to public key g_μ.

Optimized verification. As in Figure 2, verifying the aggregate signature requires 1 exponentiation and 13 pairings. We further reduce the verification cost to 8 exponentiations and 8 pairings using the standard random linear combination approach. More precisely, $\mathcal{V}$ samples a uniformly random $\gamma \in \mathbb{F}$ and checks the following:

$$e\left(g_\mu, v \cdot \mathsf{H}(m)^\gamma\right) = e(v_\mu, g) \cdot e\left(g, \sigma_{\text{BLS}}^\gamma\right) \tag{38}$$

$$\begin{aligned} e\left(g_s \cdot g_w^\xi \cdot (g/g_b)^\gamma, g_b\right) \cdot e\left(h_p, g^{\gamma^2}\right) = e\left(g_q \cdot g_{qb}^\gamma, g^{z_H(\tau)}\right) \cdot \\ e\left(g_r, g^\tau \cdot h^{\tau\gamma^2}\right) \cdot e\left(g_\mu \cdot g^{\xi t'}, g^{1/n} \cdot h^{\gamma^2/n}\right) \end{aligned} \tag{39}$$

Intuitively, the check in equation (38) merges the checks in equations (35) and (34), by taking their random linear combination. Similarly, the check in equation (39), merges the checks in equations (32), (31), and (33). Similar to Lemma 5.5, the soundness of these optimized checks follows from the Schwartz-Zippel lemma.

5 ANALYSIS

We prove security of our threshold signature scheme in the Algebraic Group Model (AGM). We will prove the security in two parts. First, we will prove that assuming hardness of q-SDH in the AGM, our protocol for $\mathcal{R}_{\text{TS}}$ is knowledge sound. More precisely, for any PPT adversary $\mathcal{A}$ that successfully convinces a verifier $\mathcal{V}$ with respect to a committed key $g^{s(\tau)}$ and committed weights $g^{w(\tau)}$, then there exists an efficient extractor $\mathcal{E}$, who interacts with $\mathcal{A}$ and outputs a bit vector $\boldsymbol{b}$ such that $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle = g^\mu$ and $\langle \boldsymbol{w}, \boldsymbol{b} \rangle \geq t$.

We then use the knowledge soundness of the protocol for $\mathcal{R}_{\text{TS}}$ and hardness of co-CDH assumption to prove that our threshold signature scheme is existentially unforgeable as per the security game in Figure 1. Our unforgeability proof follows the security proof of Boneh et al., [13, Theorem 5]. We also follow the proof-of-possession approach adopted in that paper.

5.1 Knowledge soundness of the IPA protocol.

Throughout our analysis, we use the following theorem which we prove in Appendix E.

Theorem 5.1. *Let $\boldsymbol{g_m} = [g, g^\tau, g^{\tau^2}, \dots, g^{\tau^m}]$ be the q-SDH parameters for any given m. Assuming hardness of q-SDH, no PPT adversary $\mathcal{A}$ on input $\boldsymbol{g_m}$ can output a non-zero polynomial $a(\cdot)$ of degree $\leq m$ such that $a(\tau) = 0$.*

We prove the knowledge soundness of our protocol for $\mathcal{R}_{\text{TS}}$ in parts. Lemma 5.2, which we prove in Appendix E, first shows knowledge soundness of the bit vector relation. We will then prove security of the remaining IPA protocol in Lemma 5.6 and 5.5.

Lemma 5.2 (Bit vector). *Assuming hardness of q-SDH in the AGM, the protocol for proving that the committed vector $\boldsymbol{b}$ is binary is knowledge sound with probability* $1 - \mathsf{negl}(\kappa)$.

Recall that the IPA proof consists of tuple (g_μ, v_μ) such that $e\left(g_\mu, v\right) = e\left(v_\mu, g\right)$. For $g_\mu, v_\mu \in \mathbb{G}$, let $\boldsymbol{\mu}, \hat{\boldsymbol{\mu}}$ be the vectors such that $g_\mu = \langle \boldsymbol{\mu}, \boldsymbol{g_n} \rangle$ and $v_\mu = \langle \hat{\boldsymbol{\mu}}, \boldsymbol{g_n} \rangle$, respectively. Let $\mu(x)$ and $\hat{\mu}(x)$ be the polynomials defined using the elements of $\boldsymbol{\mu}$ and $\hat{\boldsymbol{\mu}}$ as coefficients, respectively. Then, in Appendix E, we prove the following.

Lemma 5.3. *Assuming hardness of q-SDH in the AGM, $\mu(x)$ is a constant polynomial with probability* $1 - \mathsf{negl}(\kappa)$.

The next part in our proof is to bound the degree of the polynomial $r_o(x)$ given the g_r and h_p that satisfy the following constraint:

$$e\left(h_p, g\right) = e\left(g_r, h^\tau\right) \cdot e\left(g_\mu \cdot g^{\xi t}, h^{1/n}\right) \tag{40}$$

Again, since $\mathcal{A}_{\text{IPA}}$ is algebraic, let $\boldsymbol{p}, \boldsymbol{r}$ be the vectors such that $h_p = \langle \boldsymbol{p}, \boldsymbol{g_n} \rangle$ and $g_r = \langle \boldsymbol{r}, \boldsymbol{g_n} \rangle$, respectively. Also, let $p(x)$ and $r(x)$ be the polynomials defined using the elements of $\boldsymbol{p}$ and $\boldsymbol{r}$ as coefficients, respectively. Then, in Appendix E, we prove the following.

Lemma 5.4. *Assuming hardness of q-SDH in the AGM, $r(x)$ is a polynomial of degree at most $n - 2$ with probability* $1 - \mathsf{negl}(\kappa)$.

Given the claimed aggregated public key g_μ and claimed threshold t, let $g_\nu = g_\mu g^{\xi t}$. Recall from §4.7, $o(x)$ is the polynomial defined as $o(x) = s(x) + \xi w(x)$. Let $\boldsymbol{o} = [o(\omega), o(\omega^i), \dots, o(\omega^n)]$. Then, in $\mathcal{R}_{\text{TS}}$, $\mathcal{P}$ convinces $\mathcal{V}$ that $g_\nu = g^{\langle \boldsymbol{b}, \boldsymbol{o} \rangle}$. Then, in the next lemma, we prove that, except with negligible probability, correctness of g_ν implies correctness of g_μ and t.

Lemma 5.5. *Let $\boldsymbol{o}$ be the vector defined as above. If $g_\nu = g^{\langle \boldsymbol{b}, \boldsymbol{o} \rangle}$, then except with probability* $\mathsf{negl}(\kappa)$, *$g_\mu = g^{\langle \boldsymbol{b}, \boldsymbol{s} \rangle}$ and $t = \langle \boldsymbol{b}, \boldsymbol{w} \rangle$.*

We now use Lemma 5.3, Lemma 5.4 and theorem 5.1 to prove that the claimed g_ν is indeed $g^{\langle \boldsymbol{b}, \boldsymbol{o} \rangle}$.

Lemma 5.6 (Sumcheck). *Let $\boldsymbol{o}$ be the vector as defined above and $\boldsymbol{b}$ be the bit vector as per Lemma 5.2. Let g_μ and t be the claimed aggregated public key and threshold, respectively. Let $g_\nu = g_\mu g^{\xi t}$. Then, assuming hardness of q–SDH in the AGM, $g_\nu = g^{\langle \boldsymbol{b}, \boldsymbol{o} \rangle}$.*

Finally, combing Lemma 5.2, Lemma 5.6, and Lemma 5.5, we get the following main theorem.

Theorem 5.7 ($\mathcal{R}_{\text{TS}}$ Knowledge-Soundness). *Assuming hardness of* q-SDH *in the AGM, the protocol for the relation $\mathcal{R}_{\text{TS}}$ is knowledge sound with* $1 - \mathsf{negl}(\kappa)$ *probability.*

5.2 Security of threshold signature scheme

We prove the security of our signature scheme in the presence of an *adaptive* adversary. In particular, we prove that if an adversary $\mathcal{A}_{\text{TS}}$ produces a non-trivial forgery of our threshold signature scheme, then we use $\mathcal{A}_{\text{TS}}$ in a black-box manner to design an adversary $\mathcal{A}_{\text{coCDH}}$ that breaks the co-CDH assumption. Recall from §3.1, a forgery is non-trivial when $\mathcal{A}_{\text{TS}}$ produces a signature with threshold t, while querying partial signatures from honest signers of weight less than $t - w_F$. Here w_F is the weight of the corrupt signers. Due to space restrictions, we describe our reduction in Appendix E and only state the main theorem next.

Theorem 5.8. *For any PPT adversary $\mathcal{A}_{\text{TS}}$, if $\mathcal{A}_{\text{TS}}$ successfully creates a non-trivial forgery with probability ϵ, then $\mathcal{A}_{\text{coCDH}}$ breaks the q-SDH assumption with probability $\epsilon \cdot \delta_{\mathcal{R}_{\text{TS}}}/\mathsf{poly}(n, q_H)$.*

Here $\delta_{\mathcal{R}_{\text{TS}}}$ is the knowledge soundness error in Theorem 5.7, and q_H is the number of random oracle queries $\mathcal{A}$ makes.

5.3 Performance

The CRS and aggregation key each consist of $O(n)$ group elements. The verification key consists of 7 group elements. The one-time preprocessing requires $O(n^2)$ computation costs to compute n group elements, each requiring n group exponentiations. The per signer signing key is a single field element and signing requires one group exponentiation. During signature aggregation, $\mathcal{P}$ performs $O(n)$ group exponentiations, $O(n \log n)$ field operations. The signature consists of 8 group elements and 1 integer for specifying the threshold. Verification requires 8 exponentiations and 8 pairings.

6 IMPLEMENTATION AND EVALUATION

We implement and evaluate our threshold signature scheme in golang. Our implementation is publicly available at https://github.com/sourav1547/wts. For our experiments, we only implement the computation component without any networking. We use the BLS12-381 pairing based curve implementation from gnark-crypto [17]. We also use (for both in our implementation and the existing works) the multi-exponentiation of group elements using Pippenger's method [10, §4] to increase the efficiency of the aggregator. All experiments are run on a *t3.2xlarge* Amazon Web Service (AWS) instance with 32 GB RAM and 8 virtual cores.

We measure the computation cost in terms of latency for preprocessing, signing, verification, and aggregation algorithm. Throughout our evaluation, we use the pairing based BLS signature [15] as our underlying signature scheme. We compare our scheme with the following schemes, as described below. (1) generic SNARK approach, (2) compact certificate in Micali et. al. [55], (3) (vanila) BLS threshold signature [11], and (4) BLS multisignature [13, §6].

Threshold signature using generic SNARK. We consider the following generic SNARK construction. Each signer has one signing key and a weight. The signer signs only once using its signing key. The aggregator functions as a SNARK prover $\mathcal{P}$ who convinces the verifier $\mathcal{V}$ that it knows a set of valid signatures, each with distinct public key, with a total weight greater than or equal to the desired threshold. We build the SNARK prover atop the open source SNARK prover implementation of [5]. We use the gnark library [17] to create the SNARK proof. We choose the most SNARK-friendly signature scheme available in the gnark library, which is the EdDSA signature — with gnark frontend. A single EdDSA verification produces 6.5k constraints in the Groth16 proof system [45], and 13.6k constraints in the PLONK system [38]. For this experiment, we also assume that the verifier has the list of all public-keys and all weights are equal. Note that, it is also possible to construct the proof with respect to a commitment of the public keys and distinct weights. This will further increase the running time of the aggregator. We want to note that, the EdDSA signature implementation of gnark uses MiMc [3] hash function as the underlying random oracle.

Compact certificates of knowledge (CCoK) [55]. We benchmark CCoK based on their open source implementation of Algorand [4]. We use EdDSA signature over the curve25519 elliptic curve as the underlying signature scheme, and SHA256 implementation from `libsodium` as the underlying hash function. We adapted existing benchmarks for their implementation in the unweighted setting for our desired threshold values. Also, for any given threshold t, we consider the collected weight to be $1.25t$. Note that the CCoK scheme requires an additional soundness security parameter, which is then used to compute the number of Merkle paths to be revealed in the certificate. For all benchmarks, we use pick the parameter to achieve 128 bit of security.

Table 3: Key generation and preprocessing time of our approach.

Number of signers	64	256	1024	4096
Key generation time (milliseconds)	2.25	7.11	28.34	112.23
Preprocessing time (seconds)	0.10	0.90	10.65	149.11

BLS threshold and multisignature. We implement the virtualization approach with BLS threshold signature and the BLS multisignature scheme [13, §6] as described in §1, §3.2, and §7. We do not include the cost of the DKG for the virtualization approach. For BLS threshold signature, we use $O(t \log^2 t)$ time algorithm from [64] to compute the Lagrange coefficients.

With our evaluation we seek to demonstrate that our scheme supports arbitrary weight distribution and multiple thresholds while maintaining a signature size and verification time comparable to that of standard threshold signature and multisignature schemes. Recall, that existing threshold signature schemes are very inefficient with arbitrary weight distribution. Alternatively, multisignature schemes require a linear-size public verification key. Our evaluation also illustrates that existing off-the-self SNARKs (as described below) are inefficient when used as a threshold signature scheme.

6.1 Evaluation Setup

With the exception of BLS threshold signature and CCoK, the aggregation time and signature size of the other schemes depend solely on the number of signers used to compute the final signature. To evaluate these schemes, we begin by evaluating all signatures in the unweighted setting with varying numbers of signers to aggregate, specifically with $t = 64, 256, 1024$, and 4096.

For BLS threshold signature, the aggregation time only depends on the required threshold t. However, in the weighted setting, t may be much larger than the total number of signers. Thus, to examine the effect of weights, we also evaluate the BLS threshold signature scheme with $n = t = 2^{15}$ and 2^{16}.

Finally, since in CCoK, the aggregator needs to collect a larger fraction of signatures than t, we use $n = 2t$ while evaluating CCoK. Note that, CCoK's performance, depends on the actual weight distribution. Nevertheless, our unweighted evaluation shows that the signature size of CCoK is more than 80 KBytes even with $t = 256$. Thus, we do not evaluate CCoK in the weighted setting.

6.2 Evaluation Results

Preprocessing and key generation time. In Table 3 we report the per-signer key generation time and the preprocessing time i.e., the time an aggregator takes to compute the the aggregation key, of our scheme. Observe that generating keys in our scheme is very efficient, i.e., it only takes 112 milliseconds to generate the keys with 4096 signers. Also, the key generation time grows only linearly with the number of signers. In comparison, the preprocessing time is much higher, i.e., 149 seconds for 4096 signers, and grows quadratically with number of signers. As we mention before, this is because, an aggregator needs to perform $O(n^2)$ group exponentiations to

Table 4: Unweighted aggregation time (in milliseconds).

t	64	256	1024	4096
BLS Threshold	3.29	10.93	38.86	150.79
Multisignature	0.15	0.54	2.69	11.23
CCoK	20.53	79.36	313.12	1246.76
Groth16	6535.86	25695.95	—	—
Plonk	46081.93	—	—	—
Our approach	3.49	7.77	21.56	71.02

compute the aggregation key. Fortunately, as we discuss in §4.5 the aggregation key is efficiently verifiable, hence, can be delegated to external entities.

Signature aggregation time. We report the unweighted signature aggregation time in Table 4. The reported aggregation time does not include the time the aggregator spends verifying the signatures, which is identical in BLS threshold, multisignature, and our scheme.

Note that multisignature scheme have the shortest aggregation time, as aggregation in a multisignature scheme only requires $O(n)$ group operations. Contrary to that, our approach and BLS threshold signature scheme need $O(n)$ group exponentiations and $O(n \log n)$ field operations. The longer aggregation time in CCoK is because the aggregator needs to compute a large number of hashes.

Observe that generic SNARK based approaches require orders of magnitude higher aggregation time and are impractical to be used to build threshold signatures for a large number of signers. The dashed entries in the table indicate that we could not run the SNARK prover with the chosen parameters. The Groth16 setup ceremony ran out of memory with 4096 signers. Similarly, the plonk aggregator ran out of memory while aggregating signatures with 1024 signers or higher.

Finally, we measure the aggregation cost of the BLS threshold signature scheme in the weighted setting. Recall that the aggregation cost in the BLS threshold signature depends only on t and not the actual weight distribution. In our evaluation with $t = 2^{15}$ and $t = 2^{16}$, computing the aggregated signature requires 1.15 and 2.27 seconds, respectively. This matches our expectation as the aggregator performs $O(t \log^2 t)$ field operations during aggregation [64]. Also, with 2^{15} and 2^{16} signers, for each signature, the signers will need to send a total of 3 and 6 Megabytes of data, respectively.

Verification time, verification key size, and signature size. We report the unweighted signature verification time and signature size in Table 5. As expected, the BLS threshold signature scheme has the smallest signature size (only one $\mathbb{G}_2$ element) and shortest verification time (only two pairings). Also, as expected, the signature size of the CCoK approach is very large. Note that although CCoK requires the longest verification, the verification is still less than 100 milliseconds. This might be reasonable for many applications. We want to note that despite having an asymptotically linear verification time, the concrete verification time multisignature is very fast. This is because the multisignature only requires a linear number of group multiplications and not group exponentiations.

The only practical downside of a multisignature scheme is the verification key size, i.e., the verifier must store all signers' public keys. The linear verification key size can be prohibitive for applications where a blockchain acts as a verifier, as storing large data on-chain is very expensive (since each node in the blockchain needs to replicate the verification key).

Table 5: Unweighted verification time, signature size, and verification key size

Scheme	Verification time (ms)	Signature size (bytes)	Verification key size (bytes)
BLS Threshold	1.05	96	48
Multisig. (t = 4096)	5.63	608	196608
Groth16	4.6	192	1440
Plonk	5.5	624	1306
CCoK (t = 64)	57.73	27033	64
CCoK (t = 4096)	89.24	206085	64
Our approach	8.21	536	672

Table 6: EVM gas cost using BN254 elliptic curve. $k\mathbb{P}$ refers to the pairing product check with k pairs; $\mathbb{G}_1$, $\mathbb{G}_2$ are the number of group operations in these groups; Exp refers to group exponentiations.

Scheme	# group ops.	Gas cost
Ours implementation	$15\mathbb{P} + 3\mathbb{G}_1 + 15\mathbb{G}_1$ Exp	772k
Multisig est.(n = 4096)	$2\mathbb{P} + (n-1)\mathbb{G}_1 + n\mathbb{G}_1$ storage reads	>23M

Memory usage. Our protocol has low memory usage except for the preprocessing step (cf. §4.5). Note that only the aggregator (a single machine) performs the preprocessing step. During preprocessing, our scheme with 256, 1024, and 4096 signers uses 0.03GB, 0.25GB, and 3.44GB of memory, respectively. The higher memory usage during preprocessing is an implementation choice, as we store vectors of size $O(n^2)$ in the memory. We adopt this approach for the faster preprocessing time. Alternatively, one could implement the preprocessing step with lower memory usage at the cost of a longer running time.

6.3 Verification using Ethereum smart contract

We implemented our threshold signature verifier in Solidity using the BN254 asymmetric pairing curve. We chose BN254 curve as it is natively supported in Ethereum and is the most efficient curve [22, 59]. We want to note that, we make the following changes to the signature scheme to be able to run it efficiently on Ethereum: First, since BN254 is an asymmetric curve, the signature includes commitments to the bit vector in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and we use two additional pairings to check their consistency. Second, we do not implement the verification optimizations in §4.8 since $\mathbb{G}_2$ operations are not efficiently supported in Ethereum. Finally, since a pairing check involving k pairings is more efficient than k independent pairings, we merge all the verification checks into a single pairing by taking an appropriate random linear combination. More precisely, the cost a pairing check with k pairs is $34000k + 45000$ [60].

We report our evaluation results in Table 6. Observe that verifying a signature using our scheme requires 772 thousand gas, while the cost of verifying a multisignature with 4096 signers is over 23 million gas due to reading public keys. Note that the 23M gas does not include the cost of storing public keys on the blockchain, which would require additional 44.8k gas per signer, hence over 183 million gas for all 4096 signers.

7 RELATED WORK

The closest signature scheme to our approach is the standard multisignature scheme. Since we already discuss its properties in detail throughout the paper, we focus on other schemes below.

Threshold Signatures. Threshold signature schemes were first proposed for ElGamal and RSA signatures [29, 30, 42, 46, 62] and later for BLS signatures [11, 13], often utilizing Shamir secret sharing [61]. This approach has many advantages: the signature size, verification key size, and verification time are all constant. Also, many of these schemes produce *unique* threshold signatures, a property that is crucial for threshold signature-based randomness beacons [24]. These standard threshold signatures do not efficiently support arbitrary weight distributions or multiple thresholds.

As mentioned, one approach to support arbitrary weights is *virtualization* of threshold signatures. Here, the signing key is secret-shared using a $(\|\mathbf{w}\|_1, t)$ Shamir secret sharing. Each signer with weight w receives w shares of the secret and signs using all w shares. This approach is inefficient for both the signer and the aggregator. More precisely, an aggregator incurs a computation cost of one $O(\|\mathbf{w}\|_1)$ wide multi-exponentiations and $O(\|\mathbf{w}\|_1 \log^2 \|\mathbf{w}\|_1)$ field operations. In contrast, our aggregator performs one $O(n)$ wide multi-exponentiations and $O(n \log n)$ field operations.

Compact Certificate of Knowledge (CCoK). Micali et al. [55] presents an elegant protocol CCoK to address these issues. CCoK uses a specialized SNARK analogous to Kilian's protocol [48]. CCoK has several nice properties. A signer only needs to sign once, independent of its weight. Their protocol also supports multiple thresholds. The underlying signature scheme is used in a black box manner and hence is compatible with any signature scheme. However, CCoK has several downsides. First, it cannot prove the exact weight of the signers who signed the message. In particular, to prove that a signature is signed by signers with a total weight t, the aggregator needs to collect partial signatures of weight $(1 + \epsilon)t$ for some $\epsilon > 0$. The signature size depends on ϵ. The smaller the ϵ, the larger the signature. As we illustrate in §6, the signature becomes very large even with $\epsilon = 0.25$.

Sampling-based approach. Chaidos and Kiayias present a sampling based weighted threshold signature scheme [26]. The idea is to sample a subset of signers in a verifiable manner based on the weight distribution and then let the sampled signers sign the message. This approach has a few drawbacks. First, it requires a mechanism to securely sample signers proportional to their weights. Second, it increases the costs for signers with large weights. Finally, this approach is typically vulnerable to adaptive corruption.

Generic weighted secret sharing. A generic approach to designing a threshold signature that supports arbitrary weight distribution is to use a weighted secret sharing scheme (WSS), i.e., a secret sharing scheme that inherently considers the weight of each signer. Beimel [6, 7] presented the first characterization of WSS where the share size is sublinear than the weight of the signer. Prior works on WSS has explored other approaches such as Chinese remainder theorem [39, 67], allowing only restricted classes of hierarchical weights [34, 63], and wiretap channels [9]. All these works are theoretical and have very high concrete costs.

Concurrent work. The concurrent and independent work [40] proposed a similar approach and achieved a signature size of $9\mathbb{G}+5\mathbb{F}$ and verification cost of 1 exponentiation and 10 pairings. Similar to our work, they prove the security of their scheme in the AGM. They also discuss mechanisms to extend the signature scheme to support arbitrary access structure and proactive security.

8 CONCLUSION AND OPEN PROBLEMS

We have presented a new threshold signature scheme that supports arbitrary weight distribution and arbitrary thresholds. The signature consists of only 8 group elements. Verifying the signature requires 8 group exponentiations and 8 bilinear pairings. A core component of our scheme is an inner-product argument (IPA) between a vector of group elements and a vector of field elements This part may be of independent interest. For our IPA to work, the discrete logarithms of the group elements must be known in a distributed manner. A potential application could be accountable private threshold signatures [14]. Our threshold signature scheme uses the IPA scheme in a modular way. Thus, any improvement to the IPA scheme immediately results in an improvement in our signature scheme.

Security without AGM. We prove the unforgeability of our threshold signature scheme in the AGM. The reliance on AGM can be removed with some tradeoffs. More precisely, if we use GIPA [20] as our IPA, we can prove our scheme secure in the random oracle model. However, as we discuss in §3.3 it has a considerable performance overhead. Alternatively, we can prove our scheme secure in the standard model but with knowledge assumption for the IPA.

Limitations and Open problems. One drawback of our threshold signature scheme is that the signatures are not unique. This prevents us from using our signature to implement a randomness beacon. Designing a weighted unique threshold signature scheme is a fascinating open problem. Another limitation of our scheme is that the pre-processing cost of the aggregator is quadratic in the number of signers. Other interesting future directions to improve our scheme include reducing the public keys each signer needs to publish and improving the underlying cryptographic assumptions (e.g., removing the need for pairing).

ACKNOWLEDGMENTS

The authors would like to thank Andrew Miller and Lefteris Kokoris-Kogias for their feedback on the paper. This work is funded in part by a VMware early career faculty grant, a Chainlink Labs Ph.D. fellowship, and the National Science Foundation award #2240976.

REFERENCES

[1] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. 2010. Structure-preserving signatures and commitments to group elements. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer, 209–236.

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

[3] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and*

Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Springer, 191–219.

[4] Algorand. 2023. Algorand's official implementation in Go. (2023). https://github.com/algorand/go-algorand

[5] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2023. Threshold Signatures in the Multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[6] Amos Beimel, Tamir Tassa, and Enav Weinreb. 2005. Characterizing ideal weighted threshold secret sharing. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*. Springer, 600–619.

[7] Amos Beimel and Enav Weinreb. 2006. Monotone circuits for monotone weighted threshold functions. *Inform. Process. Lett.* 97, 1 (2006), 12–18.

[8] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. 2019. Aurora: Transparent succinct arguments for R1CS. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 103–128.

[9] Fabrice Benhamouda, Shai Halevi, and Lev Stambler. 2022. Weighted Secret Sharing from Wiretap Channels. *Cryptology ePrint Archive* (2022).

[10] Daniel J Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. 2012. Faster batch forgery identification. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer, 454–473.

[11] Alexandra Boldyreva. 2003. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *Public Key Cryptography*, Vol. 2567. Springer, 31–46.

[12] Dan Boneh and Xavier Boyen. 2004. Short signatures without random oracles. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*. Springer, 56–73.

[13] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II*. Springer, 435–464.

[14] Dan Boneh and Chelsea Komlo. 2022. Threshold signatures with private accountability. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*. Springer, 551–581.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*. Springer, 514–532.

[16] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K Jakobsen. 2017. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 336–365.

[17] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. 2023. ConsenSys/gnark-crypto: v0.9.0. (Jan. 2023). https://doi.org/10.5281/zenodo.5815453

[18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 315–334.

[19] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 677–706.

[20] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. 2021. Proofs for inner pairing products and applications. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*. Springer, 65–97.

[21] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. 2020. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052* (2020).

[22] Vitalik Buterin and Christian Reitwiessner. 2017. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-197. (2017).

[23] Christian Cachin. 2021. Asymmetric distributed trust. In *Proceedings of the 22nd International Conference on Distributed Computing and Networking*. 3–3.

[24] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.

[25] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. 2022. Linear-Map Vector Commitments and Their Practical Applications. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 189–219.

[26] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based threshold multisignatures. *Cryptology ePrint Archive* (2021).

[27] Sourav Das, Zhuolun Xiang, and Ling Ren. 2022. Powers of Tau in Asynchrony. *Cryptology ePrint Archive* (2022).

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

[29] Yvo Desmedt. 1988. Society and group oriented cryptography: A new concept. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*. Springer, 120–127.

[30] Yvo Desmedt. 1993. Threshold cryptosystems. In *Advances in Cryptology—AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings 3*. Springer, 1–14.

[31] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. *Cryptology ePrint Archive* (2022).

[32] Steve Ellis, Ari Juels, and Sergey Nazarov. 2017. Chainlink: A decentralized oracle network. *Retrieved March* 11, 2018 (2017), 1.

[33] Ethereum. 2023. Decentralized autonomous organizations (DAOs). https://ethereum.org/en/dao/. (2023).

[34] Oriol Farras and Carles Padró. 2012. Ideal hierarchical secret sharing schemes. *IEEE transactions on information theory* 58, 5 (2012), 3273–3286.

[35] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[36] Ethereum Foundation. 2020. PROOF-OF-STAKE (POS). https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. (2020).

[37] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The algebraic group model and its applications. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*. Springer, 33–62.

[38] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).

[39] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2023. Cryptography with Weights: MPC, Encryption and Signatures. In *Annual International Cryptology Conference*. Springer, 295–327.

[40] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2024. hinTS: Threshold Signatures with Silent Setup. (2024).

[41] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. In *International conference on financial cryptography and data security*. Springer.

[42] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1996. Robust threshold DSS signatures. In *Advances in Cryptology—EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15*. Springer, 354–371.

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

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

[45] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.

[46] Lein Harn. 1994. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proceedings-Computers and Digital Techniques* 141, 5 (1994), 307–313.

[47] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*. Springer, 177–194.

[48] Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.

[49] Chelsea Komlo and Ian Goldberg. 2021. FROST: flexible round-optimized Schnorr threshold signatures. In *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*. Springer, 34–65.

[50] Jonathan Lee. 2021. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*. Springer, 1–34.

[51] Helger Lipmaa, Janno Siim, and Michał Zając. 2023. Counting vampires: from univariate sumcheck to updatable ZK-SNARK. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*. Springer, 249–278.

[52] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 129–138.

[53] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1041–1053.

[54] Dahlia Malkhi and Michael Reiter. 1998. Byzantine quorum systems. *Distributed computing* 11, 4 (1998), 203–213.

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

[56] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 31–42.

[57] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. 2022. Powers-of-Tau to the People: Decentralizing Setup Ceremonies. *Cryptology ePrint Archive* (2022).

[58] National Institute of Standard and Technology. 2023. Multi-Party Threshold Cryptography. https://csrc.nist.gov/Projects/threshold-cryptography. (2023).

[59] Christian Reitwiessner. 2017. EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. https://eips.ethereum.org/EIPS/eip-196. (2017).

[60] Antonio Salazar Cardozo and Zachary Williamson. 2018. EIP-1108: Reduce alt_bn128 precompile gas costs. https://eips.ethereum.org/EIPS/eip-1108. (2018).

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

[62] Victor Shoup. 2000. Practical threshold signatures. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer, 207–220.

[63] Tamir Tassa. 2007. Hierarchical threshold secret sharing. *Journal of cryptology* 20 (2007), 237–264.

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

[65] Zhuolun Xiang, Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2021. Strengthened fault tolerance in Byzantine fault tolerant replication. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 205–215.

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

[67] Xukai Zou, Fabio Maino, Elisa Bertino, Yan Sui, Kai Wang, and Feng Li. 2011. A new approach to weighted multi-secret sharing. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–6.

A SUCCINCT NON-INTERACTIVE IPA [25]

In this section, we will describe the succinct non-interactive inner product argument (IPA) protocol between a vector of two field elements [25]. The IPA protocol does not rely on a random oracle for non-interactivity and only requires a universal setup.

A.1 Design

Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ be the two input vectors of length n. The prover $\mathcal{P}$ wants to convince a verifier $\mathcal{V}$ that $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \mu$, where $\mathcal{V}$ possesses μ, and commitments to $\boldsymbol{a}$ and $\boldsymbol{b}$.

Setup. For any security parameter κ, let $(\mathbb{G}, \mathbb{G}_T)$ be the description of a bilinear pairing group with scalar field $\mathbb{F}$. Also, let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently computable bilinear pairing map. Let $g, h \in \mathbb{G}$ be two uniformly random generators of $\mathbb{G}$. The IPA protocol assumes the following common reference string (CRS)

$$\left\{ \left[g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^n} \right] ; \left[h, h^{\tau}, h^{\tau^2}, \ldots, h^{\tau^{n-1}} \right] \right\}$$

Here $\tau \in \mathbb{F}$ is the q-SDH trapdoor.

Let $H = \{\omega, \omega^2, \ldots, \omega^n\}$ be a multiplicative subgroup of $\mathbb{F}$ of order n. Here ω is a n-th root of unity of $\mathbb{F}$.

Proof generation. Let $a(\cdot)$ and $b(\cdot)$ be polynomials of degree $n-1$ such that $a(\omega^i) = \boldsymbol{a}[i]$ and $b(\omega^i) = \boldsymbol{b}[i]$ for all $i \in [n]$. Then,

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{i \in [n]} a(\omega^i) b(\omega^i)$$

Let $z_H(x)$ be the vanishing polynomial over H, i.e.,

$$z_H(x) = \prod_{i \in [n]} (x - \omega^i) = x^n - 1$$

The IPA scheme uses the following sumcheck Lemma [8] of univariate polynomials.

$$a(x)b(x) = q(x)z_H(x) + r(x)x + n^{-1}\langle \boldsymbol{a}, \boldsymbol{b} \rangle$$

Here, both $q(x)$ and $r(x)$ are unique polynomials of degree $n-2$. Let $p(x) = r(x)x + n^{-1}\langle \boldsymbol{a}, \boldsymbol{b} \rangle$.

The IPA π for $\mu = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is the tuple is

$$\pi = \left\{ g^{q(\tau)}, g^{r(\tau)}, h^{p(\tau)} \right\} \tag{41}$$

Proof verification. We use KZG commitments to polynomials $a(x)$ and $b(x)$, i.e., $(g_a, g_b) = (g^{a(\tau)}, g^{b(\tau)})$ as the commitments to the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively.

$\mathcal{V}$ upon receiving the proof $\pi = (g_q, g_r, h_p)$, accepts μ as the inner product, if following checks pass

$$e(g_a, g_b) = e(g_q, g^{z_H(\tau)}) \cdot e\left(g_r, g^{\tau}\right) \cdot e\left(g^{\mu}, g^{1/n}\right); \text{ and} \tag{42}$$

$$e(h_p, g) = e\left(g_r, h^{\tau}\right) \cdot e\left(g^{\mu}, h^{1/n}\right) \tag{43}$$

Analysis. The completeness of the protocol clear. The proof consists of 3 $\mathbb{G}$ elements. Also, assuming $g^{z_H(\tau)}, g^{1/n}, h^{1/n}$ are part of the CRS, verification requires one exponentiation and 7 pairings. In terms of provers computation cost, $\mathcal{P}$ computes the polynomials $q(x)$ and $r(x)$ in $O(n \log n)$ field operations using number theoretic transform. Then, $\mathcal{P}$ computes $(g^{q(\tau)}, g^{r(\tau)}, h^{p(\tau)})$ using $O(n)$ group exponentiations. The knowledge soundness follows from [25, Theorem 5.2].

B EXTENSIONS AND APPLICATIONS

B.1 Subvector accountable

A signature scheme is accountable if any signature σ on a message m reveals the identity of the signers whose signatures are aggregated to compute σ. Moreover, it is not feasible for a quorum of signers to frame another quorum. For any given n, since there are 2^n possible quorum of signers, any accountable threshold signature must be n bits long. Hence, there is an inherent trade-off between accountability and succinctness. The classic example of this trade-off is the existing threshold signature and multisignatures. The signatures in a threshold signature are succinct but not accountable, whereas signatures in multisignature schemes are accountable but are n bits long.

Note that the existing threshold signatures and multisignatures achieve an all-or-nothing property. Contrary to these schemes, our scheme achieves a more fine-grained trade-off between accountability and succinctness, which we define as the subvector accountable.

Definition B.1 (Subvector Accoutable). For any subset of signers $I \subseteq [n]$ of size ℓ, $\mathcal{P}$ can produce a $\ell + \kappa$ bits long proof that convinces $\mathcal{V}$ on the accountability of the signers in I.

Our signature scheme achieves subvector accountable as follows. Recall from §4.8, the aggregate σ includes the KZG polynomial commitment to the polynomial $b(x)$. Also, let $\boldsymbol{b}_I$ be the subvector of $\boldsymbol{b}$ for the subset I, i.e., $\boldsymbol{b}_I = [b(\omega^i)]_{i \in I}$. Then, the subvector accountable proof consists of $\boldsymbol{b}_I$, and the batched KZG evaluation proof [47], with respect to the commitment to polynomial $b(x)$. Note that the proof consists of a single group element.

Remark. We do not find any agreed-upon distinctions between multisignature and threshold signature in the literature. But the main differences are that multisignature schemes are accountable and support multiple thresholds. If one views multisignatures as threshold signatures with accountability and multiple thresholds (and no other requirements), then our scheme with subvector accountability is also a new succinct multisignature scheme.

B.2 Signer Inclusion

Our signature scheme lets $\mathcal{P}$ efficiently compute a succinct proof that a signer is included in the set of signers, a property we refer to as the *signer inclusion*. The signer inclusion property, along with subvector accountability defined in the previous section, can be used in the context of PoS blockchains to slash a participant who signs conflicting proposals.

Definition B.2 (Signer Inclusion). For any signer with public key pk included in the set of signers with a public verification key vk, $\mathcal{P}$ can produce a succinct proof of its inclusion in vk.

Let i be the index of the signer for which $\mathcal{P}$ wants to compute the inclusion proof. Recall from §4.8, vk consists of $g^{s(\tau)}, g^{\tau}$, and ak consists of $g^{q_i(\tau)}$. For any index $i \in [n]$, using the polynomial remainder lemma (Lemma 3.2), we can write $s(x)$

$$s(x) = \hat{q}_i(x)(x - \omega^i) + s(\omega^i) \tag{44}$$

Moreover, from equation (22) we get that

$$\hat{q}_i(x) = q_i(x) \cdot c_i; \quad \text{for} \quad c_i = \prod_{j \in [n]; j \neq i} (\omega^i - \omega^j) \tag{45}$$

The inclusion proof is $\pi = (i, g^{s_i}, g^{q_i(x)c_i}, v^{s_i})$. $\mathcal{V}$ upon receiving the proof $\pi = (i, g_{s_i}, g_{q_i}, v_{s_i})$ checks that:

$$e(g_s/g_{s_i}, g) = e(g_{q_i}, g^{\tau}/g^{\omega^i}); \quad \text{and} \quad e(v_{s_i}, g) = e(g_{s_i}, v) \tag{46}$$

The soundness relies on a similar argument as in Theorem 5.7. Briefly, the latter check guarantees that g_{s_i} is a commitment to a constant element. The former check guarantees that assuming the hardness of q-SDH, g_{s_i} is binding. Also, given ak, $\mathcal{P}$ can compute the proof π using only $O(n)$ field operations and 1 group exponentiation.

B.3 Improved Multiverse Threshold Signature

Our signature scheme immediately improves the setup required for the multiverse threshold signatures (MTS) scheme proposed in [5]. Baird et al. in [5] introduce the notion of multiverse threshold signature, where they refer to a set of signers as a *universe*, and multiple "universes" of signers to co-exist. Each universe has its own threshold, and a signer can be simultaneously part of different universes. Intuitively, MTS lets a client decide on which subset of signers it trusts and also picks the corresponding threshold. We refer the reader to [5] for the motivation behind defining MTS and the formal definition.

The MTS protocol of [5] has several nice properties: (i) The signing key of each signer is a single field element, and the signer needs to sign only once, independent of the number of universes it is part of. (ii) The per universe aggregate signature consists of only 3 group elements. However, there are also some disadvantages: (i) For each universe, the signers in the universe need to run an interactive setup protocol to generate the public verification keys. (ii) They do not support multiple thresholds among a single subset of signers. Instead, they create a separate universe for each threshold, requiring an independent setup for each threshold. (iii) Their scheme does not efficiently support weighted signers. Instead, they adopt an approach similar to virtualization, where the signing keys of the signers are generated using a pseudorandom function.

We address the abovementioned issues while maintaining its benefits by using our threshold signature scheme as the per universe signature scheme in MTS. As a result, each universe can efficiently support multiple thresholds and arbitrary weights and only needs PKI and a universal powers-of-tau setup. Moreover, the powers-of-tau setup is reusable across different universes with the same number of signers. We believe that the security analysis of [5] can be easily extended to use our threshold signature scheme. We leave the detailed security analysis as a future work.

C ADDITIONAL PRELIMINARIES

C.1 Bilinear Pairing and Assumptions

Definition C.1 (Bilinear Pairing). Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three prime order cyclic groups with scalar field $\mathbb{F}$. Let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be the generators. A pairing is an efficiently computable function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following properties.

(1) bilinear: For all $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_1$ we have

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v), \text{ and}$$
$$e(u, v \cdot v') = e(u, v) \cdot e(u, v')$$

(2) non-degenerate: $g_T := e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.

We refer to $\mathbb{G}_1$ and $\mathbb{G}_2$ as the pairing groups or source groups, and refer to $\mathbb{G}_T$ as the target group.

Assumption 1 (co-CDH)**.** A pairing group $(\mathbb{G}_1, \mathbb{G}_2)$ with generator (g_1, g_2) and a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies the co-CDH assumption if, for all PPT adversary $\mathcal{A}$, it holds that

$$\Pr[\mathcal{A}(g_1, g_2, g_1^s, g_2^s, g_2^r) = g_2^{sr} : s, r \leftarrow \mathbb{F}] = \mathsf{negl}(\kappa) \tag{47}$$

Assumption 2 (q-Strong Diffie-Hellman [12])**.** The q-Strong Diffie-Hellman (q-SDH) assumption states that for every efficient adversary $\mathcal{A}$ and degree bound $n \in \mathbb{N}$, the following probability is negligible in the security parameter κ:

$$\Pr\left[g_c = g^{1/(\tau+c)} \;\middle|\; \begin{array}{r} (g, \mathbb{F}, \mathbb{G}) = \mathsf{Setup}(1^{\kappa}) \\ \tau \leftarrow \mathbb{F} \\ pp \leftarrow \left\{ g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^n} \right\} \\ (c, g_c) \leftarrow \mathcal{A}((g, \mathbb{F}, \mathbb{G}), pp) \end{array} \right]$$

C.2 Algebraic Group Model (AGM)

We prove security of our protocol in the Algebraic Group Model [37]. In the AGM, all algorithms are modeled as algebraic, i.e., whenever the algorithm outputs a group element h, the algorithm also outputs an *representation* of h with respect to all the group elements it has seen so far.

Definition C.2 (Algebraic Algorithm). Let $\mathbb{G}$ be a prime order cyclic group with scalar field $\mathbb{F}$. Let $\mathcal{A}_{\text{Alg}}$ be a probabilistic algorithm run on initial inputs including the description $(\mathbb{F}, \mathbb{G})$. During its execution $\mathcal{A}_{\text{Alg}}$ receive further inputs including obliviously sampled group elements (which it can not sample directly). Let $\boldsymbol{g} \in \mathbb{G}^n$ be the list of all group elements $\mathcal{A}_{\text{Alg}}$ has been given so far such that any other inputs $\mathcal{A}_{\text{Alg}}$ has received do not depend on the input. We call $\mathcal{A}_{\text{Alg}}$ algebraic, if whenever $\mathcal{A}_{\text{Alg}}$ outputs a new group element $h \in \mathbb{G}$, it also outputs a vector $\boldsymbol{a} = [a_1, a_2, \ldots, a_n] \in \mathbb{F}^n$ such that $h = \prod_{i \in [n]} g_i^{a_i}$. The coefficients $\boldsymbol{a}$ are called the *representation* of h with respect to $h = \langle \boldsymbol{a}, \boldsymbol{g} \rangle$.

D POLYNOMIAL IDENTITIES DERIVATION

Our construction uses the following lemma from [31].

LEMMA D.1. *Let $s(x)$ and $b(x)$ be two polynomials of degree $n-1$ each over the field $\mathbb{F}$ such that $s(\omega^i) = s_i$ and $b(\omega^i) = b_i$. Also, let $q(x)$ and $p(x)$ are the unique quotient and remainder polynomials of degree $n-2$ each defined as follows:*

$$s(x)b(x) = q(x)z_H(x) + p(x) \tag{48}$$

here $z_H(x)$ is the polynomial that evaluates to 0 at every $x \in H = \{\omega, \omega^2, \ldots, \omega^n\}$. Then the following holds,

$$q(x) = \sum_{i \in [n]} q_i(x) b_i; \quad \text{and} \quad p(x) = \sum_{i \in [n]} b_i s_i \mathcal{L}_i(x)$$

where $\mathcal{L}_i(x)$ is the i-th Lagrange polynomial defined on H and $q_i(x)$ are defined as

$$\mathcal{L}_i(x)s(x) = s_i\mathcal{L}_i(x) + z_H(x)q_i(x) \tag{49}$$

PROOF. Note that by definition, we can write $b(x)$ as

$$\begin{aligned} b(x) &= \sum_{i \in [n]} b_i \mathcal{L}_i(x) \\ \Rightarrow s(x)b(x) &= \sum_{i \in [n]} b_i \mathcal{L}_i(x) s(x) \end{aligned} \tag{50}$$

Next, by substituting equation (49) in equation (50), we get,

$$\begin{aligned} s(x)b(x) &= \sum_{i \in [n]} b_i \mathcal{L}_i(x) s(x) \\ &= \sum_{i \in [n]} b_i \left(s_i \mathcal{L}_i(x) + z_H(x) q_i(x) \right) \\ &= \sum_{i \in [n]} b_i s_i \mathcal{L}_i(x) + \sum_{i \in [n]} b_i q_i(x) z_H(x) \end{aligned} \tag{51}$$

Since $q(x)$ and $r(x)$ are the unique quotient and remainder polynomials, we get that the first term of equation (51), $\sum_{i \in [n]} b_i s_i \mathcal{L}_i(x) = p(x)$, and the second term $\sum_{i \in [n]} b_i q_i(x) = q(x)$. □

E PROOFS

E.1 Proofs of Knowledge soundness of the IPA

PROOF OF THEOREM 5.1. For the sake of contradiction, let's assume that $\mathcal{A}$ outputs a non-zero polynomial $a(\cdot)$ of degree $\leq m$ such that $a(\tau) = 0$. Without loss of generality let's assume that $\mathcal{A}$ outputs $a(\cdot)$ in the coefficient representation, i.e., let $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$ be the polynomial ouput by $\mathcal{A}$.

Now consider two exclusive possibilities: (i) $a_0 \neq 0$ and (ii) $a_0 = 0$. In the former case, we can write

$$\tau(a_m\tau^{m-1} + \cdots + a_1\tau) = -a_0 \Rightarrow \frac{a_m\tau^{m-1} + \cdots + a_1}{-a_0} = \frac{1}{\tau}$$

Since $\mathcal{A}$ knows all the coefficients $a_0, a_1, \ldots, a_m$ and the q-SDH parameters consists of all powers up to degree $m-1$, $\mathcal{A}$ can efficiently compute $g^{1/\tau}$ and output the q-SDH tuple $(0, g^{1/\tau})$. Next, for the latter case where $a_0 = 0$, since $a(x)$ is a non-zero polynomial, we can rewrite $a(x)$ to be $a(x) = x^k \cdot a'(x)$ for some $0 < k < m$ such that $a'(x)$ has a non-zero constant term a'_0. Then, $\mathcal{A}$ can follow the same approach as the former case with polynomial $a'(x)$ instead of $a(x)$ to output a q-SDH tuple. □

PROOF OF LEMMA 5.2. For any m, let $\boldsymbol{g}_m = [g, g^\tau, g^{\tau^2}, \ldots, g^{\tau^m}]$. $\mathcal{A}_{q\text{SDH}}$ on input the q-SDH parameters $\boldsymbol{g}_{2n}$, emulates $\mathcal{A}_{\text{IPA}}$ for n, i.e., half the size of the q-SDH parameters. Looking ahead, $\mathcal{A}_{q\text{SDH}}$ uses the remaining powers of tau to output the q-SDH tuple. $\mathcal{A}_{q\text{SDH}}$ sends $\boldsymbol{g}_n$ to $\mathcal{A}_{\text{IPA}}$ and also uses $\boldsymbol{g}_n$ to generate the public parameters pp of $\mathcal{R}_{\text{TS}}$. Note that the pp consists of evaluations of degree at most $n-1$ polynomials at τ in the exponent. Thus $\mathcal{A}_{q\text{SDH}}$ can efficiently compute them using only $\boldsymbol{g}_n$.

Let $\mathcal{H}$ be the set of honest signers and $\mathcal{M} = [n] \setminus \mathcal{H}$ be the set of malicious signers. $\mathcal{A}_{q\text{SDH}}$ samples signing keys for all signers in $\mathcal{H}$, computes the corresponding public keys and proof of possessions. $\mathcal{A}_{q\text{SDH}}$ then sends the public keys and the proofs to $\mathcal{A}_{\text{IPA}}$. $\mathcal{A}_{\text{IPA}}$ then responds back with the public keys of the malicious signers, i.e., signers in $\mathcal{M}$ and their corresponding proof of possession. Since $\mathcal{A}_{\text{IPA}}$ is algebraic and outputs a proof-of-possession, due to reasons similar to Lemma 5.3, for each malicious public key $\mathcal{A}_{\text{IPA}}$ outputs the corresponding signing key to $\mathcal{A}_{q\text{SDH}}$.

Let (g_b, g_q) be the claimed commitment of the bit vector and its correctness proof output by $\mathcal{A}_{\text{IPA}}$, respectively. Since $\mathcal{A}_{\text{IPA}}$ is algebraic, it additionally outputs the tuple of vectors $(\hat{\boldsymbol{b}}, \hat{\boldsymbol{q}})$ such that $(g_b, g_q) = (\langle \boldsymbol{g}_n, \hat{\boldsymbol{b}} \rangle, \langle \boldsymbol{g}_n, \hat{\boldsymbol{q}} \rangle)$. Given g_b and g_q, a successful validation check implies that:

$$e\left(g_b, g/g_b\right) = e\left(g_q, g^{z_H(\tau)}\right) \tag{52}$$

Let $\hat{b}(x)$ and $\hat{q}(x)$ be the polynomials whose coefficients are defined by vectors $\hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{q}}$, respectively. Note that $\hat{\boldsymbol{b}} \in \mathbb{F}^{n+1}$, hence $\hat{b}(x)$ is a polynomial of degree at most n, i.e., 1 greater than the polynomial an honest prover commits to. Nevertheless, for the sake of contradiction, let us assume that $\hat{b}(\omega^i) \notin \{0, 1\}$ for any $i \in [n]$. Then, using the polynomial remainder theorem, the following holds

$$\hat{b}(x)(1 - \hat{b}(x)) = q(x)z_H(x) + p(x) \tag{53}$$

Here, $q(x)$ and $p(x)$ are the quotient and remainder polynomials of degree at most n and $n-1$, respectively. Also, our assumption of $\hat{b}(\omega^i) \notin \{0, 1\}$, implies that $p(x)$ is a non-zero polynomial. Because, $\hat{b}(\omega^i)(1 - \hat{b}(\omega^i) = p(\omega^i) \neq 0$.

Let $\Delta(x) = (q(x) - \hat{q}(x))z_H(x) + p(x)$. Then, the successful check in equation (52) and equation (53) implies that $\Delta(\tau) = 0$, i.e.,

$$\Delta(\tau) = (q(\tau) - \hat{q}(\tau))z_H(\tau) + p(\tau) = 0 \tag{54}$$

By definition, the degree of $\Delta(x)$ is at most $2n$. We will now prove that $\Delta(x)$ is a non-zero polynomial. By definition $p(x)$ is a polynomial of degree at most $n-1$ and $z_H(x) = x^n - 1$. Now, consider two exclusive possibilities: (i) $q(x)-\hat{q}(x)$ is a zero polynomial, and (ii) $q(x)-\hat{q}(x)$ is a non-zero polynomial. In the former, $\Delta(x)$ is trivially non-zero. In the latter, when $q(x)-\hat{q}(x)$ is a polynomial of degree k, then $(q(x)-\hat{q}(x))z_H(x)$ will be a polynomial of degree $n+k$ where the coefficient of the monomial x^{k+n} is non-zero.

Combining all the above we get that if $\hat{b}(\omega^i) \notin \{0,1\}$ for any $i \in [n]$, then $\Delta(x)$ is a non-zero polynomial of degree at most $2n$ such that $\Delta(\tau) = 0$. Hence, using Theorem 5.1, this implies that q-SDH is easy. Thus, assuming hardness of q-SDH $\hat{b}(\omega^i) \in \{0,1\}$ for each $i \in [n]$ and the extractor $\mathcal{E}$ outputs $\boldsymbol{b} = [\hat{b}(\omega), \hat{b}(\omega^2), \ldots, \hat{b}(\omega^n)]$. □

Proof of Lemma 5.3. Let $[g, g^\rho, \ldots, g^{\rho^{2n}}]$ be the input to the q-SDH adversary $\mathcal{A}_{q\mathrm{SDH}}$. Next, with probability 1/2 $\mathcal{A}$ uses $g^{\tau^i} = g^{\rho^i}$ for each $i \in [n]$. Alternatively, with probability 1/2, $\mathcal{A}_{q\mathrm{SDH}}$ uses $v = g^\beta = g^\rho$. $\mathcal{A}_{q\mathrm{SDH}}$ locally samples the trapdoors of the remaining CRS of the threshold signature scheme.

During the threshold signature protocol, since $\mathcal{A}_{\mathrm{IPA}}$ is algebraic, we can write $\mu(\tau)$ and $\hat{\mu}(\tau)$ as:

$$\mu(\tau) = z\beta + \sum_{i=0}^{n} \mu_i \tau^i \ ; \quad \hat{\mu}(\tau) = \hat{z}\beta + \sum_{i=0}^{n} \hat{\mu}_i \tau^i$$

Let $\Delta(x) = \beta\mu(x) - \hat{\mu}(x)$. Then, a successful verification check in equation (16) implies $\Delta(\tau) = 0$, i.e.,

$$\beta^2 z + \beta\left(\sum_{i=0}^{n} \mu_i \tau^i - \hat{z}\right) + \sum_{i=0}^{n} \hat{\mu}_i \tau^i = 0 \tag{55}$$

We can view equation (55) as a quadratic polynomial in β. If either of the coefficients of β^2 or β is non-zero, then, we can use equation (55) to solve discrete logarithm for g^β, as per [37, Theorem 3.1]. Otherwise, if both of these coefficients are zero, the constant term is also zero, and hence using Theorem 5.1 q-SDH is easy. This implies that $\forall i \in [n],\ \hat{\mu}_i = \mu_i = 0$, and $\mu(x)$ is a constant polynomial. □

Proof of Lemma 5.4. The proof follows a similar approach to that of Lemma 5.3. Let $[g, g^\rho, \ldots, g^{\rho^{2n}}]$ be the input to the q-SDH adversary $\mathcal{A}_{q\mathrm{SDH}}$. Next, with probability 1/2 $\mathcal{A}$ uses $g^{\tau^i} = g^{\rho^i}$ for each $i \in [n]$ and with probability 1/2, $\mathcal{A}_{q\mathrm{SDH}}$ uses $h = g^\alpha = g^\rho$. $\mathcal{A}_{q\mathrm{SDH}}$ locally samples the trapdoors of the remaining CRS of the threshold signature scheme.

Since $\mathcal{A}_{\mathrm{IPA}}$ is algebraic, we can write $r(\tau)$ and $p(\tau)$ as:

$$r(\tau) = \sum_{i=0}^{n} r_i \tau^i + \alpha\left(\sum_{i=0}^{n-1} \hat{r}_i \tau^i\right); \qquad p(\tau) = \sum_{i=0}^{n} p_i \tau^i + \alpha\left(\sum_{i=0}^{n-1} \hat{p}_i \tau^i\right)$$

Let $\mu' = (\mu + \xi t/n)$. Let $\Delta(x)$ be the polynomial defined as:

$$\Delta(x) = \alpha(xr(x) + \mu') - p(x)$$

Then, a successful verification check in equation (40) implies that $\Delta(\tau) = 0$. Using definition of $r(x)$ and $p(x)$, we can write $\Delta(\tau)$ as:

$$\alpha^2\left(\sum_{i=0}^{n-1} \hat{r}_i \tau^{i+1}\right) + \alpha\left(\sum_{i=0}^{n} r_i \tau^{i+1} + \mu' - \sum_{i=0}^{n-1} \hat{p}_i \tau^i\right) + \sum_{i=0}^{n} p_i \tau^i = 0 \tag{56}$$

Similar to proof of Lemma 5.3, we can view equation (56) as a quadratic equation in α. If either the coefficients of α^2 or α is non-zero, we can use equation (56) to solve the discrete logarithm for g^α. If both of these coefficients are zero, the constant term $\sum_i p_i \tau^i$ is also zero. Then, we can use Theorem 5.1 to break the q-SDH assumption. This implies $\hat{r}_{n-1} = r_{n-1} = r_n = 0$, i.e., $r(x)$ is a polynomial of degree at most $n-2$. □

Proof of Lemma 5.5. Our proof uses the standard Schwartz-Zippel lemma. Let $\hat{\mu} = \langle \boldsymbol{s}, \boldsymbol{b} \rangle$ and $\hat{t} = \langle \boldsymbol{w}, \boldsymbol{b} \rangle$ be the correct inner products. Also, let μ be such that $g_\mu = g^\mu$. Then, correctness of g_ν implies that:

$$\hat{\mu} + \xi\hat{t} = \mu + \xi t \Rightarrow (\hat{\mu} - \mu) + \xi(\hat{t} - t) = 0 \tag{57}$$

Let $\Delta(x) = (\hat{\mu} - \mu) + x(\hat{t} - t) = 0$ be the polynomial in the variable x. Then, equation (57) implies that $\Delta(\xi) = 0$ for an uniformly random $\xi \in \mathbb{F}$. This implies, using the Schwartz-Zippel lemma, $\Delta(x)$ is identically zero with probability $1 - 1/|\mathbb{F}|$, i.e., $\mu = \hat{\mu}$ and $t = \hat{t}$. □

Proof of Lemma 5.6. Let $\boldsymbol{g}_n$ be the input to the adversary $\mathcal{A}_{\mathrm{IPA}}$. Also, let $o(x)$ and $b(x)$ be the polynomials defined using the elements of vector $\boldsymbol{o}$ and $\boldsymbol{b}$ as coefficients, respectively. Then the tuple (g_q, g_r) in the IPA proof satisfies the following check.

$$e\left(g^{b(\tau)}, g^{o(\tau)}\right) = e\left(g_q, g^{z_H(\tau)}\right) \cdot e\left(g_r, g^\tau\right) \cdot e\left(g_\nu, g\right) \tag{58}$$

Let $\hat{q}(x)$ and $\hat{r}(x)$ are the polynomials defined as:

$$b(x)o(x) = \hat{q}(x)z_H(x) + x\hat{r}(x) + \hat{\nu} \tag{59}$$

Since, $\mathcal{A}_{\mathrm{IPA}}$ is algebraic, it outputs vectors $\boldsymbol{q}, \boldsymbol{r} \in \mathbb{F}^{n+1}$ such that $(g_q, g_r) = (\langle \boldsymbol{g}_n, \boldsymbol{q} \rangle, \langle \boldsymbol{g}_n, \boldsymbol{r} \rangle)$. Let $q(x), r(x)$ be two polynomials of degree n each defined using the vectors $\boldsymbol{q}$ and $\boldsymbol{r}$ as their coefficients, respectively. Let ν be such that $g_\nu = g^\nu$. Note that such ν exists due to Lemma 5.3. Let $\Delta(x)$ be the polynomial defined as

$$\Delta(x) = (\hat{q}(x) - q(x))z_H(x) + x(\hat{r}(x) - r(x)) + (\hat{\nu} - \nu) \tag{60}$$

Then, a successful verification implies that $\Delta(\tau) = 0$), i.e.,

$$(\hat{q}(\tau) - q(\tau))z_H(\tau) + \tau(\hat{r}(\tau) - r(\tau)) + (\hat{\nu} - \nu) = 0 \tag{61}$$

We now argue that if $\hat{\nu} \neq \nu$, then $\Delta(x)$ is a non-zero polynomial and q-SDH is easy. Note that $x(\hat{r}(x) - r(x))$ does not have a constant term. Hence, the for $\Delta(x)$ to be an identically zero-polynomial, $(\hat{q}(x) - q(x))z_H(x)$ must have a constant term equal to $\nu - \hat{\nu}$. This implies that $\hat{q}(x) - q(x)$ must be non-zero, and hence $(\hat{q}(x) - q(x))z_H(x)$ is a polynomial of degree at least n. But, by definition, $\hat{r}(x)$ is a polynomial of degree $n-2$. Similarly, from Lemma 5.4, $r(x)$ is also a polynomial of degree at most $n-2$. This implies that $\Delta(x)$ is a non-zero polynomial with $\Delta(\tau) = 0$, and q-SDH is easy. This concludes that assuming hardness of q-SDH, $\hat{\nu} = \nu$ and g_ν is the correct inner-product $g^{\langle \boldsymbol{b}, \boldsymbol{o} \rangle}$.

Note that $\Delta(x)$ is of degree at most $2n$. Thus, in our reduction, we start with a q-SDH tuple with $\boldsymbol{g}_{2n}$ only sends the first $\boldsymbol{g}_n$ to $\mathcal{A}_{\mathrm{IPA}}$. $\mathcal{A}_{q\mathrm{SDH}}$ uses the remaining n elements as per Theorem 5.1. □

E.2 Security of threshold signature scheme

We prove the security of our signature scheme in the presence of an *adaptive* adversary. In particular, we prove that if an adversary $\mathcal{A}_{\mathrm{TS}}$ produces a non-trivial forgery of our threshold signature scheme, then we use $\mathcal{A}_{\mathrm{TS}}$ in a black-box manner to design an adversary $\mathcal{A}_{\mathrm{coCDH}}$ that breaks the co-CDH assumption. Recall from §3.1, a forgery is non-trivial when $\mathcal{A}_{\mathrm{TS}}$ produces a signature with threshold t, while querying partial signatures from honest signers of weight less than $t - w_F$. Here w_F is the weight of the corrupt signers. Trivially, if $w_F = \|\boldsymbol{w}\|_1$, i.e., $\mathcal{A}_{\mathrm{TS}}$ corrupts all signers, then the signature scheme is trivially secure. Thus, we focus on $w_F < \|\boldsymbol{w}\|_1$, i.e., at least one honest signer exists.

We describe $\mathcal{A}_{\text{coCDH}}$ using an asymmetric pairing group, which is more general than the symmetric group. A similar construction works for symmetric pairing groups as well. $\mathcal{A}_{\text{coCDH}}$ upon receiving a co-CDH tuple $(g_1, g_2, g_1^s, g_2^s, g_2^r)$ proceeds as follows. $\mathcal{A}_{\text{coCDH}}$ samples $\alpha, \beta \in \mathbb{F}$ and sets $h = g^\alpha, v = g^\beta$. Next, $\mathcal{A}_{\text{coCDH}}$ samples $\tau \in \mathbb{F}$ and computes the public parameters of our signature using the $\mathsf{Setup}(1^\kappa, n)$ algorithm.

Setup, key generation and proofs-of-posession. Recall, F_0 is the initial set of signers $\mathcal{A}$ corrupts. Let $\mathcal{H} = [n] \setminus F_0$ be the initial set of honest signers. $\mathcal{A}_{\text{coCDH}}$ samples a random signer index $\hat{i} \in \mathcal{H}$ and uses g_1^s as the public key of signer $\hat{i}$, i.e., $pk_{\hat{i}} = g^s$. For remaining honest signers $\mathcal{A}_{\text{coCDH}}$ samples the signing keys uniformly at random. Let s_i be the signing key of signer i. $\mathcal{A}_{\text{coCDH}}$ then computes proof-of-possession for all honest signers except $\hat{i}$ as per the honest protocol. For signer $\hat{i}$, it samples a uniform random $a \in \mathbb{F}$ and sets $\mathsf{H}_{\text{pop}}(g_1^s) = g_2^a$, and outputs g_2^{sa} as the proof-of-possession. $\mathcal{A}_{\text{coCDH}}$ then sends public keys of all the signers in $\mathcal{H}$ along with its proof-of-possession to $\mathcal{A}_{\text{TS}}$.

Also, whenever $\mathcal{A}_{\text{TS}}$ queries $\mathsf{H}_{\text{pop}}(\cdot)$ on input y, $\mathcal{A}_{\text{coCDH}}$ samples a uniformly random $a_y \in \mathbb{F}$, assigns $\mathsf{H}_{\text{pop}}(y) = g_2^{ra_y}$, and adds (y, a_y) to a list L_1.

Emulating signing and corruption oracles. $\mathcal{A}_{\text{coCDH}}$ then emulates the corruption oracle $\mathcal{O}_{\text{cur}}(\cdot)$ as follows. If $\mathcal{A}_{\text{TS}}$ corrupts $\hat{i}$, then $\mathcal{A}_{\text{coCDH}}$ aborts. Otherwise, upon corrupting signer $i \neq \hat{i}$, $\mathcal{A}_{\text{coCDH}}$ responds with the signing key s_i.

Similarly, $\mathcal{A}_{\text{coCDH}}$ emulates the signing oracle $\mathcal{O}_{\text{sign}}(\cdot, \cdot)$ as follows. Let q_H be the upper bound on the number of signing queries. $\mathcal{A}_{\text{coCDH}}$ samples a random $k \in [q_H]$. For the k-th random oracle query to $\mathsf{H}(m)$, $\mathcal{A}_{\text{coCDH}}$ outputs $\mathsf{H}(m) = g_2^r$ and adds $(m, \perp, 1)$ to a list L_0. Otherwise, $\mathcal{A}_{\text{coCDH}}$ chooses a random $a_m \in \mathbb{F}$ and assigns $\mathsf{H}(m) = g_2^{a_m}$. Also, it adds $(m, a_m, 0)$ to the list L_0.

$\mathcal{A}_{\text{coCDH}}$ emulates the signing procedure as follows. If $\mathcal{A}_{\text{TS}}$ queries partial signature from signer $\hat{i}$ on the message m_k, $\mathcal{A}_{\text{coCDH}}$ aborts. Otherwise, it uses its knowledge of the signing keys of the remaining signers and the list L_0 to compute the honest partial signature.

Breaking co-CDH. Let $\mathcal{A}_{\text{TS}}$ outputs a non-trivial forgery $\sigma = (\cdots, g_\mu, \sigma_{\text{BLS}}, t, \cdots)$ for message m^*. If $\mathsf{H}(m^*)$ was not the k-th random oracle query, then $\mathcal{A}_{\text{coCDH}}$ aborts. Otherwise, using knowledge soundness of the protocol for $\mathcal{R}_{\text{TS}}$, we can extract a bit vector $\boldsymbol{b}$ such that $\langle \boldsymbol{pk}, \boldsymbol{b} \rangle = g_\mu$ and $\langle \boldsymbol{w}, \boldsymbol{b} \rangle = t$. Let I be the set of indices such that $\boldsymbol{b}[i] = 1$, $\forall i \in I$, and 0 otherwise.

Note that the non-triviality of the forgery implies that I includes at least one honest signer, who $\mathcal{A}_{\text{TS}}$ did not query for partial signature on the message m^*. If $\hat{i} \notin I$, then $\mathcal{A}_{\text{coCDH}}$ aborts. Otherwise, $\mathcal{A}_{\text{coCDH}}$ proceeds as follows.

For each $i \in I \backslash \{\hat{i}\}$, $\mathcal{A}_{\text{coCDH}}$ looks up $(pk_i = y_i, a_i)$ in L_1. Then, the proof-of-possession of signer i is $\pi_i = (g_2^r)^{a_i s_i}$. Here s_i is the discrete logarithm of y_i with respect to g_1, i.e., $y_i = g_1^{s_i}$. Note that a succesful signature verification implies that $\sigma_{\text{BLS}} = g_2^{rs} \cdot \prod_{i \in I \backslash \{\hat{i}\}} g_2^{rs_i}$. Thus, $\mathcal{A}_{\text{coCDH}}$ then outputs the co-CDH break as:

$$\sigma_{\text{BLS}} \cdot \prod_{i \in I \backslash \{\hat{i}\}} \pi_i^{-a_i^{-1}} = \sigma_{\text{BLS}} \cdot \prod_{i \in I \backslash \{\hat{i}\}} \left(g_2^{rs_i a_i} \right)^{-a_i^{-1}} = g_2^{rs}$$

Success probability. Let ϵ be the probability with which $\mathcal{A}_{\text{TS}}$ outputs a forgery. It is easy to see that if $\mathcal{A}_{\text{coCDH}}$ does not abort, then it outputs a correct co-CDH break. Thus, we first lower-bound the probability that $\mathcal{A}_{\text{coCDH}}$ does not abort.

First, since $\hat{i}$ is chosen uniformly at random, $\mathcal{A}_{\text{TS}}$ does not corrupt $\hat{i}$ with probability at least $1/n$. Next, m^* is the k-th random oracle query with probability at least $1/q_{\text{H}}$. Also, let $\delta_{\mathcal{R}_{\text{TS}}}$ be the probability that the extractor $\mathcal{E}$ outputs a bit vector $\boldsymbol{b}$ that satisfies $g_\mu = \langle \boldsymbol{pk}, \boldsymbol{b} \rangle$. Finally, the probability that $\hat{i} \in I$ is at least $1/n$. Combining all the above, we get that with probability ϵ_{CDH}, $\mathcal{A}_{\text{coCDH}}$ outputs a valid co-CDH break, where:

$$\epsilon_{\text{CDH}} \geq \epsilon \cdot \delta_{\mathcal{R}_{\text{TS}}} \cdot \frac{1}{n^2 q_H} \tag{62}$$

Theorem E.1. *For any PPT adversary $\mathcal{A}_{\text{TS}}$, if $\mathcal{A}_{\text{TS}}$ successfully creates a non-trivial forgery with probability ϵ, then $\mathcal{A}_{\text{coCDH}}$ breaks the q-SDH assumption with probability $\epsilon \cdot \delta_{\mathcal{R}_{\text{TS}}} / \mathsf{poly}(n, q_H)$.*