Doubly Aggregatable Signatures

Georg Fuchsbauer[1] Pranav Garimidi[2] Joachim Neu[2] Guru-Vamsi Policharla[3]
Max Resnick[4] Ertem Nusret Tas[2]

[1]TU Wien
[2]a16z Crypto Research
[3]Commonware, Inc.
[4]Anza

Abstract

Multi-signatures allow many signers to jointly generate a single (short) signature on a message. In this context, we introduce *doubly aggregatable signatures*, a new primitive that consists of two sets of signers. It enables a *layer-1* signer to create a succinct attestation, called the layer-1 signature, to the set of observed *layer-0* signatures on the same message, and publicly aggregate many layer-1 signatures into a *succinct* certificate. This certificate can be verified against the public keys, the message, and a bit map of "who observed whom". Our security model captures both the standard notion of unforgeability (*i.e.*, the adversary cannot forge layer-0 or layer-1 signatures on behalf of honest parties), and resistance to equivocation attacks, where the adversary tries to create a layer-1 signature attesting to layer-0 signatures it has not observed.

We give two concretely efficient constructions in the random-oracle model, both of which attain constant-size aggregate layer-1 signatures. The first scheme enables verification using only group additions and two pairings, but requires linear-sized layer-1 public keys per party. The second scheme achieves constant-size public keys, but requires a linear number of pairings for verification. By leveraging the random modular subset sum (RMSS) problem, both schemes attain purely algebraic verification, enabling commit-and-prove SNARKs to check succinct predicates on the bit map. A major application of doubly aggregatable signatures is incentivizing timely all-to-all vote dissemination in consensus protocols. We demonstrate the concrete efficiency of our schemes with a prototype implementation.

Keywords: Consensus protocols, votes, aggregate signatures, BLS signatures

1 Introduction

Public blockchains are operated by rational parties who seek to maximize their profits. To save on costs, they might deviate from the protocol, especially if they can do so without any repercussions. One area in which parties have been caught "cutting corners" is in vote dissemination [Anz24]. Many widely deployed consensus protocols such as Tendermint [BKM18] (Cosmos), CasperFFG [BG17, BHK+20] (Ethereum), and Alpenglow [CP23, KSW25] (Solana) require parties to send votes in a timely manner. However, if the network costs of disseminating votes are high and there are no explicit incentives to vote, parties may free-ride off the others working to keep the protocol live by not sending votes themselves. To combat this behavior, blockchains issue rewards for voting in a timely manner. However, existing reward schemes on Ethereum [Edg23b] or Solana's

TowerBFT [Anz24] only incentivize sending votes to an aggregator or a consensus leader [Anz24]. This not only puts the protocol's liveness at risk when votes are not widely broadcast, but also enables adversarial leaders to manipulate the reward scheme for consensus attacks [STM+25]. More importantly, modern consensus protocols such as Tendermint [BKM18], Alpenglow [KSW25], and variants of Simplex [CP23, CLPO25] all rely on *all-to-all* communication to avoid unnecessary network hops on the critical path.

1.1 A Strawman Solution

To incentivize timely all-to-all vote dissemination, we need a mechanism to verify that votes were broadcast to all other parties. Towards this goal, suppose each party P_j *attests* to the set of votes received from others in a timely manner by signing the votes using an aggregate signature scheme, *e.g.*, BLS [BLS01, BGLS03]. To create an attestation, P_j first aggregates the received votes yielding a short multi-signature, denoted by σ^0. It then signs σ^0 using another aggregate signature scheme and broadcasts its *signature on* σ^0 as its attestation. Now, anyone can gather attestations from all parties and combine them into a short aggregate signature, called the *certificate*. The certificate is verified against a bit matrix M, where $M[i, j] = 1$ if P_j claims to have seen a valid vote from P_i, and $M[i, j] = 0$ otherwise.

To incentivize all-to-all voting, parties are rewarded if their votes are attested to by sufficiently many others according to M. To incentivize timely voting, parties are instructed to attest to a vote only if they observe it early on. To ensure that parties report all observed votes, they are also issued rewards for attestations, which scales in proportion to the number of attested votes. If nodes were not rewarded for their attestations in addition to their votes, they might not attest to others' votes to increase their share of the voting rewards.

Challenges with the Strawman Solution Unfortunately, the scheme above suffers from two key problems. First, verifying the certificate requires not only the bit matrix M and the parties' public keys, but also the multi-signatures signed by the attestors. Each attestor might have observed a different subset of the votes, resulting in $O(n)$ different multi-signatures that must be sent as part of their respective attestations and posted on-chain. This requires $O(n\lambda)$ bits of storage and $O(n)$ pairings for verification. While the bit matrix M itself is $O(n^2)$, we can often avoid storing the entire matrix M by compressing it to a more compact form,[1] making the signature size the dominant term.

Second, attestors can *steal votes* from the certificates and other attestations. They can do this whether the parties in σ^0 sent their votes to P_j or not. While this may seem benign (as the votes were effectively forwarded to P_j via the multi-signature of $P_{j'}$), recall that we want to reward a pair of parties (P_i, P_j) *only if* P_i sent its vote to P_j in a timely manner, and P_j attested to this. But if we allowed such *vote-stealing*, we would fail to achieve our goal: rational parties would only send their votes to a handful of other nodes, knowing that other rational parties will extract their votes from these certificates, since parties are rewarded not just for voting but also for attesting to others' votes.

[1] On average, we expect that if a node broadcasts a vote, then all other parties hear the vote on time. Thus, the entries of M will mostly consist of 1's, allowing us to specify the locations of k zeros using $O(k \log n)$ bits with n signers, which can be substantially smaller than n^2.

1.2 Problem Statement

The limitations of the strawman solution leave us with the following question:

> *Can we construct a concretely efficient "doubly" aggregatable signature scheme where:*
>
> *i) a public aggregation algorithm gathers attestations to produce a succinct certificate, and*
>
> *ii) parties can succinctly attest to a set of signatures if and only if they have directly received them.*

Ideally, each attestation and the final certificate are just a few elements of an elliptic-curve group, and aggregation and verification are concretely efficient, so that they can be completed comfortably within a consensus slot.[2]

To the best of our knowledge, there has not been any prior work studying such a primitive. One can of course generically use a (recursive) zkSNARK [BCCT13, BSCTV14, BCMS20, COS20, BDFG21, KST22, BC23, KS24, BGH19, KS22] with constant-size proofs to resolve the issues: To attest to a set of signatures, parties create a proof of knowledge of signatures from the specified set of parties; the public aggregation algorithm uses recursion to produce a final compact certificate. While this solves the size issues, the zkSNARK would have to be simulation-extractable [GM17, AB19] to prevent "vote-stealing", where an adversary transforms an attestation (proof of knowledge) into an attestation by the adversary.The latency introduced to generate proofs is impractical for high-performance consensus protocols, where new proposals are generated every few hundred milliseconds.

1.3 Contributions

In Section 4, we formalize the primitive of doubly aggregatable signatures (cf. Figure 1). In Section 5, we then provide an efficient construction, where the final certificate is a *single* group element in addition to the bit matrix. The public key size of each attestor (a so-called layer-1 signer) is linear in the number of parties that create the original set of votes (*i.e.*, layer-0 signers), and generating the layer-1 public keys requires observing the layer-0 public keys. We show security in the random oracle model (ROM) assuming the hardness of co-CDH [BLS01].

In Section 5.3, we then enable a purely algebraic verification of aggregate layer-1 signatures ("certificates"), assuming the hardness of the random modular subset sum (RMSS) problem (whose complexity has been studied extensively [Bri83, Fri86, CLOS91, LO85, Lyu05]). This enables using commit-and-prove SNARKs to prove (succinctly expressible) properties about the bit matrix with succinct verification, *e.g.*, the bit matrix contains t layer-1 signers, each of whom observed t layer-0 signatures.

In Section 6, we provide an efficient construction, where both layer-0 and layer-1 signers have *constant*-size public keys that can be independently generated, at the expense of a (concretely) higher verification cost. We show security in the ROM assuming a variant of co-CDH where the adversary is given additional powers of the challenge randomness in the exponent. We again enable algebraic verification via similar techniques to Section 5.3, again assuming the hardness of the RMSS problem.

[2] Each slot is 400 ms in Solana. Although Ethereum has 12-second slots, it also has over 30,000 parties voting in a slot.

In Section 7, we evaluate the concrete performance of our schemes via an implementation in Rust. We benchmark the constructions in Sections 5 and 6 against the strawman solution for 10, 100 and 1000 parties. Both constructions outperform the strawman in terms of aggregate verification time, besides achieving succinct aggregate signatures (see Table 2). The benchmarks show the suitability of our schemes for high-performance blockchains with block times of a few hundred milliseconds.

1.4 Related Work

Multi-signatures [Ita83, MOR01] are a type of digital signature that allows n signers to jointly generate a single (short) signature on a message m. They have been based on assumptions and schemes such as RSA [Ita83, OO93], discrete logarithm, DDH and/or Schnorr signatures [NKDM03, BN06, BCJ08, NRSW20, NRS21, CKM21, Lin22, TZ23, PW23], pairings [Bol03, LOS+06, RY07, BGOY07, BDN18, CKP+23] and lattices [ES16, FH20]. One can optionally demand *public key aggregation*, so that a multi-signature can be verified against a publicly computable compact *aggregation* of the public keys, which can be computed independently of the message.

Schnorr signatures [Sch91] can be modified to support *interactive* multi-signatures [BN06], where signers interact with each other to produce a multi-signature. Follow-up works showed that they can be extended to additionally support public key aggregation [MPSW19, NRS21]. BLS signatures [BLS01] on the other hand are *non-interactive* multi-signatures [Bol03]: signers can independently sample their keys and sign messages, oblivious to other signers. Signatures from different signers can then be *publicly* aggregated into a multi-signature and verified against an aggregation of the corresponding public keys. Aggregate signatures [BGLS03, LOS+06] generalize multi-signatures by allowing signatures on *distinct* messages to be combined into a single short signature.

A well-known vulnerability in pairing-based multi-signature and aggregate signature schemes is the *rogue key attack*, where an adversary chooses a public key as a function of honest parties' keys to forge signatures. Defenses include requiring proofs of knowledge of secret keys [Bol03, RY07] and using random linear combinations for aggregation [BDN18]. Baldimtsi et al. [BCG+24] further optimize the random-coefficient approach for settings with a fixed set of public keys, leveraging the random modular subset sum (RMSS) assumption.

2 Technical Overview

We briefly discuss the syntax and security of doubly aggregatable signatures and then focus on the technical ideas behind our constructions.

2.1 Definition

Doubly aggregatable signatures are defined by two "layers" of algorithms, each consisting of key generation, signing, aggregation and verification algorithms (see Figure 1).

Layer-0: By themselves, the layer-0 algorithms constitute a multi-signature scheme: messages can be signed, and signatures from different parties on the same message can be aggregated and verified against an aggregate public key.

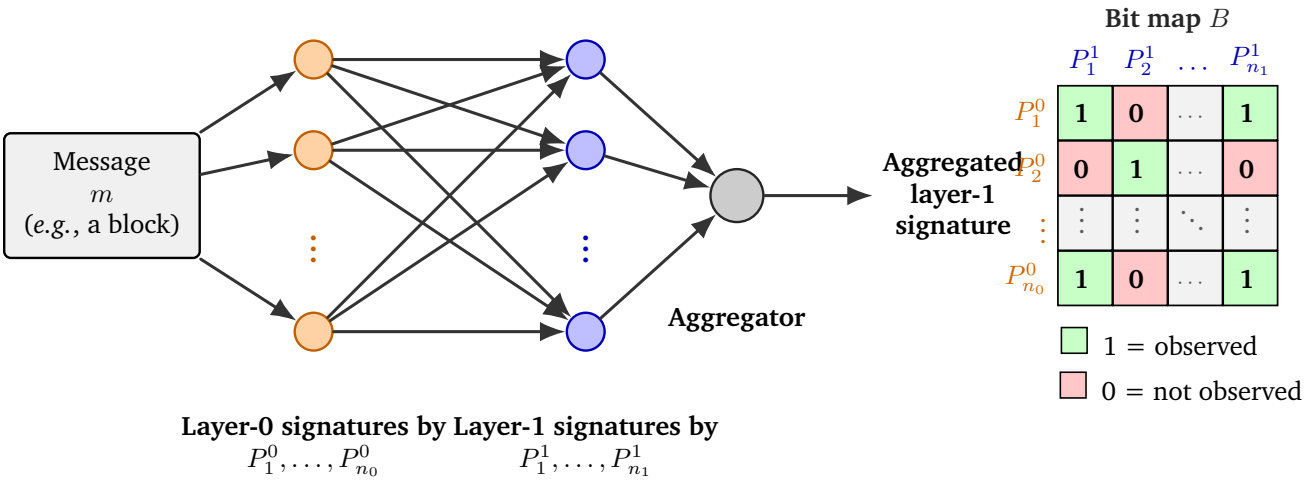

Figure 1: Overview of doubly aggregatable signatures.

Layer-1: Layer-1 algorithms differ as follows:

- There is a separate $\mathsf{KeyGen}^1(\cdot)$ algorithm that takes as input a set of layer-0 public keys $\{\mathsf{pk}_i^0\}_{i\in[n_0]}$ and outputs a layer-1 key pair $(\mathsf{pk}^1, \mathsf{sk}^1)$.
- The layer-1 secret key sk^1 can then be used to produce a layer-1 signature σ^1 for any subset $S_0 \subseteq [n_0]$ of signatures from $\{\mathsf{pk}_i^0\}_{i\in S_0}$.
- Layer-1 signatures by different parties can then be publicly aggregated to produce an aggregate signature which can be verified given the layer-1 public keys (corresponding to the aggregated layer-1 signatures) and the subset of layer-0 public keys whose layer-0 signatures were attested.

Each layer has its own notion of unforgeability. Layer-0 unforgeability is the standard notion of multi-signature unforgeability.

Layer-1 unforgeability. For this we define two security properties, which we model using a game-based definition. The adversary controls a subset of "corrupt" users and is allowed to make layer-0 and layer-1 signature queries for honest parties. We restrict its layer-1 queries to attesting only one set of signatures for every message (this restriction is justified by the intended applications of our primitive). The adversary wins if it returns one of the following:

(a) a layer-1 signature by an honest party P that is different from the layer-1 queries made to P, or

(b) a layer-1 signature by a corrupt party, claiming to have seen a layer-0 signature from an honest party that it has not queried.

The former captures the natural notion of unforgeability for layer-1 signatures from honest parties and the latter captures protections against vote-stealing: no adversary can claim to have seen a vote (i.e., a layer-0 signature) from an honest party it has not seen, even if another honest party has attested to this vote (i.e., it is included in a layer-1 signature).

2.2 Doubly Aggregatable Signature Schemes

Our starting point is the BLS signature scheme [BLS01], defined as follows. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be groups of prime order p with generators g_1 and g_2 respectively, and let $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear map ("pairing"). The public key for BLS signatures is set to be $\mathsf{pk} = g_2^{\mathsf{sk}}$ for $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ and a signature on a message m is computed as $\sigma = H(m)^{\mathsf{sk}}$, where $H\colon \{0,1\}^* \to \mathbb{G}_1$ is a cryptographic hash function modeled as a random oracle. Signatures can be verified using the pairing to check whether $e(H(m), \mathsf{pk}) \stackrel{?}{=} e(\sigma, g_2)$.

In our doubly aggregatable signature schemes, the layer-0 algorithms are essentially BLS multi-signatures. The key generation outputs a BLS key pair $(\mathsf{pk}^0, \mathsf{sk}^0)$, and signatures $\{\sigma_i^0\}_{i\in[n_0]}$ by n_0 parties with public keys $\{\mathsf{pk}_i^0\}_{i\in[n_0]}$ are aggregated as $\sigma_{\mathsf{agg}}^0 = \prod_i \sigma_i^0$, which can be verified against the aggregated public key $\mathsf{pk}_{\mathsf{agg}}^0 = \prod_i \mathsf{pk}_i^0$ like a regular signature. We defend against rogue key attacks by demanding a proof of knowledge of the secret key [Bol03].

Strawman Solution Let layer-1 keys also be BLS key pairs $(\mathsf{pk}^1, \mathsf{sk}^1)$. A layer-1 signer j first aggregates the observed layer-0 signatures: $\sigma_{\mathsf{agg},j}^0 = \prod_i \sigma_i^0$. It then signs this multi-signature and publishes the following tuple as its layer-1 signature:

$$\left(\sigma_j^{1,1} = H(\sigma_{\mathsf{agg},j}^0)^{\mathsf{sk}_j^1},\ \sigma_j^{1,2} = \sigma_{\mathsf{agg},j}^0\right)$$

As mentioned in the introduction, the strawman solution falls short of the desiderata for efficient doubly aggregatable signatures. First, although we can aggregate $\sigma_j^{1,1}$ across different parties as in BLS signatures, verification of the aggregate layer-1 signature $\sigma_{\mathsf{agg}}^{1,1} = \prod_j \sigma_j^{1,1}$ requires not only the bit map M and the layer-1 public keys, but also the signed multi-signatures $\sigma_{\mathsf{agg},j}^0 = \sigma_j^{1,2}$ for all j:

$$e(\sigma_{\mathsf{agg}}^{1,1}, g_2) \stackrel{?}{=} \prod\nolimits_j e(\sigma_{\mathsf{agg},j}^0, \mathsf{pk}_j^1)$$

Therefore, the layer-1 signatures cannot be fully aggregated to yield a succinct signature.

Second, an adversarial party j can attest to the layer-0 signatures attested by another party j' by signing $\sigma_{j'}^{1,2}$ with its layer-1 secret key and publishing

$$\sigma_j^1 = \left(H(\sigma_{j'}^{1,2})^{\mathsf{sk}_j^1}, \sigma_{j'}^{1,2}\right),$$

without having seen the constituent layer-0 signatures (*i.e.*, steal votes). In other words, it wins the layer-1 unforgeability game.

One way to achieve layer-1 unforgeability is to return, as the layer-1 signature, a *simulation-extractable* zero-knowledge argument of knowledge of a witness from the following set:

$$\left\{(\mathsf{sk}^1, \sigma_{\mathsf{agg}}^0) \mid (\mathsf{pk}^1, \mathsf{sk}^1) \in R \wedge \mathsf{AggVerify}^0(\{\mathsf{pk}_i^0\}_{i\in[n_0]}, m, \sigma_{\mathsf{agg}}^0) = 1\right\}$$

It attests to the knowledge of a layer-1 secret key such that $(\mathsf{pk}^1, \mathsf{sk}^1) \in R$ for a known key relation R, and a layer-0 multi-signature on the message m. Simulation-extractability would ensure that the proof cannot be *mauled* by an adversary to produce a layer-1 signature under its public key.

One could use Groth-Sahai [GS08] proofs to prove knowledge of the multi-signature and secret key. Since they have *algebraic* verification, they can be aggregated using known techniques [GMN22, BMM+21], resulting in a layer-1 aggregate signature of size $O(\log n_1)$. However, for

these techniques the concrete verification costs can be quite high due to the target group operations performed by the verifier.

Moreover, Groth-Sahai proofs are not simulation-extractable; they can be made so using any one-time signature [Gro06], but it is not clear how to choose a signature so that the overall proof remains aggregatable and succinct in n_1. Instead, we take a completely different, and far simpler approach by observing that zero-knowledge proofs are *not necessary* to prevent vote stealing.

First Attempt. As a first attempt, given a set of layer-0 signatures $\{\sigma_i^0 \coloneqq H(m)^{\mathsf{sk}_i^0}\}_{i\in[n_0]}$, suppose we set the layer-1 signature to be:

$$\sigma^1 = \textstyle\prod_{i\in[n_0]} (\sigma_i^0)^{\mathsf{sk}^1} .$$

To aid in verification, suppose we also included *cross-terms* of the form

$$\left\{ g_2^{\mathsf{sk}_i^0 \cdot \mathsf{sk}^1} \right\}_{i\in[n_0]}$$

in the layer-1 public key. Then the layer-1 signature can be verified as

$$e(\sigma^1, g_2) \stackrel{?}{=} e\Big(H(m),\ \textstyle\prod_{i\in[n_0]} g_2^{\mathsf{sk}_i^0 \cdot \mathsf{sk}^1}\Big) . \tag{1}$$

Signatures $\{\sigma_j^1\}_{j\in[n_1]}$ from different public keys $\{\mathsf{pk}_j^1\}_{j\in[n_1]}$ can now be easily aggregated $\sigma_{\mathsf{agg}}^1 = \prod_{j\in[n_1]} \sigma_j^1$ and verified!

$$e(\sigma_{\mathsf{agg}}^1, g_2) \stackrel{?}{=} e\Big(H(m),\ \textstyle\prod_{j\in[n_1]} \prod_{i\in[n_0]} g_2^{\mathsf{sk}_i^0 \cdot \mathsf{sk}_j^1}\Big) \tag{2}$$

Although correctness holds, there is an entirely new class of *rogue public key attacks* that we must handle. Recall that for multi-signatures, unforgeability states that it is computationally infeasible to produce a multi-signature that includes an honest party P, unless the adversary explicitly queried a signature from P. If the adversary can choose its public key pk^* (say) as a function of honest party public keys $\{\mathsf{pk}_i\}_{i\in[n]}$, it could set

$$\mathsf{pk}^* = \big(\textstyle\prod_{i\in[n]} \mathsf{pk}_i\big)^{-1} \cdot g_1^{\mathsf{sk}^*} .$$

It can then produce a BLS signature on behalf of all honest users as $\sigma = H(m)^{\mathsf{sk}^*}$, which verifies against $\mathsf{pk}_{\mathsf{agg}} = \mathsf{pk}^* \cdot \prod_{i\in[n]} \mathsf{pk}_i$. A proof of knowledge of secret key ("proof of possession") prevents this, as an adversary can no longer create a public key that *non-trivially* depends on honest parties' public keys.

While this secures BLS multi-signatures, it is not sufficient for the sketched doubly aggregatable scheme. The reason is that proofs of possession provide no guarantees about the distribution of corrupt public keys – in fact, the adversary is free to pick any desired secret key. This manifests as two concrete attacks on the scheme above:

Equivocation. Suppose an adversary chooses layer-0 secret keys $(\mathsf{sk}_1^0, \mathsf{sk}_2^0, \mathsf{sk}_3^0)$ such that $\mathsf{sk}_1^0 = \mathsf{sk}_2^0 + \mathsf{sk}_3^0$. Let $(\mathsf{pk}_1^0, \mathsf{pk}_2^0, \mathsf{pk}_3^0)$ be the corresponding public keys and pk_4^0 be some layer-0 honest party public key. When an honest party pk^1 attests to having seen signatures from $(\mathsf{pk}_1^0, \mathsf{pk}_4^0)$, creating the layer-1 signature $H(m)^{(\mathsf{sk}_1^0 + \mathsf{sk}_4^0)\mathsf{sk}^1}$, a malicious aggregator can then *maul* the statement to "pk^1 saw signatures from $(\mathsf{pk}_2^0, \mathsf{pk}_3^0, \mathsf{pk}_4^0)$", and the same signature would pass verification. This constitutes a break of layer-1 unforgeability (of type (a) in Section 2.1).

Table 1: Comparison of our doubly aggregatable signature schemes against an (insecure) strawman solution, where n_0 is the number of layer-0 public keys, and $\mathbb{G}$ denotes the base group operation and P denotes pairings. In all our schemes, layer-0 public keys, aggregate signatures, and layer-1 signatures consist of $O(1)$ group elements. "L[ayer]-1 AggSig size" refers to the *additional* information sent on top of the attestation bit matrix.

Construction	Assumption	Layer-1 PK Size	Verification Cost	L-1 AggSig size
Strawman	insecure scheme	$O(1)$	$O(n_0 \cdot n_1)$ $\mathbb{G}$ + $O(n_1)$ P	$O(n_0)$
Sect. 5	co-CDH (Def. 1)	$O(n_0)$	$O(n_0 \cdot n_1)$ $\mathbb{G}$ + $O(n_1)$ MSM(n_0) + $O(1)$ P	$O(1)$
Sect. 5.3	co-CDH + RMSS (Def. 3)	$O(n_0)$	$O(n_0 \cdot n_1)$ $\mathbb{G}$ + $O(1)$ P	$O(1)$
Sect. 6	(2,1)-CDH (Def. 2)	$O(1)$	$O(n_0 \cdot n_1)$ $\mathbb{G}$ + $O(n_1)$ $\mathbb{G}$-exp + $O(n_1)$ P	$O(1)$
Sect. 6.3	(2,1)-CDH + RMSS	$O(1)$	$O(n_0 \cdot n_1)$ $\mathbb{G}$ + $O(n_1)$ P	$O(1)$

Zero-izing. The adversary could also choose layer-1 secret keys $(\mathsf{sk}_1^1, \mathsf{sk}_2^1, \mathsf{sk}_3^1)$ such that $\mathsf{sk}_1^1 + \mathsf{sk}_2^1 + \mathsf{sk}_3^1 = 0$. Let $(\mathsf{pk}_1^1, \mathsf{pk}_2^1, \mathsf{pk}_3^1)$ be their respective public keys and pk^0 be an honest party layer-0 public key. A malicious aggregator can now produce a layer-1 aggregate signature $H(m)^0 = H(m)^{\mathsf{sk}^0(\mathsf{sk}_1^1 + \mathsf{sk}_2^1 + \mathsf{sk}_3^1)}$ claiming that all three parties have seen a signature from pk^0 even though no queries were made to pk^0. (This constitutes a break of layer-1 unforgeability of type (b) in Section 2.1.)

2.3 Handling Rogue Public Keys

We provide three different (concretely efficient) constructions that are provably secure against the above attacks.

Random Linear Combination? A natural solution is to use a random linear combination (sampled using a random oracle) during attestation Eq. (1) and aggregation Eq. (2). A similar strategy was used by Boneh et al. [BDN18] to avoid proofs of possession in BLS multi-signatures. We expect that this would work, but verification requires a multi-scalar multiplication (MSM) of size $O(n_0 \cdot n_1)$. For context, at $n_0 = n_1 = 1024$ (for comparison, Solana has ~ 800 validators), this would take over 11 seconds! But the same number of group operations requires just 600 ms.[3] In applications where verification happens on a *critical* path, but public keys are updated infrequently, minimizing the verifier time is crucial. In fact, this is precisely the reason why Ethereum uses proofs of possession for BLS multi-signatures despite the additional complexity it introduces [Edg23a].

We now outline three approaches that a) minimize expensive group exponentiations during verification and b) satisfy our security definition, thus in particular preventing equivocation and zero-izing attacks. The main tradeoff (see Table 1) across these schemes is in the hardness assumption, public key size and verifier complexity.

A Simple First Construction. Let $\{X_i = g_2^{x_i} : x_i \leftarrow\!\!\$\ \mathbb{Z}_p\}_{i \in [n_0]}$ denote the layer-0 public keys. Layer-0 algorithms in all three constructions are still BLS multi-signatures. In our first construc-

[3] Single-threaded benchmarks on a 2019 Macbook Pro using `arkworks` on the $\mathbb{G}_1$ curve of BLS12-381.

tion (Section 5), we still use a random linear combination during attestation, but instead of sampling it via a random oracle, we embed it in (the exponent of) the layer-1 public key during key generation. That is, the j-th party's layer-1 secret key is now n_0 field elements $\{y_{j,i} \leftarrow\$ \mathbb{Z}_p\}_{i\in[n_0]}$ and the public key is $\{Y_{j,i} = g_2^{x_i \cdot y_{j,i}}\}_{i\in[n_0]}$, computed from the layer-0 keys $\{X_i\}_{i\in[n_0]}$. To attest to layer-0 signatures $\{\sigma_i^0\}_{i\in S^0}$ by a set $S^0 \subseteq [n_0]$ of signers, the j-th party now creates a layer-1 signature using $y_{j,i}$'s as the linear combination

$$\sigma_j^1 = \prod_{i\in S^0} (\sigma_i^0)^{y_{j,i}} .$$

Even though an honest party uses the *same* linear combination for all layer-1 signatures, the linear combination itself is *hidden* in the exponent. We show that, assuming hardness of CDH, *equivocation* is now infeasible, *i.e.* the adversary cannot produce two different sets of public keys that verify with respect to the same honest party layer-1 signature. Furthermore, verifying a layer-1 signature only requires $O(n_0)$ group operations: $e(\sigma_j^1, g_2) \stackrel{?}{=} e(H(m), \prod_{i\in S_0} Y_{j,i})$.

Next, we focus on zero-izing attacks that manifest at the aggregation stage. In our first construction (Section 5), we use a random linear combination $\{\rho_j\}_{j\in[n_1]}$ sampled appropriately via a random oracle. The aggregation of layer-1 signatures from a subset S^1 of signers is then computed as $\sigma_{\mathsf{agg}}^1 = \prod_{j\in S^1} (\sigma_j^1)^{\rho_j}$. This can be verified using $O(n_0 \cdot n_1)$ group operations, $O(n_1)$-sized MSM and $O(1)$ pairings as:

$$e\big(\sigma_{\mathsf{agg}}^1,\, g_2\big) \stackrel{?}{=} e\Big(H(m),\, \prod_{j\in S^1} \big(\prod_{i\in S_j^0} Y_{j,i}\big)^{\rho_j}\Big)$$

At $n_0 = n_1 = 1024$, the concrete cost of verification is already dominated by the $O(n_0 \cdot n_1)$ group operations, and the verifier reads the $O(n_0 \cdot n_1)$ attestation matrix M. So, there seems to be very little incentive to improve the scheme by removing the random linear combination during aggregation. However, consider an application where the verifier is only interested in learning a *predicate* about the attestation matrix M, *e.g.* whether there are at least t parties who saw signatures from at least t parties. Here, we would like a protocol with succinct $O(\mathsf{poly}(\log n_0 + \log n_1))$ verification time.

There are techniques based on folding [Lee21, BMM+21, GMN22] and the univariate sumcheck [GJM+24, DCX+23] to produce succinct proofs of *aggregation* of BLS public keys, where the coefficients used for aggregation are made available in the form of a short commitment. One can then use a commit-and-prove SNARK, to prove arbitrary statements about the coefficients.

Algebraic Verification. The mentioned techniques cannot be applied to our first construction, precisely because we sample the coefficients for aggregation using a random oracle. Thus, in order to produce a succinct proof, we need a doubly aggregatable signature with purely *algebraic* verification. This leads us to our second construction (Section 5.3) based on the hardness of CDH and the well-studied random modular subset sum (RMSS) problem [Bri83, Fri86, CLOS91, LO85, Lyu05]. RMSS means solving the subset sum problem over $\mathbb{Z}_p$ which is conjectured to be hard when the elements are sampled uniformly at random.

The main observation is that if all secret keys were sampled uniformly, then an adversary launching a zero-izing attack will have to solve the RMSS problem. Of course, an adversary is not obliged to sample its keys uniformly, but we can *rerandomize* all public keys *after* key generation to achieve the same effect. This final scheme aggregates layer-1 public keys by simply computing their product – making verification algebraic and hence conducive to generating a succinct proof. A similar strategy was previously used [BCG+24] to optimize verification of BLS multi-signatures.

Our second construction has an additional advantage: if there are only a few layer-1 signatures missing – say $O(\sqrt{n_0})$ – then the verification can be carried out using just $O(\sqrt{n_0})$ group operations, as we can *divide out* the missing public keys from a fully aggregated public key computed in advance.

2.4 Reducing Public Key Sizes

In both of the above constructions, the layer-1 public key is linear in the number of parties in layer-0. While the running time of the verifier must grow as $\Omega(n_0 \cdot n_1)$ (as it has to at least read the input), this does not mean the public key size must do so. Recall that the *cross-terms*, which are needed for the verification of layer-1 signatures, are the reason layer-1 public keys are $O(n_0)$ in the above constructions. Thus, if we hope to reduce layer-1 public key size to $O(1)$, we need a new approach to prevent equivocation.

Suppose both layer-0 and layer-1 public keys are just BLS public keys (along with proofs of knowledge of the secret key). Let these be $\{X_i = g_2^{x_i} : x_i \leftarrow\$ \mathbb{Z}_p\}_{i\in[n_0]}$ and $\{Y_j = g_2^{y_j} : y_j \leftarrow\$ \mathbb{Z}_p\}_{j\in[n_1]}$, respectively. Layer-0 signatures are standard BLS signatures, and the layer-1 signature by party P_j on the set $S^0 \subseteq [n_0]$ is $\sigma_j^1 = \prod_{i\in S^0}(\sigma_i^0)^{y_j}$. But now, the layer-1 public keys do not contain cross-terms, preventing computation of $\prod_{i\in S^0} g_2^{x_i\cdot y_j}$ for verifying the layer-1 signature as before.

We solve this by letting the layer-1 signer provide $\prod_{i\in S^0} g_2^{x_i\cdot y_j}$ as part of the layer-1 signature and *prove* that it was computed correctly. In fact, if we place the layer-0 and layer-1 public keys in different base groups, this can simply be checked via the pairing. Concretely, we now set $\{Y_j = g_1^{y_j} : y_j \leftarrow\$ \mathbb{Z}_p\}_{j\in[n_1]}$ and layer-1 signatures contain two terms:

$$\sigma_j^{1,1} = \prod_{i\in S^0}(\sigma_i^0)^{y_j}, \quad \sigma_j^{1,2} = \prod_{i\in S^0}(X_i)^{y_j},$$

which can now be verified as:

$$e\big(\sigma_j^{1,1},\, g_2\big) \stackrel{?}{=} e\left(H(m),\, \sigma_j^{1,2}\right), \quad e\big(g_1,\, \sigma_j^{1,2}\big) \stackrel{?}{=} e\Big(Y_j,\, \textstyle\prod_{i\in S^0} X_i\Big).$$

While correctness holds, equivocation is still possible using the same strategy. Stepping back for a moment, we see that equivocation is possible because the signer does not *commit* to the set they saw in any meaningful manner. Thus, we *sign* the set S^0 by adding another BLS signature, using a different hash function H_1 and an additional layer-1 key component $Z = g_2^z$, namely $\sigma'_j = H_1(m, S^0, j)^z$.

While this is secure, it would further increase the signature size. Fortunately, we show that we can avoid an additional element σ'_j in the layer-1 signature. Assuming a simple variant of CDH (Definition 2), we show that defining the signature component as the *product* of $\sigma_j^{1,1}$ and σ'_j results in a secure scheme.

Under RMSS we can again define a variant with algebraic verification and reduce the verification cost for the aggregate layer-1 signature to $O(n_0 n_1)$ group operations and $O(n_1)$ pairings. Algebraic verification enables using commit-and-prove SNARKs to verify important properties about the $O(n_0 n_1)$-sized bit map succinctly, without reading the full matrix. Finally, as in the first construction, further optimizations are possible in common situations where almost all layer-1 signers create signatures on the layer-0 signatures. This further reduces the number of group operations and pairings in the verification to a constant.

In Appendix 8.1, we show how the verifier complexity can be reduced from $O(n)$ pairings to

$O(\log(n))$ target group exponentiations and $O(1)$ source group MSMs, at the expense of more work for the aggregators, using the statistically-secure Inner Pairing Product (SIPP) in [BMM+21, Appendix B].

In Appendix 8.2, we generalize our schemes to the setting of layer-0 signatures on multiple messages.

3 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. The security parameter is denoted by $\lambda \in \mathbb{N}$. A function $f\colon \mathbb{N} \to \mathbb{N}$ is said to be polynomial if there exists a constant c such that $f(n) \leq n^c$ for all $n \in \mathbb{N}$, and we write $\mathsf{poly}(\cdot)$ to denote such a function. A function $f\colon \mathbb{N} \to [0,1]$ is negligible if for every $c \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $f(n) < n^{-c}$, and we write $\mathsf{negl}(\cdot)$ to denote such a function. A probability is *noticeable* if it is not negligible, and *overwhelming* if it is equal to $1 - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\lambda)$. For a set $\mathcal{S}$, we write $s \leftarrow\!\!\$\, \mathcal{S}$ to indicate that s is sampled uniformly at random from $\mathcal{S}$. We denote the domain of a mapping Q by $\mathsf{dom}(Q)$. An algorithm $\mathcal{A}$ is PPT (probabilistic polynomial-time) if its running time is polynomial in the size of its input.

The Random Oracle Model. In the random oracle model (ROM), parties are given oracle access to some function H that is sampled uniformly at random from the space of all functions $H\colon \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are finite non-empty sets. That is, parties can query their oracle on an input $x \in \mathcal{X}$ and receive in return $H(x) \in \mathcal{Y}$. When proving the security of a protocol in the random oracle model, the reduction can "control" the oracle, observing queries made by the adversary and simulating responses.

Non-Interactive Zero-Knowledge Proofs of Knowledge. Let $\mathcal{L}$ be an NP-language and $\mathcal{R}$ the corresponding NP-relation. A Simulation Extractable-NIZK for $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: Takes as input a security parameter, and outputs a common reference string crs and trapdoor td.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: Takes as input crs and any pair $(x, w) \in \mathcal{R}$, and outputs a proof π for the statement $x \in \mathcal{L}$.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: Takes as input crs, statement x and proof π, and outputs a bit b indicating acceptance or rejection.
- $\mathsf{SimProve}(\mathsf{crs}, \mathsf{td}, x) \to \pi$: Takes as input crs, trapdoor td, and statement x and outputs a *simulated* proof π.

We will drop the crs term wherever implicit. A Simulation Extractable-NIZK satisfies the following properties:

- *Perfect Completeness.* An SE-NIZK satisfies perfect completeness if for any $(x, w) \in \mathcal{R}$, we have

$$\Pr\left[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)\,; \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) : \mathsf{Verify}(x, \pi) = 1\right] = 1.$$

- *Proof of knowledge (and soundness).* For every PPT adversary $\mathcal{A}$, there is a PPT extractor Ext such that

$$\Pr\left[\begin{matrix}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,\pi) \leftarrow \mathcal{A}(\mathsf{crs})\,;\ w \leftarrow \mathsf{Ext}(\mathsf{crs})\end{matrix} : \begin{matrix}\mathsf{Verify}(x,\pi)=1 \\ (x,w)\notin\mathcal{R}\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

- *Computational Zero-knowledge.* There exists an algorithm SimProve such that for all PPT adversaries $\mathcal{A}$:

$$\left|\Pr\left[\begin{matrix}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs},x,w)\end{matrix} : \begin{matrix}(x,w)\in\mathcal{R} \\ \mathcal{A}(\pi)=1\end{matrix}\right] - \Pr\left[\begin{matrix}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{SimProve}(\mathsf{crs},\mathsf{td},x)\end{matrix} : \begin{matrix}(x,w)\in\mathcal{R} \\ \mathcal{A}(\pi)=1\end{matrix}\right]\right| \leq \mathsf{negl}(\lambda).$$

- *Weak Simulation-Extractability.* For every PPT adversary $\mathcal{A}$, there exists a PPT extractor Ext such that

$$\Pr\left[\begin{matrix}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,\pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}}(\mathsf{crs}): \\ w \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{transcript}_{\mathcal{A}})\end{matrix} \begin{matrix}\mathsf{Verify}(x,\pi)=1 \\ \wedge\,(x,w)\notin\mathcal{R} \\ \wedge\, x\notin Q\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

  where $\mathcal{A}$ has oracle access to $\mathsf{SimProve}(\mathsf{crs},\mathsf{td},\cdot)$, $\mathsf{transcript}_{\mathcal{A}}$ is the transcript of the adversary, and Q is a list of queries made by the adversary.

We will also demand that the proof is straight-line extractable. This means that the extractor can, on input a valid proof and (potentially) the list of random-oracle queries made by the adversary (prover), extract a witness with overwhelming probability without rewinding the prover.

Bilinear Groups. We let e denote an efficiently computable non-degenerate bilinear map from the groups $\mathbb{G}_1 \times \mathbb{G}_2$ to a target group $\mathbb{G}_T$, all of order p. We let g_1 and g_2 denote generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. We assume an algorithm GGen that on input a security parameter λ returns the description of a bilinear group with order p of bitlength λ.

Cryptographic Hardness Assumptions. Security of our constructions relies on the hardness of the following computational problems.

Definition 1 (co-CDH)**.** *The* **co-CDH assumption** *for a bilinear group generator* GGen *states that for all PPT algorithms $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,g_1,g_2,e) \leftarrow \mathsf{GGen}(1^\lambda); \\ \alpha,\beta \leftarrow\!\!\$\ \mathbb{Z}_p\,;\ w \leftarrow \mathcal{A}\big(g_1^\alpha, g_1^\beta, g_2^\alpha\big) : w = g_1^{\alpha\beta}\right]$$

Note that w is a solution for an instance (u_1,v,u_2) iff $e(w,g_2)=e(v,u_2)$.

Definition 2 ($(2,1)$-CDH)**.** *The* $(2,1)$**-CDH assumption** *for a bilinear group generator* GGen *states that for all PPT algorithms $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,g_1,g_2,e) \leftarrow \mathsf{GGen}(1^\lambda); \\ \alpha,\beta \leftarrow\!\!\$\ \mathbb{Z}_p\,;\ w \leftarrow \mathcal{A}\big(g_1^\alpha, g_1^{1/\alpha}, g_2^\alpha, g_2^{1/\alpha}, g_1^\beta, g_1^{\beta/\alpha}\big) : w = g_1^{\alpha\beta}\right]$$

In other words, given an instance $u_1 := g_1^{\alpha}, u_1' := g_1^{1/\alpha}, u_2 := g_2^{\alpha}, u_2' := g_2^{1/\alpha}, v_1 := g_1^{\beta}, v_1' := g_1^{\beta/\alpha}$, a solution w to $(u_1, u_1', u_2, u_2', v_1, v_1')$ must satisfy $e(w, g_2) = e(v_1, u_2)$. Assuming that the group generation algorithm selects the generators g_1 and g_2 of $\mathbb{G}_1$ and $\mathbb{G}_2$ uniformly at random, this problem is equivalent to computing $g_1^{\alpha^2\beta}$ given $(g_1^{\alpha}, g_2^{\alpha}, g_1^{\alpha^2}, g_2^{\alpha^2}, g_1^{\beta}, g_1^{\alpha\beta})$, hence the name $(2,1)$-CDH, since α is given up to its 2nd power and β up to its first. Both problems are instances of Boyen's uber assumption framework [Boy08] and therefore hold in the generic group model [Sho97]. Note that the $(2,1)$-CDH assumption implies the co-CDH assumption.

Definition 3 (RMSS)**.** *The* **random modular subset sum (RMSS)** *assumption with parameters (n, p) states that for all PPT algorithms, given $\{s_j\}_{j\in[n]} \leftarrow\!\!\$\ \mathbb{Z}_p^n$ and $t \leftarrow\!\!\$\ \mathbb{Z}_p$, it is hard to find a set $J \subseteq [n]$ such that $\sum_{j\in J} s_j = t$.*

Although a solution exists with non-negligible probability when $n = c\log(p)$ for constant c, solving the RMSS problem is hardest for this parameter range [IN96], whereas the probability of the existence of a solution becomes negligible at $n = O(\sqrt{\log(p)})$.

4 Definition

In this section, we formally define doubly aggregatable signatures.

Definition 4 (Doubly-Aggregatable Signatures)**.** *A Doubly-Aggregatable Signature scheme for message space $\mathcal{M}$ can be viewed as two layers of algorithms, layer-0 and layer-1, each with a key generation, signing, aggregation and verification algorithm.*

$\mathsf{Setup}(1^{\lambda}) \to \mathsf{pp}$**:** *On input λ, outputs public parameters* pp*.*

Layer 0. *The layer-0 algorithms are defined as follows:*

$\mathsf{KeyGen}^0(\mathsf{pp}) \to (\mathsf{pk}^0, \mathsf{sk}^0)$**:** *On input the parameters, samples a layer-0 key pair $(\mathsf{pk}^0, \mathsf{sk}^0)$.*

$\mathsf{Sign}^0(\mathsf{sk}^0, m) \to \sigma^0$**:** *On input a layer-0 secret key sk^0 and a message $m \in \mathcal{M}$, outputs a signature σ^0 on m.*

$\mathsf{Verify}^0(\mathsf{pk}^0, m, \sigma^0) \to \{0,1\}$**:** *On input public key pk^0, message m and signature σ^0, outputs 1 ifσ^0 is a valid signature on m under pk^0, and 0 otherwise.*

$\mathsf{Aggregate}^0(\{(\mathsf{pk}_i^0, \sigma_i^0)\}_{i\in[n_0]}, m) \to \sigma_{\mathsf{agg}}^0$**:** *On input a set of tuples containing public keys and signatures $\{(\mathsf{pk}_i^0, \sigma_i^0)\}_{i\in[n_0]}$ on a message m, outputs an aggregate signature σ_{agg}^0.*

$\mathsf{AggVerify}^0(\{\mathsf{pk}_i^0\}_{i\in[n_0]}, m, \sigma_{\mathsf{agg}}^0) \to \{0,1\}$**:** *On input an aggregate signature σ_{agg}^0, a set of public keys $\{\mathsf{pk}_i^0\}_{i\in[n_0]}$ and a message m, outputs 1 if σ_{agg}^0 is a valid aggregate signature on m under $\{\mathsf{pk}_i^0\}_{i\in[n_0]}$ and 0 otherwise.*

Layer 1. *The layer-1 algorithms are defined as follows:*

$\mathsf{KeyGen}^1(\mathsf{pp}, \{\mathsf{pk}_i^0\}_{i\in[n_0]}) \to (\mathsf{pk}^1, \mathsf{sk}^1)$**:** *On input the parameters and (optionally) the set of layer-0 public keys $\{\mathsf{pk}_i^0\}_{i\in[n_0]}$ that will be attested to, samples a layer-1 key pair $(\mathsf{pk}^1, \mathsf{sk}^1)$.*

$\mathsf{Sign}^1(\mathsf{sk}^1, \{\mathsf{pk}_i^0, \sigma_i^0\}_{i\in S^0}, m) \to \sigma^1$**:** *On input a secret key, a set of layer-0 public-key/signature pairs $\{(\mathsf{pk}_i^0, \sigma_i^0)\}_{i\in S^0}$ with $S^0 \subseteq [n_0]$, each valid for a message m, outputs a signature σ^1, which attests to having seen each of the signatures σ_i^0 on m.*

$\mathsf{Verify}^1(\mathsf{pk}^1, \{\mathsf{pk}_i^0\}_{i\in S^0}, m, \sigma^1) \to \{0,1\}$**:** *On input a layer-1 public key pk^1, a set of layer-0 public keys $\{\mathsf{pk}_i^0\}_{i\in S^0}$ with $S^0 \subseteq [n_0]$, a message m and a signature σ^1, outputs 1 if σ^1 is a valid signature under pk^1, attesting to having seen an aggregate signature on m under $\{\mathsf{pk}_i^0\}_{i\in S^0}$.*

$\mathsf{Aggregate}^1(\{\mathsf{pk}_j^1, \sigma_j^1, \{\mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in[n_1]}, m) \to \sigma_{\mathsf{agg}}^1$**:** *On input a set of tuples, each containing a layer-1 public key pk_j^1, and corresponding layer-1 signature σ_j^1, attesting to having seen aggregate signatures on m under a set of layer-0 public keys $\{\mathsf{pk}_i^0\}_{i\in S_j^0}$, where $S_j^0 \subseteq [n_0]$, outputs an aggregate signature σ_{agg}^1.*

$\mathsf{AggVerify}^1(\{\mathsf{pk}_j^1, \{\mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in[n_1]}, m, \sigma_{\mathsf{agg}}^1) \to \{0,1\}$**:** *On input a set of tuples, outputs 1 iff σ_{agg}^1 is a valid aggregate signature on m under the set of layer-1 public keys $\{\mathsf{pk}_j^1\}_j$, each attesting to having seen an aggregate signature on m under their respective set of layer-0 public keys $\{\mathsf{pk}_i^0\}_{i\in S_j^0}$.*

Remark 5. *One can extend the above definition to k-aggregatable signatures in a natural manner, where the j-th layer attests to a statement about the output of the $(j-1)$-th layer, but we restrict ourselves to the case of $k = 2$ for simplicity.*

Definition 6 (Correctness)**.** *A Doubly-Aggregatable Signature scheme is said to satisfy perfect correctness if the following conditions hold:*

Layer-0. *For all $\lambda, n_0 \in \mathbb{N}$ and all $m \in \mathcal{M}$, let*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
- $\{(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}^0(\mathsf{pp})\}_{i\in[n_0]}$
- $\{\sigma_i^0 \leftarrow \mathsf{Sign}^0(\mathsf{sk}_i^0, m)\}_{i\in[n_0]}$
- $\sigma_{\mathsf{agg}}^0 \leftarrow \mathsf{Aggregate}^0(\{(\mathsf{pk}_i^0, \sigma_i^0)\}_{i\in[n_0]}, m)$

then $\Pr[\mathsf{Verify}^0(\mathsf{pk}_1^0, m, \sigma_1^0) = 1 \ \wedge \ \mathsf{AggVerify}^0(\{\mathsf{pk}_i^0\}_{i\in[n_0]}, m, \sigma_{\mathsf{agg}}^0) = 1] = 1.$

Layer-1. *For all $\lambda, n_0, n_1 \in \mathbb{N}$, $m \in \mathcal{M}$, for all subsets $S^1 \subseteq [n_1]$, all sets of subsets $\{S_j^0 \subseteq [n_0]\}_{j\in S^1}$, let*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
- $\{(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}^0(\mathsf{pp})\}_{i\in[n_0]}$
- $\{(\mathsf{pk}_i^1, \mathsf{sk}_i^1) \leftarrow \mathsf{KeyGen}^1(\mathsf{pp}, \{\mathsf{pk}_i^0\}_{i\in[n_0]})\}_{i\in[n_1]}$
- $\{\sigma_i^0 \leftarrow \mathsf{Sign}^0(\mathsf{sk}_i^0, m)\}_{i\in[n_0]}$
- $\left\{\sigma_j^1 \leftarrow \mathsf{Sign}^1(\mathsf{sk}_j^1, \{\mathsf{pk}_i^0, \sigma_i^0\}_{i\in S_j^0}, m)\right\}_{j\in S^1}$
- $\sigma_{\mathsf{agg}}^1 \leftarrow \mathsf{Aggregate}^1(\{\mathsf{pk}_j^1, \sigma_j^1, \{\mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in S^1}, m)$

then, $\Pr\left[\mathsf{Verify}^1\left(\mathsf{pk}_1^1, \{\mathsf{pk}_i^0\}_{i\in S_1^0}, m, \sigma_1^1\right) = 1 \wedge \mathsf{AggVerify}^1\left(\{\mathsf{pk}_j^1, \{\mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in S^1}, m, \sigma_{\mathsf{agg}}^1\right) = 1\right] = 1.$

We next define security of our new primitive, requiring that for a collusion of parties, it is hard to return aggregate signatures that they could not compute using the algorithms of the scheme. We require this to hold even if the collusion sees (layer-0 and/or layer-1) signatures by honest parties.

Definition 7 (Unforgeability)**.** *Unforgeability consists of two parts:*

Layer 0. *This notion prevents an adversary from forging a layer-0 aggregate signature without having seen valid signatures from all honest parties in the aggregate. This implies the standard unforgeability notion (EUF-CMA) for the (non-aggregate) signature scheme* $(\mathsf{KeyGen}^0, \mathsf{Sign}^0, \mathsf{Verify}^0)$*. Consider the following game, which maintains a list* Q_i *of queried messages for every honest user* i*, played by a PPT adversary* $\mathcal{A}$*:*

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(n_0, \mathcal{C}_0 \subseteq [n_0]) \leftarrow \mathcal{A}(\mathsf{pp})$
3. $\{(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}^0(\mathsf{pp})\}_{i\in[n_0]\setminus\mathcal{C}_0}$
4. $\{\mathsf{pk}_i^0\}_{i\in\mathcal{C}_0} \leftarrow \mathcal{A}(\{\mathsf{pk}_i^0\}_{i\in[n_0]\setminus\mathcal{C}_0})$
5. $\mathcal{A}$ *makes adaptive queries of the form*
   - (i, m)*: if* $i \in [n_0] \setminus \mathcal{C}_0$*, set* $Q_i := Q_i \cup \{m\}$*, run* $\sigma \leftarrow \mathsf{Sign}^0(\mathsf{sk}_i^0, m)$ *and return* σ*.*
6. $\mathcal{A}$ *outputs a tuple* $\left(m^*, S^* \subseteq [n_0], \sigma_{\mathsf{agg}}^*\right)$*.*

Then we demand that

$$\Pr\left[\mathsf{AggVerify}^0(\{\mathsf{pk}_i^0\}_{i\in S^*}, m^*, \sigma_{\mathsf{agg}}^*) = 1 \ \wedge \ \exists\, i \in S^* \setminus \mathcal{C}_0 : m^* \notin Q_i\right] \leq \mathsf{negl}(\lambda).$$

Layer 1. *This notion prevents an adversary from forging a layer-1 aggregate signature without having seen valid layer-0 signatures from all honest parties in the aggregate, even if the adversary has seen such layer-0 signatures attested to in layer-1 signatures by honest parties. We assume that for every message* m*, honest parties only attest to exactly one set of layer-0 signatures. The notion moreover prevents an adversary from changing honest party attestations.*

Consider the following game played by a (stateful) PPT adversary $\mathcal{A}$*. The game maintains lists* Q_i^0 *for all honest layer-0 users storing the signed messages, as well as an associative map* $Q_j^1 : \mathcal{M} \to \{0,1\}^n$ *for each honest layer-1 party* $j \in [n_1]\setminus\mathcal{C}_1$*, which maps a message* m *to the subset of* $[n_0]$ *that represents the layer-0 signatures for* m *that user* j *has attested to.*

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2. $(n_0, \mathcal{C}_0 \subseteq [n_0]) \leftarrow \mathcal{A}(\mathsf{pp})$
3. $\{(\mathsf{pk}_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{KeyGen}^0(\mathsf{pp})\}_{i\in[n_0]\setminus\mathcal{C}_0}$
4. $(\{\mathsf{pk}_i^0\}_{i\in\mathcal{C}_0}, n_1, \mathcal{C}_1 \subseteq [n_1]) \leftarrow \mathcal{A}(\{\mathsf{pk}_i^0\}_{[n_0]\setminus\mathcal{C}_0})$
5. $\{(\mathsf{pk}_j^1, \mathsf{sk}_j^1) \leftarrow \mathsf{KeyGen}^1(\mathsf{pp}, \{\mathsf{pk}_i^0\}_{i\in[n_0]})\}_{j\in[n_1]\setminus\mathcal{C}_1}$*,*
6. $\{\mathsf{pk}_j^1\}_{j\in\mathcal{C}_1} \leftarrow \mathcal{A}(\{\mathsf{pk}_j^1\}_{j\in[n_1]\setminus\mathcal{C}_1})$

7. $\mathcal{A}$ adaptively *makes queries of the form:*

- (Sign^0, i, m)*: if* $i \in [n_0] \setminus \mathcal{C}_0$*, set* $Q_i^0 := Q_i^0 \cup \{m\}$*, compute* $\sigma \leftarrow \mathsf{Sign}^0(\mathsf{sk}_i^0, m)$ *and return* σ*; else return* $\perp$*.*
- $(\mathsf{Sign}^1, j, S^0 \subseteq [n_0], m, \{\sigma_i^0\}_{i \in S^0 \cap \mathcal{C}_0})$*: if* $j \in [n_1] \setminus \mathcal{C}_1$*, then*
  - *If* $m \notin \mathsf{dom}(Q_j^1)$*:* $Q_j^1[m] \leftarrow S^0$*;* $\{\sigma_i^0 \leftarrow \mathsf{Sign}^0(\mathsf{sk}_i^0, m)\}_{i \in S^0 \setminus \mathcal{C}_0}$*;*
    return $\mathsf{Sign}^1(\mathsf{sk}_j^1, \{\mathsf{pk}_i^0, \sigma_i^0\}_{i \in S^0}, m)$*.*
  - *Else, return* $\perp$*.*

8. $\mathcal{A}$ *outputs a tuple* $(m^*, (S^1)^*, \{S_j^* \subseteq [n_0]\}_{j \in (S^1)^*}, (\sigma_{\mathsf{agg}}^1)^*)$*.*

We define two events E_1 *and* E_2*.* E_1 *states that a corrupt party attested to a set* S_j^* *of layer-0 signatures that contains an honest user that was never queried to sign* m^* *(vote stealing attack).*

$$E_1 : \quad \exists\, j \in \mathcal{C}_1 : S_j^* \nsubseteq \mathcal{C}_0 \cup \{i \in [n_0] \setminus \mathcal{C}_0 \mid m^* \in Q_i^0\}$$

The second event states that in the forgery some honest party j *attested to some non-empty set* S_j^* *of layer-0 signatures, but* S_j^* *is different from the set* S^0 *queried to* Sign^1 *for* j *and* m^**:*

$$E_2 : \quad \exists\, j \in (S^1)^* \setminus \mathcal{C}_1 : S_j^* \neq \emptyset \wedge S_j^* \neq Q_j^1[m^*]$$

Then, we demand that

$$\Pr\left[\mathsf{AggVerify}^1(\{\mathsf{pk}_j^1, \{\mathsf{pk}_i^0\}_{i \in S_j^*}\}_{j \in (S^1)^*}, m^*, (\sigma_{\mathsf{agg}}^1)^*) = 1 \ \wedge \ (E_1 \vee E_2)\right] \leq \mathsf{negl}(\lambda).$$

Remark 8. *Note that one could consider a stronger definition where the adversary can ask layer-0 signatures of honest users to be produced but not revealed. The adversary can then specify* which *honest layer-0 signatures should be contained in a queried layer-1 signature. We opt for definitional simplicity and note that since the signatures in our construction are deterministic, our construction readily achieves the stronger definition.*

Remark 9. *We note that the layer-1 unforgeability definitions could be strengthened to allow the adversary to query multiple sets of layer-0 public keys to be attested to by the same honest layer-1 party for the same message. Our construction does not achieve the stronger definition but the weaker definition suffices for applications in consensus protocols, as honest users only vote/attest once per proposal. An efficient construction satisfying the stronger definition is an interesting open problem.*

5 Construction with Linear-Sized Public Keys

Our scheme is defined over a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ and uses two hash functions, $H_0 \colon \mathcal{M} \to \mathbb{G}_1$ and $H' \colon \{0,1\}^* \to \mathbb{Z}_p$, which we will model as random oracles in the security proofs.

Proof system. We furthermore use a zero-knowledge *proof system* that lets parties prove knowledge of their secret keys; that is, a proof of knowledge of a witness from $\{y \in \mathbb{Z}_p \mid Y = X^y\}$ defined by group elements X and Y. After a potential setup, $\mathsf{NIZK.Prove}((Y, X), y)$ returns a proof π, which is verified by $\mathsf{NIZK.Verify}((Y, X), \pi)$. We assume that the proof system is simulation-sound and straight-line extractable. This can for example be realized using a sigma protocol with straight-

line extraction [Fis05, CL24].

As the layer-0 scheme, we use the BLS signature scheme [BLS01], which can be aggregated by simple multiplication when parties have proved knowledge of their secret keys (in order to thwart rogue-key attacks) [Bol03].

Layer 0. The setup and layer-0 algorithms are defined as follows:

$\mathsf{Setup}(1^\lambda)$: generate a bilinear group $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ with p of bitlength λ, and select two hash functions, $H_0 \colon \mathcal{M} \to \mathbb{G}_1$, whose domain is the message space, and $H' \colon \mathbb{G}_2^* \times \mathbb{Z} \to \mathbb{Z}_p$.

$\mathsf{KeyGen}^0(\mathsf{pp})$:
- sample $x \leftarrow\!\!\$\ \mathbb{Z}_p$, and set $X := g_2^x$
- compute a proof of knowledge $\pi^0 \leftarrow \mathsf{NIZK.Prove}((X, g_2), x)$.
- return $\mathsf{pk}^0 := (X, \pi^0)$ and $\mathsf{sk}^0 := x$.

$\mathsf{Sign}^0(x, m)$: return $\sigma^0 := H_0(m)^x$

$\mathsf{Verify}^0((X, \pi), m, \sigma^0)$: return 1 iff $\mathsf{NIZK.Verify}((X, g_2), \pi^0) = 1$ and $e(\sigma^0, g_2) = e(H_0(m), X)$.

$\mathsf{Aggregate}^0(\{((X_i, \pi_i^0), \sigma_i^0)\}_{i \in S^0}, m)$: return $\overline{\sigma}^0 := \prod_{i \in S^0} \sigma_i^0$.

$\mathsf{AggVerify}^0(\{(X_i, \pi_i^0)\}_{i \in S^0}, m, \overline{\sigma}^0)$: return 1 iff
- $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi_i^0) = 1\}_{i \in S^0}$, and
- $e(\overline{\sigma}^0, g_2) = e(H_0(m), \prod_{i \in S^0} X_i)$.

Layer 1. The layer-1 algorithms additionally use the hash function H', and are defined as follows:

$\mathsf{KeyGen}^1(\mathsf{pp}, \{\mathsf{pk}_i^0 = (X_i, \pi_i^0)\}_{i \in [n_0]})$:
- check that $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi_i^0) = 1\}_{i \in [n_0]}$
- sample $\{y_i \leftarrow\!\!\$\ \mathbb{Z}_p\}_{i \in [n_0]}$, and set $\{Y_i := (X_i)^{y_i}\}_{i \in [n_0]}$
- compute proofs of knowledge $\{\pi_i^1 \leftarrow \mathsf{NIZK.Prove}((Y_i, X_i), y_i)\}_{i \in [n_0]}$
- return $\mathsf{pk}^1 := \{Y_i, \pi_i^1\}_{i \in [n_0]}$ and $\mathsf{sk}^1 := \{y_i\}_{i \in [n_0]}$.

$\mathsf{Sign}^1(\{y_i\}_{i \in [n_0]}, \{\mathsf{pk}_i^0, \sigma_i^0\}_{i \in S^0}, m)$:
- if $\{\mathsf{Verify}^0(\mathsf{pk}_i^0, m, \sigma_i^0) = 1\}_{i \in S^0}$, return $\sigma^1 := \prod_{i \in S^0} (\sigma_i^0)^{y_i}$.
- else, return $\perp$.

$\mathsf{Verify}^1(\mathsf{pk}^1, \{\mathsf{pk}_i^0\}_{i \in S^0}, m, \sigma^1)$:
- parse $\mathsf{pk}^1 = \{Y_i, \pi_i^1\}_{i \in S^0}$, $\{\mathsf{pk}_i^0 = \{X_i, \pi_i^0\}\}_{i \in S^0}$
- check that $\{\mathsf{NIZK.Verify}((Y_i, X_i), \pi_i^1) = 1\}_{i \in S^0}$
- check that $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi_i^0) = 1\}_{i \in S^0}$
- return 1 iff $e(\sigma^1, g_2) = e\big(H_0(m), \prod_{i \in S^0} Y_i\big)$.

$\mathsf{Aggregate}^1(\{\mathsf{pk}_j^1, \sigma_j^1, \{\mathsf{pk}_i^0\}_{i \in S_j^0}\}_{j \in S^1}, m)$: return $\overline{\sigma}^1 := \prod_{j \in S^1} (\sigma_j^1)^{H'(\mathsf{pk}_{S^1}^1, j)}$, where $\mathsf{pk}_{S^1}^1 := \{\mathsf{pk}_j^1\}_{j \in S^1}$.

$\mathsf{AggVerify}^1(\{\mathsf{pk}_j^1, \{\mathsf{pk}_i^0\}_{i \in S_j^0}\}_{j \in S^1}, m, \overline{\sigma}^1)$:
- For each $j \in S^1$:
  - parse $\mathsf{pk}_j^1 = \{Y_{j,i}, \pi_{j,i}^1\}_{i \in [n_0]}$, and $\{\mathsf{pk}_i^0 = \{X_i, \pi_i^0\}\}_{i \in S_j^0}$

– check that $\{\mathsf{NIZK.Verify}((Y_{j,i}, X_i), \pi^1_{j,i}) = 1\}_{i \in S^0_j}$
– check that $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi^0_i) = 1\}_{i \in S^0_j}$

return 1 iff $e(\overline{\sigma}^1, g_2) = e\big(H_0(m),\ \prod_{j \in S^1}(\prod_{i \in S^0_j} Y_{j,i})^{H'(\mathsf{pk}^1_{S^1}, j)}\big)$.

Remark 10. *In the verification algorithms above, proof verification does not depend on the message or the subset participating in signing, and therefore can be pre-processed in $O(n_0 \cdot n_1)$ time.*

5.1 Correctness

From $\mathsf{Aggregate}^1$ and Sign^1 and Sign^0, we get $\sigma^0_i = H_0(m)^{\mathsf{sk}^0_i}$,

$$\overline{\sigma}^1 = \prod_{j \in S^1} (\sigma^1_j)^{H'(\mathsf{pk}^1_{S^1}, j)} \qquad \sigma^1_j = \prod_{i \in S^0_j} (\sigma^0_i)^{y_j}$$

and thus

$$\begin{aligned} e\left(\overline{\sigma}^1, g_2\right) &= e\left(\prod_{j \in S^1} \left(\sigma^1_j\right)^{H'\left(\mathsf{pk}^1_{S^1}, j\right)}, g_2\right) = \prod_{j \in S^1} e\left(\prod_{i \in S^0_j} \left(\sigma^0_i\right)^{y_j H'\left(\mathsf{pk}^1_{S^1}, j\right)}, g_2\right) \\ &= \prod_{j \in S^1} e\left(\prod_{i \in S^0_j} \left(H_0\left(m\right)\right)^{\mathsf{sk}^0_i y_j H'\left(\mathsf{pk}^1_{S^1}, j\right)}, g_2\right) = e\left(H_0\left(m\right), \prod_{j \in S^1} \prod_{i \in S^0_j} \left(g_2^{\mathsf{sk}^0_i y_j}\right)^{H'\left(\mathsf{pk}^1_{S^1}, j\right)}\right) \\ &= e\left(H_0\left(m\right), \prod_{j \in S^1} \left(\prod_{i \in S^0_j} \left(Y_{j,i}\right)^{H'\left(\mathsf{pk}^1_{S^1}, j\right)}\right)\right) \end{aligned}$$

5.2 Security

For both layer-0 and layer-1 aggregate security, we assume that the co-CDH assumption holds.

Layer-0 security. BLS signatures are securely aggregatable when users prove knowledge of their secret key [Bol03, BGLS03, LOS+06]. As preparation for showing layer-1 security, we reduce security of layer-1 signatures to the co-CDH assumption (Definition 1).

Let $\mathcal{A}$ be an adversary against layer-0 aggregate unforgeability creating n_0 users, making at most q_s signing queries, and winning the game with probability ϵ. We describe a reduction that solves co-CDH with probability at least $\frac{1}{2.72 \cdot q_s \cdot n_0}\epsilon$.

Let (u_1, v, u_2) be the reduction's co-CDH challenge. Upon receiving $(n_0, \mathcal{C}_0)$ from the adversary $\mathcal{A}$, it picks a uniform $i^* \leftarrow\!\!\$\ [n_0] \setminus \mathcal{C}_0$, sets $X_{i^*} := u_2$, simulates a proof π_{i^*} of knowledge of the logarithm of X_{i^*}, and sets $\mathsf{pk}^0_{i^*} := (X_{i^*}, \pi_{i^*})$; for $i \in [n_0] \setminus (\mathcal{C}_0 \cup \{i^*\})$, it runs $(\mathsf{pk}^0_i, \mathsf{sk}^0_i) \leftarrow \mathsf{KeyGen}^0(\mathsf{pp})$ and gives $\{\mathsf{pk}^0_i\}_{i \in [n_0] \setminus \mathcal{C}_0}$ to $\mathcal{A}$. Upon receiving $\{(X_i, \pi_i)\}_{i \in \mathcal{C}_0}$, the reduction extracts the secret keys $\mathsf{sk}^0_i = x_i$ from π_i for all i.

Random oracle query. Answer the j-th query for a message m_j as follows: Choose $\mu_j \leftarrow\!\!\$\ \mathbb{Z}_p$ and a bit b_j with $\Pr[b_j = 1] = 1/q_s$.

If $b_j = 1$ then return v^{μ_j}; otherwise return $g_1^{\mu_j}$.

Signing query for message m. Queries for users $i \in [n_0] \setminus (\mathcal{C}_0 \cup \{i^*\})$ are answered by returning $\mathsf{Sign}^1(\mathsf{sk}_i^0, m)$; queries for i^* are answered as follows: let j be the index of the random-oracle query for m (if none had been made, the reduction makes it now). If $b_j = 1$ then abort. Otherwise return $\sigma^0 := u_1^{\mu_j}$.

If the reduction does not abort, the game is perfectly simulated. Consider $\mathcal{A}$'s output $(m^*, S^*, \overline{\sigma}^*)$. If $i^* \notin S^*$ then abort. Let j^* be the index of the random-oracle query for m^*. If $b_{j^*} = 0$ then abort.

If the reduction did not abort and the adversary wins, $S^* \not\subseteq \mathcal{C}_0$ and from $\mathsf{AggVerify}^0(\{(X_i, \pi_i)\}_{i \in S^*}, m^*, \overline{\sigma}^*) = 1$ we have

$$\begin{aligned} e\big(\overline{\sigma}^*, g_2\big) = e\big(H_0(m^*), \textstyle\prod_{i \in S^*} X_i\big) &= e\big(v^{\mu_{j^*}}, u_2 \cdot \textstyle\prod_{i \in S^* \setminus \{i^*\}} g_2^{x_i}\big) \\ &= \big(e(v, u_2) \cdot e(v^{\sum_{i \in S^* \setminus \{i^*\}} x_i}, g_2)\big)^{\mu_{j^*}} \end{aligned}$$

and thus $e\big(w^*, g_2\big) = e(v, u_2)$ for $w^* := (\overline{\sigma}^*)^{\mu_{j^*}^{-1}} \cdot v^{-\sum_{i \in S^* \setminus \{i^*\}} x_i}$. The reduction can thus return w^* as the co-CDH solution.

It remains to lower-bound the probability that the reduction does not abort. It answers all signing queries with probability at least $(1 - 1/q_s)^{q_s} \geq 1/e > 1/2.72$. We have $b_{j^*} = 1$ with probability $1/q_s$ and the probability that $i^* \in S^*$ (which must contain an honest user) is at least $1/n_0$. All these probabilities are mutually independent and independent of the adversary's view. The simulation by the reduction thus succeeds with probability at least $\frac{1}{2.72 \cdot q_s \cdot n_0}$, which proves the statement.

Layer-1 security

Theorem 11. *The construction in Section 5 satisfies layer-1 aggregate security under the co-CDH assumption in the random oracle model.*

Proof. Let $\mathcal{A}$ be an adversary that after making at most q_0 queries for H_0 and q' queries for H' wins the layer-1 unforgeability game (Definition 7) with probability at least ϵ. Let m^* denote the message of the forgery. We define a reduction that receives a co-CDH challenge (u_1, v, u_2) and (for simplicity) guesses a query to H_0, in which it embeds v, hoping that $H_0(m^*) = v$; other queries are answered with $H_0(m_k) = g_1^{\mu_k}$ for $\mu_k \leftarrow\!\!\$\, \mathbb{Z}_p$. (Note that this strategy would make the number q_0 of queries for H_0 a factor in the security loss; however, embedding v in many random oracle responses works analogously to the proof of layer-0 security and reduces the loss from q_0 to $2.72 \cdot q_s$.) The element $u_2 = g_2^{\alpha}$ is embedded depending on the type of forgery, of which we distinguish the following:

Event E_1 occurs. The reduction guesses an honest user $i^* \notin \mathcal{C}_0$ such that $i^* \in S_j^*$ and $m^* \notin Q_{i^*}^0$. By E_1, such i^* exists. If the guess is incorrect, the reduction aborts. The challenge element $u_2 = g_2^{\alpha}$ is set as user i^*'s public key X_{i^*}, and the challenger simulates the proof π_{i^*}. The keys Y_{j,i^*} of layer-1 honest users are set as $Y_{j,i^*} := g_2^{\psi_j}$ for $\psi_j \leftarrow\!\!\$\, \mathbb{Z}_p$, implicitly defining $y_{j,i^*} = \psi_j/\alpha$, and the corresponding proofs π_{j,i^*}^1 are also simulated. (Note that this makes use of the fact that $g^{y_{j,i}}$ is never made public in the scheme.)

By assumption there is no query $(\mathsf{Sign}^0, i^*, m^*)$, so all layer-0 signing queries can be simulated: for i^* and $m \neq m^*$ with $H_0(m) = g_1^{\mu_k}$, compute $\sigma = u_1^{\mu_k}$; for other users use their secret keys.

All layer-1 signing queries $(\mathsf{Sign}^1, j, S^0, m, \{\sigma_i\}_{i \in S^0 \cap \mathcal{C}_0})$ with $i^* \in S^0$ can also be simulated by

computing $\sigma_i := H_0(m)^{x_i}$ for $i \in S^0 \setminus (\mathcal{C}_0 \cup \{i^*\})$ and returning

$$\big(\textstyle\prod_{i \in S^0 \setminus \{i^*\}} \sigma_i^{y_{j,i}}\big) \cdot H_0(m)^{\psi_j} \,.$$

(which also works for $m = m^*$). Queries Sign^1 with $i^* \notin S^0$ are answered honestly.

The reduction extracts all malicious keys $\{x_i\}_{i \in \mathcal{C}_0}$ and $\{y_{j,i}\}_{j \in \mathcal{C}_1, i \in [n_0]}$ from the proofs of knowledge.[4]. Let $(\overline{\sigma}^1)^*$ be the adversary's forgery and assume that the reduction guessed correctly; *i.e.*, for $J^* := \{j \in [n_1] \,|\, i^* \in S_j^0\}$, we have that $J_{\mathcal{C}}^* := J^* \cap \mathcal{C}_1$ is nonempty. Let $J_{\mathcal{H}}^* := J^* \setminus J_{\mathcal{C}}^*$ be the honest indices. From $\mathsf{AggVerify}^1$, we get (letting $\mathsf{pk}^1_{(S^1)^*}$ denote $\{\mathsf{pk}^1_k\}_{k \in (S^1)^*}$)

$$(\overline{\sigma}^1)^* = H_0(m^*)^{\gamma} \tag{3}$$

with

$$\begin{aligned}
\gamma &= \textstyle\sum_{j \in (S^1)^*} \sum_{i \in S_j^*} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) \\
&= \sum_{j \in (S^1)^* \setminus (J_{\mathcal{C}}^* \cup J_{\mathcal{H}}^*)} \sum_{i \in S_j^*} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) \\
&\quad + \sum_{j \in J_{\mathcal{C}}^*} \Big(\sum_{i \in S_j^* \setminus \{i^*\}} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) + \alpha y_{j,i^*} H'(\mathsf{pk}^1_{(S^1)^*}, j) \Big) \\
&\quad + \sum_{j \in J_{\mathcal{H}}^*} \Big(\sum_{i \in S_j^* \setminus \{i^*\}} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) + \psi_j H'(\mathsf{pk}^1_{(S^1)^*}, j) \Big) \,.
\end{aligned}$$

If

$$\sum_{j \in J_{\mathcal{C}}^*} y_{j,i^*} H'(\mathsf{pk}^1_{(S^1)^*}, j) \neq 0 \,, \tag{4}$$

then, using knowledge of all the other values contained in γ, the reduction can compute the co-CDH solution $H_0(m^*)^{\alpha} = v^{\alpha}$.

The probability that the adversary finds keys $\mathsf{pk}^1_{(S^1)^*}$, with corresponding secret keys $y_{j,i}$ and a corresponding set $(S^1)^*$ so that (4) does not hold is bounded by q'/p (where q' is a bound on the queries to H'). Since the reduction solves co-CDH if it guesses correctly (with probability $\frac{1}{q_0 \cdot n_0}$) and if Equation (4) holds, the reduction succeeds with probability at least $\frac{1}{q_0 \cdot n_0} \cdot \epsilon - \frac{q'}{p}$.

Event E_2 occurs and $S_j^* \setminus Q_j^1[m^*] \neq \emptyset$. This means there is an honest user j and some i^* which (for m^*) user j did not attest to ($i^* \notin Q_j^1[m^*]$) but the forgery says it did ($i^* \in S_j^*$). The reduction tries to guess both this $j^* \notin \mathcal{C}_1$ and $i^* \in [n_0]$. It sets $Y_{j^*,i^*} := u_1$ (implicitly setting $y_{j^*,i^*} := \alpha / x_{i^*}$), simulates the proof $\pi^1_{j^*,i^*}$ and computes all other keys as prescribed.

As before, the reduction also answers a randomly picked query to H_0 with v. If the query does not correspond to m^*, or if the guesses for j^* and i^* were not correct, the reduction aborts. If all guesses are correct, the reduction can answer all Sign^0 and Sign^1 queries honestly, since $i^* \notin Q_{j^*}^1[m^*]$ and thus y_{j^*,i^*} is never used.

[4]Since the reduction simulates the layer-1 signing queries to j with $i^* \in S^0$ and extracts the malicious keys from the proofs of knowledge, we require the proof of knowledge scheme to be straight-line simulation-extractable. This is also the case in Section 6

Consider the adversary's forgery for which we have $j^* \in (S^1)^*$ and $i^* \in S^*_{j^*}$. From $\mathsf{AggVerify}^1$, we again have (3), this time with

$$\gamma = \sum\nolimits_{j \in (S^1)^* \setminus \{j^*\}} \sum\nolimits_{i \in S^*_j} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) + \sum\nolimits_{i \in S^*_{j^*} \setminus \{i^*\}} x_i y_{j^*,i} H'(\mathsf{pk}^1_{(S^1)^*}, j^*) + \alpha H'(\mathsf{pk}^1_{(S^1)^*}, j^*)$$

With probability $1 - q'/p$, we have $H'(\mathsf{pk}^1_{(S^1)^*}, j^*) \neq 0$, in which case the reduction can compute the co-CDH solution $H_0(m^*)^\alpha$. The reduction succeeds thus with probability at least $\frac{1}{q_0 \cdot n_0 \cdot n_1} \cdot \epsilon - \frac{q'}{p}$.

Event E_2 occurs and $S^*_j \setminus Q^1_j[m^*] = \emptyset$. (Now the adversary *removed* an index from j's attestation.) In addition to the honest user j^*, now the reduction guesses two indices $i^*, i^- \in [n_0]$. Since by E_2 and $S^*_j \setminus Q^1_j[m^*] = \emptyset$, we have $\emptyset \neq S^*_j \subsetneq Q^1_{j^*}[m^*]$, there exist $i^*, i^- \in Q^1_{j^*}[m^*]$ with $i^* \in S^*_j$ and $i^- \notin S^*_j$. If the guessed values do not satisfy this, the reduction aborts.

The reduction chooses $\psi \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $Y_{j^*,i^*} := u_2$ and $Y_{j^*,i^-} := g_2^\psi / u_2$ and simulates the proofs $\pi^1_{j^*,i^*}$ and $\pi^1_{j^*,i^-}$. This implicitly sets $y_{j^*,i^*} = \alpha / x_{i^*}$ and $y_{j^*,i^-} = (\psi - \alpha)/x_{i^-}$. We thus have

$$x_{i^*} y_{j^*,i^*} + x_{i^-} y_{j^*,i^-} = \psi \ . \tag{5}$$

As before, the reduction also answers a randomly picked query to H_0 with v. If the query does not correspond to m^* or if the guesses for j^*, i^* and i^- were not correct, the reduction aborts. If all guesses are correct, the reduction can answer all Sign^0 queries and Sign^1 queries for users $j \neq j^*$ (since it knows all keys). It can also answer Sign^1 queries for j^* for $m \neq m^*$ (since it knows μ s.t. $H_0(m) = g_1^\mu$). For the query $(\mathsf{Sign}^1, j^*, S^0, m^*, \{\sigma^0_i\}_{i \in S^0 \cap \mathcal{C}_0})$, we have $S^0 = Q^1_{j^*}[m^*] \ni i^*, i^-$. The reduction computes $\sigma_i := H_0(m)^{y_{j,i}}$ for $i \in S^0 \setminus (\mathcal{C}_0 \cup \{i^*, i^-\})$ and returns

$$\left(\prod\nolimits_{i \in S^0 \setminus \{i^*, i^-\}} \sigma_i^{y_{j,i}}\right) \cdot H_0(m^*)^\psi = \left(\prod\nolimits_{i \in S^0 \setminus \{i^*, i^-\}} \sigma_i^{y_{j,i}}\right) \cdot v^\psi \ ,$$

which, by (5) is the correct signature.

Consider the adversary's forgery, for which we have $j^* \in (S^1)^*$, $i^* \in S^*_{j^*}$ and $i^- \notin S^*_{j^*}$. From $\mathsf{AggVerify}^1$ we again have (3) with

$$\gamma = \sum\nolimits_{j \in (S^1)^* \setminus \{j^*\}} \sum\nolimits_{i \in S^*_j} x_i y_{j,i} H'(\mathsf{pk}^1_{(S^1)^*}, j) + \sum\nolimits_{i \in S^*_{j^*} \setminus \{i^*\}} x_i y_{j^*,i} H'(\mathsf{pk}^1_{(S^1)^*}, j^*) + \alpha H'(\mathsf{pk}^1_{(S^1)^*}, j^*)$$

With probability $1 - q'/p$, we have $H'(\mathsf{pk}^1_{(S^1)^*}, j^*) \neq 0$, in which case the reduction can compute the co-CDH solution $H_0(m^*)^\alpha$. It thus succeeds with probability at least $\frac{1}{q_0 \cdot n_0^2 \cdot n_1} \cdot \epsilon - \frac{q'}{p}$. □

5.3 Reducing the Complexity of Aggregation and Verification

Baldimtsi et al. [BCG+24] propose making BLS-aggregation using random coefficients [BDN18] more efficient when there is a fixed set of public keys. This can, for example, be assumed during a slot of proof-of-stake protocols.

Let $\{\mathsf{pk}_i\}_{i \in [n]}$ be the set of signers. Then, instead of aggregating the signatures $\{\sigma_j\}_{j \in S}$ of a set of signers $S \subseteq [n]$ as $\overline{\sigma} := \prod_{j \in S} \sigma_j^{H'(\mathsf{pk}_S, j)}$, it is defined as $\overline{\sigma} := \prod_{j \in S} \sigma_j^{H'(\mathsf{pk}_{[n]}, j)}$, that is, the exponents are independent of the subset.

The authors [BCG+24] show that, without assuming proofs of knowledge of secret keys, this variant can be shown secure in the algebraic group model [FKL18] via a loose reduction (due to rewinding of the adversary) and additionally assuming hardness of the random modular subset sum problem (RMSS, Def. 3).

Modifying our scheme. We can modify our layer-1 scheme analogously; that is, $\mathsf{Aggregate}^1$ now takes all ("fixed") public keys $\mathsf{PK} := \big(\{\mathsf{pk}_i^0\}_{i\in[n_0]}, \{\mathsf{pk}_j^1\}_{j\in[n_1]}\big)$ as input and is modified as follows:

$\mathsf{Aggregate}^1(\mathsf{PK}, \{\sigma_j^1\}_{j\in S^1}, m)$: return $\overline{\sigma}^1 := \prod_{j\in S^1}(\sigma_j^1)^{H'(\mathsf{PK},j)}$,

and $\mathsf{AggVerify}^1$ is adapted accordingly.

Security. To prove security, we additionally assume hardness of RMSS. Inspection of the proof of Theorem 11 shows that the arguments for event E_2 are unchanged, since the probability of finding a preimage of 0 under H' is still bounded by q'/p.

For event E_1, we need to bound the probability that (4) is violated, which now becomes

$$\sum_{j\in J_{\mathcal{C}}^*} y_{j,i^*} H'(\mathsf{PK}, j) \neq 0\ . \tag{6}$$

Leveraging the fact that the reduction knows the adversary's secret keys $\{y_{j,i^*}\}_{i\in\mathcal{C}_1}$, we can (without resorting to either the AGM or rewinding) use an adversary that violates (6) to solve RMSS: the reduction receives an RMSS instance $(\{s_j\}_{j\in[n_1]}, t)$, guesses a random user $j^* \leftarrow\!\!\$\ \mathcal{C}_1$ and answers queries to H' as follows: for $j \neq j^*$, set $H'(\mathsf{PK}, j) := y_{j,i^*}^{-1} \cdot s_j$ and set $H'(\mathsf{PK}, j^*) := -y_{j^*,i^*}^{-1} \cdot t$. If the guess was correct and $j^* \in J_{\mathcal{C}}^*$, and if (6) is violated, then $J := J_{\mathcal{C}}^* \setminus \{j^*\}$ is a solution to RMSS since

$$0 = \sum_{j\in J_{\mathcal{C}}^*\setminus\{j^*\}} y_{j,i^*} H'(\mathsf{PK}, j) + y_{j^*,i^*} H'(\mathsf{PK}, j^*) = \sum_{j\in J} s_j - t\ .$$

Improving efficiency. Having made the "coefficients" $H'(\dots)$ independent of the subset being aggregated now allows us to improve the efficiency of the scheme following Baldimtsi et al. [BCG+24]:

- Once the proofs contained in the keys are verified, there is a re-randomization phase setting layer-1 public keys to $\hat{Y}_{j,i} := Y_{j,i}^{H'(\mathsf{PK},j)}$ and secret keys to $\hat{y}_{j,i} := y_{j,i} \cdot H'(\mathsf{PK}, j)$.
- Sign^1 is run with $\{\hat{y}_i\}_{i\in S^0}$ instead of $\{y_i\}_{i\in S^0}$, returning $\hat{\sigma}^1 := \prod_{i\in S^0}(\sigma_i^0)^{\hat{y}_i}$ (Verify^1 is unchanged).
- $\mathsf{Aggregate}^1(\mathsf{PK}, \{\sigma_i^1\}_{i\in S^1}, m)$ now simply returns $\overline{\sigma}^1 := \prod_{j\in S^1} \hat{\sigma}_j^1$.
- $\mathsf{AggVerify}^1$ simply checks that $e\big(\overline{\sigma}^1, g_2\big) = e\big(H_0(m), \prod_{j\in S^1}(\prod_{i\in S_j^0} \hat{Y}_{j,i})\big)$

Therefore, in both $\mathsf{Aggregate}^1$ and $\mathsf{AggVerify}^1$, we save $O(n_1)$ $\mathbb{G}_1$ multi-scalar multiplications, making the total cost of $\mathsf{Aggregate}^1$ $O(n_1)$ $\mathbb{G}_1$ operations, and the total cost of $\mathsf{AggVerify}^1$, $O(n_0 n_1)$ $\mathbb{G}_1$ operations.

6 Construction with Constant-sized Public Keys

We now present an alternative construction, where the layer-1 public keys are constant-sized and can be generated independently from layer-0 public keys (instead of public keys of size $O(n_0)$ in the previous construction). On the other hand, the layer-1 signatures contain an extra $\mathbb{G}_2$ element, and verifying the layer-1 aggregate signature requires $O(n)$ pairings (instead of $O(1)$) in addition to

$O(n^2)$ base group operations. Security now follows from the (2,1)-CDH assumption (Definition 2) in the random oracle model.

In Appendix 8.1, we show how the Inner Pairing Product proof system by Bünz et al. [BMM+21, Appendix B] can be used to reduce the verifier complexity, at the expense of a constant-factor increase in the aggregator's work (from $O(n^2)$ base group operations and $O(n)$ pairings to $O(n^2)$ base group operations, $O(\log(n))$ target-group exponentiations and $O(1)$ source-group MSMs and pairings).

Layer 0. As in our previous construction in Section 5, the layer-0 algorithms are standard BLS multi-signatures.

Layer 1. The scheme uses three hash functions, $H_0 \colon \mathcal{M} \to \mathbb{G}_1$, $H' \colon \{0,1\}^* \to \mathbb{Z}_p$ and $H_1 \colon \{0,1\}^* \to \mathbb{G}_1$, which we model as random oracles. Layer-1 algorithms are defined as follows:

$\mathsf{Setup}(1^\lambda)$**:** generate a bilinear group $\mathsf{pp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ with p of bitlength λ, and select hash functions $H_0 \colon \mathcal{M} \to \mathbb{G}_1$, taking input messages from the message space $\mathcal{M}$, $H' \colon \mathbb{G}_2^* \times \mathbb{Z} \to \mathbb{Z}_p$, and $H_1 \colon \{0,1\}^* \to \mathbb{G}_1$.

$\mathsf{KeyGen}^1(\mathsf{pp})$**:**
- sample $y, z \leftarrow\!\!\$\ \mathbb{Z}_p$, and set $Y := g_1^y$ and $Z := g_2^z$
- compute a proof of knowledge $\pi^y \leftarrow \mathsf{NIZK.Prove}((Y, g_1), y)$ and $\pi^z \leftarrow \mathsf{NIZK.Prove}((Z, g_2), z)$
- return $\mathsf{pk}^1 := (Y, Z, \pi^y, \pi^z)$ and $\mathsf{sk}^1 := (y, z)$.

$\mathsf{Sign}^1((y,z), \{\mathsf{pk}_i^0, \sigma_i^0\}_{i \in S^0}, m)$**:** If $\{\mathsf{Verify}^0(\mathsf{pk}_i^0, m, \sigma_i^0) = 1\}_{i \in S_0}$, then return $\sigma^1 := (\sigma^{1,1}, \sigma^{1,2})$ with $\sigma^{1,1} := H_1(m, S^0, \mathsf{pk}^1)^z \cdot \prod_{i \in S^0} (\sigma_i^0)^y$ and $\sigma^{1,2} := \prod_{i \in S^0} (X_i)^y$. Else, return $\perp$.

$\mathsf{Verify}^1(\mathsf{pk}^1, \{\mathsf{pk}_i^0\}_{i \in S^0}, m, \sigma^1)$**:**
- parse $\mathsf{pk}^1 = (Y, Z, \pi^y, \pi^z)$, $\{\mathsf{pk}_i^0 = \{X_i, \pi_i^0\}\}_{i \in S^0}$, and $\sigma^1 := (\sigma^{1,1}, \sigma^{1,2})$
- check that $\mathsf{NIZK.Verify}((Y, g_1), \pi^y) = 1$
- check that $\mathsf{NIZK.Verify}((Z, g_2), \pi^z) = 1$
- check that $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi_i^0) = 1\}_{i \in S^0}$
- return 1 iff

$$e(\sigma^{1,1}, g_2) = e(H_0(m), \sigma^{1,2}) \cdot e(H_1(m, S^0, \mathsf{pk}^1), Z),$$
$$e(g_1, \sigma^{1,2}) = e(Y, \textstyle\prod_{i \in S^0} X_i).$$

$\mathsf{Aggregate}^1(\{\mathsf{pk}_j^1, \sigma_j^1, \{\mathsf{pk}_i^0\}_{i \in S_j^0}\}_{j \in S^1}, m)$**:** Let $\mathsf{pk}_{S^1}^1 := \{\mathsf{pk}_j^1\}_{j \in S^1}$.

- parse $\sigma_j^1 = (\sigma_j^{1,1}, \sigma_j^{1,2})$
- compute $\overline{\sigma}^{1,1} := \prod_{j \in S^1} (\sigma_j^{1,1})^{H'(\mathsf{pk}_{S^1}^1, j)}$ and
  $\overline{\sigma}^{1,2} := \prod_{j \in S^1} (\sigma_j^{1,2})^{H'(\mathsf{pk}_{S^1}^1, j)}$,
- return $\overline{\sigma}^1 := (\overline{\sigma}^{1,1}, \overline{\sigma}^{1,2})$.

$\mathsf{AggVerify}^1(\{\mathsf{pk}_j^1, \{\mathsf{pk}_i^0\}_{i \in S_j^0}\}_{j \in S^1}, m, \overline{\sigma}^1)$**:**
- For each $j \in S^1$
  - parse $\mathsf{pk}_j^1 = (Y_j, Z_j, \pi_j^y, \pi_j^z)$, and $\{\mathsf{pk}_i^0 = \{X_i, \pi_i^0\}\}_{i \in S_j^0}$

- check that $\mathsf{NIZK.Verify}((Y_j, g_1), \pi_j^y) = 1$
- check that $\mathsf{NIZK.Verify}((Z_j, g_2), \pi_j^z) = 1$
- check that $\{\mathsf{NIZK.Verify}((X_i, g_2), \pi_i^0) = 1\}_{i \in S_j^0}$

• return 1 iff the following hold

$$e\big(\overline{\sigma}^{1,1}, g_2\big) \;=\; e\big(H_0(m), \overline{\sigma}^{1,2}\big) \cdot \prod_{j \in S^1} e\Big(H_1(m, S_j^0, \mathsf{pk}_j^1)^{H'(\mathsf{pk}_{S^1}^1, j)},\; Z_j\Big)$$

$$e\big(g_1, \overline{\sigma}^{1,2}\big) \;=\; \textstyle\prod_{j \in S^1} e\Big((Y_j)^{H'(\mathsf{pk}_{S^1}^1, j)}, \prod_{i \in S_j^0} X_i\Big).$$

6.1 Correctness

From $\mathsf{Aggregate}^1$ and Sign^1 and Sign^0, we get $\sigma_i^0 = H_0(m)^{\mathsf{sk}_i^0}$,

$$\overline{\sigma}^{1,1} = \prod_{j \in S^1} (\sigma_j^{1,1})^{H'(\mathsf{pk}_{S^1}^1, j)} \qquad \overline{\sigma}^{1,2} = \prod_{j \in S^1} (\sigma_j^{1,2})^{H'(\mathsf{pk}_{S^1}^1, j)}$$

$$\sigma_j^{1,1} = \prod_{i \in S_j^0} (\sigma_i^0)^{y_j} \cdot H_1(m, S_j^0, \mathsf{pk}_j^1)^{z_j} \qquad \sigma_j^{1,2} = \prod_{i \in S_j^0} (\mathsf{pk}_i^0)^{y_j},$$

and thus

$$\begin{aligned}
e\left(\overline{\sigma}^{1,1}, g_2\right) &= e\left(\prod_{j \in S^1} \left(\sigma_j^{1,1}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}, g_2\right) = \prod_{j \in S^1} e\left(\left(\prod_{i \in S_j^0} \left(\sigma_i^0\right)^{y_j} \cdot H_1\left(m, S_j^0, \mathsf{pk}_j^1\right)^{z_j}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}, g_2\right) \\
&= \prod_{j \in S^1} e\left(\prod_{i \in S_j^0} \left(H_0\left(m\right)\right)^{\mathsf{sk}_i^0 y_j H'\left(\mathsf{pk}_{S^1}^1, j\right)}, g_2\right) \cdot e\left(\left(H_1\left(m, S_j^0, \mathsf{pk}_j^1\right)^{z_j}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}, g_2\right) \\
&= \prod_{j \in S^1} e\left(H_0\left(m\right), \prod_{i \in S_j^0} \left(g_2^{\mathsf{sk}_i^0 y_j}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}\right) \cdot e\left(H_1\left(m, S_j^0, \mathsf{pk}_j^1\right), \left(g_2^{z_j}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}\right) \\
&= e\left(H_0\left(m\right), \prod_{j \in S^1} \prod_{i \in S_j^0} g_2^{H'\left(\mathsf{pk}_{S^1}^1, j\right)\mathsf{sk}_i^0 y_j}\right) \cdot \prod_{j \in S^1} e\left(H_1\left(m, S_j^0, \mathsf{pk}_j^1\right), \left(g_2^{z_j}\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}\right) \\
&= e\left(H_0\left(m\right), \overline{\sigma}^{1,2}\right) \cdot \prod_{j \in S^1} e\left(H_1\left(m, S_j^0, \mathsf{pk}_j^1\right), \left(Z_j\right)^{H'\left(\mathsf{pk}_{S^1}^1, j\right)}\right).
\end{aligned}$$

Correctness for

$$e\big(g_1, \overline{\sigma}^{1,2}\big) \;=\; \textstyle\prod_{j \in S^1} e\Big((Y_j)^{H'(\mathsf{pk}_{S^1}^1, j)}, \prod_{i \in S_j^0} X_i\Big)$$

holds similarly by inspection.

6.2 Security

The aggregate security for layer-0 follows from the same arguments as in Section 5, which in turn are based on [Bol03, BGLS03, LOS+06].

Theorem 12. *The construction in Section 6 satisfies layer-1 aggregate security under the (2,1)-CDH assumption in the random oracle model.*

Proof. Let $\mathcal{A}$ be an adversary that after making at most q_0 queries for H_0, q_1 queries for H_1, and q' queries for H', wins the layer-1 aggregate unforgeability game with probability at least ϵ. Security of our scheme relies on the (2,1)-CDH assumption (Definition 2, *cf.* Section 3).

We next consider two different types of forgery corresponding to the events E_1 and E_2 (Section 4):

Event E_1 occurs. We define a reduction that receives a $(2,1)$-CDH challenge $(u_1, u_1', u_2, u_2', v_1, v_1')$ and guesses a query to H_0, in which it embeds v_1, hoping that $H_0(m^*) = v_1 = g_1^{\beta}$. All other queries to H_0 are simulated by picking a random $\mu \leftarrow\!\!\$\ \mathbb{Z}_p$ and returning g_1^{μ}. The reduction first guesses an honest user $i^* \notin \mathcal{C}_0$ such that $i^* \in S_j^*$ and $m^* \notin Q_{i^*}^0$. By E_1, such a user i^* exists. If the guess turns out to be incorrect, the reduction aborts.

The challenge element $u_2 = g_2^{\alpha}$ is set as user i^*'s public key $\mathsf{pk}_{i^*}^0 := X_{i^*}$, and the challenger simulates the proof π_{i^*}. All other layer-0 public keys are generated by running KeyGen^0. The keys Y_j of the layer-1 honest users are set as $Y_j := (u_1')^{\psi_j}$ for random ψ_j, implicitly defining $y_j := \psi_j/\alpha$, and the corresponding proofs π_j^y are also simulated. The keys Z_j of the layer-1 honest users are set as $Z_j := (g_2)^{z_j}$ for random $z_j \leftarrow\!\!\$\ \mathbb{Z}_p$, and the corresponding proofs π_j^z are computed honestly using the NIZK algorithms.

Since by assumption there is no query $(\mathsf{Sign}^0, i^*, m^*)$, all layer-0 signing queries can be simulated: for i^* and $m \neq m^*$ with $H_0(m) = g_1^{\mu_k}$ for some k, compute $\sigma = u_1^{\mu_k}$; for all other users, use their secret keys.

All layer-1 signing queries $(\mathsf{Sign}^1, j, S^0, m, \{\sigma_i\}_{i \in S^0 \cap \mathcal{C}_0})$ with $i^* \in S^0$ can also be simulated by computing and returning

- $\sigma_j^{1,1} := \left(\prod_{i \in S^0 \setminus \{i^*\}} (v_1')^{x_i \psi_j}\right) \cdot (v_1)^{\psi_j} \cdot H_1(m, S^0, \mathsf{pk}_j^1)^{z_j}$ if $m = m^*$,
- $\sigma_j^{1,1} := \left(\prod_{i \in S^0 \setminus \{i^*\}} (u_1')^{x_i \psi_j \mu_k}\right) \cdot (g_1)^{\psi_j \mu_k} \cdot H_1(m, S^0, \mathsf{pk}_j^1)^{z_j}$ if $m \neq m^*$,
- $\sigma_j^{1,2} := \left(\prod_{i \in S^0 \setminus \{i^*\}} (u_2')^{x_i \psi_j}\right) \cdot (g_2)^{\psi_j}$.

Queries for Sign^1 with $i^* \notin S^0$ are answered as follows:

- $\sigma_j^{1,1} := \left(\prod_{i \in S^0} (v_1')^{x_i \psi_j}\right) \cdot H_1(m, S^0, \mathsf{pk}_j^1)^{z_j}$ if $m = m^*$,
- $\sigma_j^{1,1} := \left(\prod_{i \in S^0} (u_1')^{x_i \psi_j \mu_k}\right) \cdot H_1(m, S^0, \mathsf{pk}_j^1)^{z_j}$ if $m \neq m^*$,
- $\sigma_j^{1,2} := \left(\prod_{i \in S^0} (u_2')^{x_i \psi_j}\right)$.

The reduction extracts all malicious keys $\{x_i\}_{i \in \mathcal{C}_0}$, $\{y_j\}_{j \in \mathcal{C}_1}$, and $\{z_j\}_{j \in \mathcal{C}_1}$ from the proofs of knowledge, and knows implicitly that $y_j := \psi_j/\alpha$ for honest keys. Let $\overline{\sigma}^* = (\overline{\sigma}^{1,1}, \overline{\sigma}^{1,2})$ be the adversary's forgery and assume that the reduction guessed correctly; *i.e.*, for $J^* := \{j \in [n_1] \mid i^* \in S_j^0\}$, we have that $J_{\mathcal{C}}^* := J^* \cap \mathcal{C}_1$ is nonempty. Let $J_{\mathcal{H}}^* := J^* \setminus J_{\mathcal{C}}^*$. From $\mathsf{AggVerify}^1$, defining

$\mathsf{pk}^1_{(S^1)^*} := \{\mathsf{pk}^1_k\}_{k\in(S^1)^*}$, we obtain

$$\log(\overline{\sigma}^{1,1}) = (\beta/\alpha)\sum_{j\in\mathcal{H}_1}\sum_{i\in S_j\setminus\{i^*\}} x_i\psi_j H'(\mathsf{pk}^1_{(S^1)^*}, j) \; + \beta\sum_{j\in\mathcal{C}_1}\sum_{i\in S_j\setminus\{i^*\}} x_i y_j H'(\mathsf{pk}^1_{(S^1)^*}, j) +$$
$$\beta\sum_{j\in J^*_{\mathcal{H}}} \psi_j H'(\mathsf{pk}^1_{(S^1)^*}, j) + \alpha\beta\sum_{j\in J^*_{\mathcal{C}}} y_j H'(\mathsf{pk}^1_{(S^1)^*}, j) +$$
$$\sum_{j\in S^1} (\log H_1(m^*, S^0_j, \mathsf{pk}^1_j)) H'(\mathsf{pk}^1_{(S^1)^*}, j) z_j$$

If

$$\sum_{j\in J^*_{\mathcal{C}}} y_j H'(\mathsf{pk}^1_{(S^1)^*}, j) \neq 0\,, \tag{7}$$

then, using knowledge of all the other values contained above (as $v'_1 = g_1^{\beta/\alpha}$, $v_1 = g_1^{\beta}$, and the answers to $H'(\cdot)$ and $H_1(\cdot)$ are known), the reduction can compute $H_0(m^*)^\alpha = (g_1)^{\alpha\beta}$ and return it as the solution.

Let q' be the number of the adversary's random oracle queries for H'. Then the probability that the adversary finds keys $\mathsf{pk}^1_{(S^1)^*}$, with corresponding secret keys $y_{j,i}$ and a corresponding set $(S^1)^*$ so that (7) does not hold is bounded by q'/p. Therefore, as in Section 5, the reduction succeeds with probability at least $\frac{1}{q_0\cdot n_0}\cdot\epsilon - \frac{q'}{p}$.

Event E_2 occurs and $S^0_{j^*} \setminus Q^1_j[m^*] \neq \emptyset$. We define a reduction that receives a co-CDH challenge (u_1, v, u_2), and embeds v into queries to H_1. The reduction first guesses an honest user $j^* \notin \mathcal{C}_1$, hoping that it will be a user j with the properties defined in E_2. Moreover, it guesses $i^* \in [n_0]$, hoping that it will satisfy $i^* \in S^0_{j^*} \setminus Q^1_j[m^*]$ (in the case we are analyzing, such j^* and i^* exist). It sets $Z_{j^*} := u_2$, implicitly setting $z_{j^*} := \alpha$, simulates the proof $\pi^z_{j^*}$, and computes all other keys and random oracle queries to $H_0(\cdot)$ and $H'(\cdot)$ as prescribed. For each query to $H_1(m^*, S^0, \mathsf{pk}^1_{j^*})$, where $i^* \in S^0$, the reduction picks a new random value $\nu \leftarrow\!\!\$\, \mathbb{Z}_p$ and returns $g_1^\nu \cdot v$. For other queries to $H_1(\cdot)$, the reduction picks a random value $\nu \leftarrow\!\!\$\, \mathbb{Z}_p$ and returns g_1^ν.

If the guesses for m^*, j^* and i^* turn out to be incorrect, the reduction aborts. If all guesses are correct, the reduction can answer all Sign^0 queries and Sign^1 queries for users $j \neq j^*$ (since it knows all keys). As for the query $(\mathsf{Sign}^1, j^*, S^0, m^*, \{\sigma^0_i\}_{i\in S^0\cap\mathcal{C}_0})$, we have $i^* \notin S^0$, and thus the reduction does not have to compute $H_1(m^*, S^0_{j^*}, \mathsf{pk}^1_{j^*})^{z_{j^*}}$ with $i^* \in S^0_{j^*}$. Hence, the reduction can compute the correct signature using the secret key y_{j^*} of j^*, challenge value u_1, and the random value ν sampled for $H_1(m^*, S^0_{j^*}, \mathsf{pk}^1_{j^*})$:

$$\sigma^1_{j^*} := \textstyle\prod_{i\in S^0_{j^*}} (\sigma^0_i)^{y_{j^*}} \cdot (u_1)^\nu, \qquad\qquad \sigma^2 := (\textstyle\prod_{i\in S^0_{j^*}} \mathsf{pk}^0_i)^{y_{j^*}}.$$

Consider the adversary's forgery, for which we have $j^* \in (S^1)^*$ and $i^* \in S^0_{j^*}$. From $\mathsf{AggVerify}^1$ we have

$$\log(\overline{\sigma}^1) = \sum_{j\in(S^1)^*}\sum_{i\in S^0_j} (\log H_0(m^*)) H'(\mathsf{pk}^1_{(S^1)^*}, j) x_i y_j \; +$$
$$\sum_{j\in(S^1)^*\setminus\{j^*\}} (\log H_1(m^*, S^0_j, \mathsf{pk}^1_j)) H'(\mathsf{pk}^1_{(S^1)^*}, j) z_j + H'(\mathsf{pk}^1_{(S^1)^*}, j^*)\alpha(\nu+\beta).$$

Here, ν is known to the challenger from its simulation of $H_1(\cdot)$, and with probability $1 - q'/p$, we have $H'(\mathsf{pk}^1_{(S^1)^*}, j^*) \neq 0$, in which case the reduction can compute the co-CDH solution $g_1^{\alpha\beta}$. Thus, the reduction succeeds with probability at least $\frac{1}{q_0 \cdot n_0 \cdot n_1} \cdot \epsilon - \frac{q'}{p}$.

Event E_2 occurs and $S^0_{j^*} \setminus Q^1_j[m^*] = \emptyset$. In addition to the honest user j^* as in the previous case, now the reduction guesses two indices $i^*, i^- \in [n_0]$. By E_2 and $S^0_{j^*} \setminus Q^1_j[m^*] = \emptyset$, we have $\emptyset \neq S^0_{j^*} \subsetneq Q^1_{j^*}[m^*]$, and thus, there exist some $i^*, i^- \in Q^1_{j^*}[m^*]$ with $i^* \in S^0_{j^*}$ and $i^- \notin S^0_{j^*}$ (and the reduction hopes that it selected the correct indices). The reduction sets $Z_{j^*} := u_2$, implicitly setting $z_{j^*} := \alpha$, simulates the proof $\pi^z_{j^*}$, and computes all other keys and random oracle queries to $H_0(\cdot)$ and $H'(\cdot)$ as prescribed. For each query to $H_1(m^*, S^0_{j^*}, \mathsf{pk}^1_{j^*})$, where $i^* \in S^0_{j^*}$, but $i^- \notin S^0_{j^*}$, the reduction picks a new random value $\nu \leftarrow\!\!\$\ \mathbb{Z}_p$ and returns $g_1^\nu \cdot v$. For other queries to $H_1(\cdot)$, the reduction picks a random value $\nu \leftarrow\!\!\$\ \mathbb{Z}_p$ and returns g_1^ν.

If the guesses for m^*, j^*, i^* and i^- turn out to be incorrect, the reduction aborts. If all guesses are correct, the reduction can answer all Sign^0 queries and Sign^1 queries for users $j \neq j^*$ (since it knows all keys). As for the query $(\mathsf{Sign}^1, j^*, S^0, m^*, \{\sigma^0_i\}_{i \in S^0 \cap \mathcal{C}_0})$, we have $S^0 = Q^1_{j^*}[m^*] \ni i^*, i^-$. Thus, the reduction can compute the correct signature using the secret key y_{j^*} of j^*, challenge value u_1, and the random value ν sampled for $H_1(m^*, S^0_{j^*}, \mathsf{pk}^1_{j^*})$:

$$\sigma^1_{j^*} := \textstyle\prod_{i \in S^0_{j^*}} (\sigma^0_i)^{y_{j^*}} \cdot (u_1)^\nu, \qquad \sigma^2 := (\textstyle\prod_{i \in S^0_{j^*}} \mathsf{pk}^0_i)^{y_{j^*}}.$$

Consider the adversary's forgery, for which we have $j^* \in (S^1)^*$, $i^* \in S^{0,*}_{j^*}$ and $i^- \notin S^{0,*}_{j^*}$. From $\mathsf{AggVerify}^1$, we again have

$$\log(\overline{\sigma}^1) = \sum_{j \in (S^1)^*} \sum_{i \in S^0_j} (\log H_0(m^*)) H'(\mathsf{pk}^1_{(S^1)^*}, j) x_i y_j + \sum_{j \in (S^1)^* \setminus \{j^*\}} (\log H_1(m^*, S^0_j, \mathsf{pk}^1_j)) H'(\mathsf{pk}^1_{(S^1)^*}, j) z_j + H'(\mathsf{pk}^1_{(S^1)^*}, j) \alpha(\nu + \beta).$$

Here, ν is known to the challenger from its simulation of $H_1(\cdot)$, and with probability $1 - q'/p$, we have $H'(\mathsf{pk}^1_{(S^1)^*}, j^*) \neq 0$, in which case the reduction can compute the co-CDH solution $g_1^{\alpha\beta}$. Thus, the reduction succeeds with probability at least $\frac{1}{q_0 \cdot n_0^2 \cdot n_1} \cdot \epsilon - \frac{q'}{p}$.

□

Remark 13. *To prove E_2 security and prevent equivocation attacks, it is sufficient to use the bit map S^0_j rather than the public keys $\{\mathsf{pk}^0_i\}_{i \in S^0_j}$ in $H_1(\cdot)$. This is because the set of public keys is fixed and known in advance. When the set of layer-0 public keys is not known through a PKI, but given by the layer-1 signer, equivocation attacks can be prevented by requiring $H_1(\cdot)$ to be queried with the tuple $(m, \{\mathsf{pk}^0_i\}_{i \in S^0_j}, j)$.*

6.3 Reducing the Complexity of Aggregation and Verification

We can again apply the technique by Baldimtsi et al. [BCG+24] to reduce the complexity of aggregation and verification. As in Section 5.3, we modify $\mathsf{Aggregate}^1$ to take all public keys

$\mathsf{PK} := \left(\{\mathsf{pk}_i^0\}_{i\in[n_0]}, \{\mathsf{pk}_j^1\}_{j\in[n_1]}\right)$ as input; $\mathsf{AggVerify}^1$ is modified appropriately. Both algorithms are described below:

- After verifying the contained proofs, the layer-1 public keys are re-randomized to $\hat{Y}_j := Y_j^{H'(\mathsf{PK},j)}$ and $\hat{Z}_j := Z_j^{H'(\mathsf{PK},j)}$, and secret keys to $\hat{y}_j := y_j \cdot H'(\mathsf{PK}, j)$ and $\hat{z}_j := z_j \cdot H'(\mathsf{PK}, j)$.
- Sign^1 is run with $(\hat{y}_j, \hat{z}_j)$, returning $\hat{\sigma}^{1,1} := \prod_{i\in S_j^0}(\sigma_i^0)^{\hat{y}_j} \cdot H_1(m, S_j^0, \mathsf{pk}_j^1)^{\hat{z}_j}$, and $\hat{\sigma}^{1,2} := \prod_{i\in S_j^0}(X_i)^{\hat{y}_j}$.
- $\mathsf{Aggregate}^1(\mathsf{PK}, \{\sigma_j^1\}_{i\in S^1}, m)$ now simply returns $\overline{\sigma}^{1,1} := \prod_{j\in S^1} \hat{\sigma}_j^{1,1}$, and $\overline{\sigma}^{1,2} := \prod_{j\in S^1} \hat{\sigma}_j^{1,2}$.
- $\mathsf{AggVerify}^1$ now returns 1 if

$$e\big(\overline{\sigma}^{1,1}, g_2\big) = e\big(H_0(m), \overline{\sigma}^{1,2}\big) \cdot \prod\nolimits_{j\in S^1} e\Big(H_1(m, S_j^0, \mathsf{pk}_j^1),\ \hat{Z}_j\Big)$$
$$e\big(g_1, \overline{\sigma}^{1,2}\big) = \prod\nolimits_{j\in S^1} e\big(\hat{Y}_j, \textstyle\prod_{i\in S_j^0} \mathsf{pk}_i^0\big).$$

Therefore, in both $\mathsf{Aggregate}^1$ and $\mathsf{AggVerify}^1$, we save $O(n_1)$ multi-scalar multiplications, making the total cost of $\mathsf{Aggregate}^1$ $O(n_1)$ $\mathbb{G}_1$ additions, and the total cost of $\mathsf{AggVerify}^1$, $O(n_1)$ pairings and $O(n_0 n_1)$ $\mathbb{G}_1$ additions. Moreover, achieving algebraic verification again enables using SNARKs to verify important properties about the $O(n_0 n_1)$-sized bit map (*e.g.*, whether k layer-1 signers observed k layer-0 signatures) in poly-logarithmic time.

Security. As in Section 5.3, to prove security, it suffices to additionally assume hardness of the RMSS problem. The arguments for the event E_2 remain unchanged, since the probability of finding a preimage of 0 under H' is still bounded by q'/p. In E_1, we must again bound the probability that (7) is violated, which now becomes

$$\sum_{j\in J_{\mathcal{C}}^*} y_j H'(\mathsf{PK}, j) \neq 0\ . \tag{8}$$

This can be done using the same reduction in Section 5.3.

7 Implementation and Evaluation

To evaluate the concrete performance of our schemes, we implemented them in Rust, using `arkworks` [ac22] for implementations of pairing-friendly curves. Our implementation can be found at https://github.com/guruvamsi-policharla/doubly-aggregatable-signatures

Setup. Experiments were run on a 2019 MacBook Pro with a 2.4 GHz Intel Core i9 processor and 16 GB of DDR4 RAM in single-threaded mode. We use BLS12-381 [BLS03] as our pairing-friendly curve.

Strawman Solution. In Table 2, we benchmark our constructions against the strawman solution from Section 2.2 for three different configurations: small (S) ($n_0 = n_1 = 10$), medium (M) ($n_0 = n_1 = 100$), and large (L) ($n_0 = n_1 = 1000$), where all layer-1 signers attest to having seen $n_0/2$ signatures from layer-0 signers. Note that the strawman solution *does not* satisfy layer-1 unforgeability, whereas our constructions are provably-secure.

Table 2: Benchmarks of our doubly aggregatable signature schemes and the (insecure) strawman solution. Layer-0 algorithms are the same across all schemes. All times are reported in milliseconds. A single entry indicates that the benchmark is the same across all configurations.

Algorithm	Timing
	S / M / L
Sign^0	0.26
Verify^0	2.14
$\mathsf{Aggregate}^0$	0.003 / 0.03 / 0.42

Algorithm	Strawman (insecure)	CDH+RMSS (Sect. 5.3)	(2,1)-CDH (Sect. 6.3)
	S / M / L	S / M / L	S / M / L
Sign^1	0.27 / 0.36 / 0.76	0.65 / 3.31 / 20.3	1.11 / 1.28 / 2.94
Verify^1	4.30 / 4.81 / 6.07	2.17 / 2.22 / 3.60	4.80 / 4.80 / 6.12
$\mathsf{Aggregate}^1$	0.01 / 0.08 / 0.86	0.01 / 0.07 / 0.78	0.03 / 0.40 / 3.75
$\mathsf{AggVerify}^1$	28.6 / 288 / 4296	2.28 / 16.9 / 1489	15.2 / 140 / 2628
Repeat $\mathsf{AggVerify}^1$	28.1 / 261.6 / 2574	2.16	14.5 / 117.7 / 1197.1

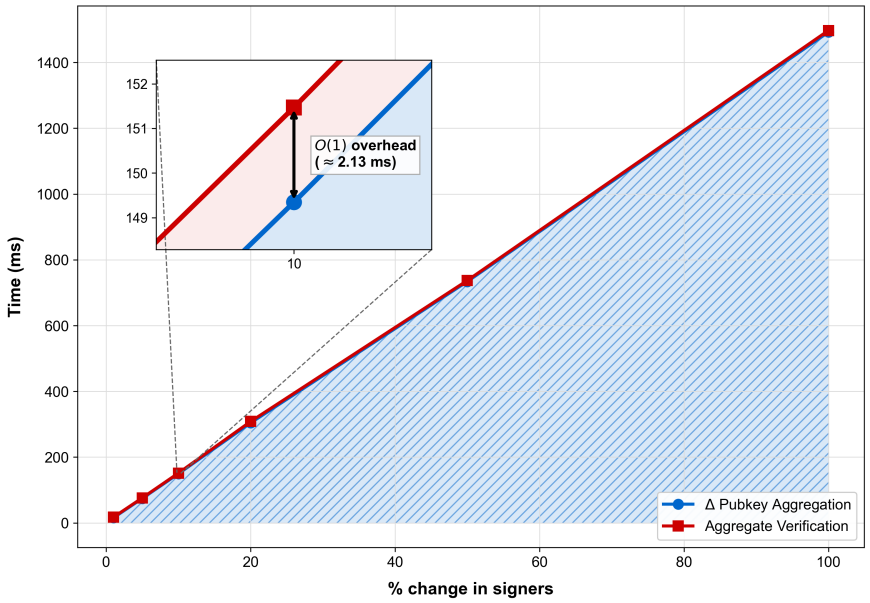

Figure 2: Optimized verification strategy of aggregate signatures in our construction from Section 5.3 with $n_0 = n_1 = 1000$. Given an aggregated attestation matrix M, the y-axis corresponds to the time taken to verify a signature on a different message under attestation matrix M'. Concretely, we start with half of layer-0 signers participating and the x-axis denotes the percentage change in layer-0 signers that participate.

Linear Public Keys. For our first construction (Section 5.3), verifying an aggregate layer-1 signature is over $12\times$ faster in the small configuration (note that in the small configuration, fixed costs such as pairings and hashing dominate). This drops down to $\approx 3\times$ in the large setting, where all schemes require $O(n_0 \cdot n_1)$ group operations to aggregate public keys, which begins to dominate the cost. However, our construction has the advantage that verifying a second message with the

same attestation bit matrix B only requires $O(1)$ pairings, whereas the strawman requires $O(n_1)$. Concretely, this translates to a $\approx 1200\times$ speedup (!) over the strawman in the large configuration. Thus, when B does not change, the verifier can *reuse* work such that verification cost is *independent* of the number of signers.

Slowly Changing Attestation Matrix. We now consider the case where the attestation matrix changes *slowly* between consecutive verifications. This is indeed the situation on the normal path of a consensus protocol, where parties regularly broadcast signatures, with very few timeouts/dropped messages. Our construction is also able to take advantage of slowly changing attestation matrices such that the additional work needed to verify only grows with the "change". To concretely demonstrate this advantage, we implement such a verification strategy that reuses work and benchmark it (Fig. 2 in the Appendix) against a percentage change in the number of signers in the large configuration. We observe that even if there is a 10% change in the set of layer-0 signers, we still obtain a $10\times$ speedup in verification time.

Constant Public Keys. The construction with constant-sized public keys (Section 6.3) can be viewed as a *secure* version of the strawman solution, and achieves $\approx 40\%$ faster aggregate signature verification compared to the strawman. Although it does not inherit the reusability benefit of the linear-sized-keys construction, verifying a second message with the same attestation bit matrix requires half the number of pairings, as $\overline{\sigma}^{1,2}$ remains the same in this case.

8 Discussion

In this section, we briefly describe potential improvements and extensions for our scheme.

8.1 Faster Verification

A major drawback of the scheme in Section 6 is that the verifier has to do up to $2n_1$ pairings to check the following relations:

$$1 = e\big((\overline{\sigma}^{1,1})^{-1}, g_2\big) \cdot e\big(H_0(m), \overline{\sigma}^{1,2}\big) \cdot \prod_{j \in S^1} e\Big(H_1(m, S_j^0, \mathsf{pk}_j^1)^{H'(\mathsf{pk}_{S^1}^1, j)},\ Z_j\Big)$$

$$1 = e\big(g_1, (\overline{\sigma}^{1,2})^{-1}\big) \cdot \prod_{j \in S^1} e\Big((Y_j)^{H'(\mathsf{pk}_{S^1}^1, j)}, \textstyle\prod_{i \in S_j^0} \mathsf{pk}_i^0\Big).$$

To avoid this, we can use the statistically-sound Inner Pairing Product (SIPP) proof system introduced in [BMM+21, Appendix B]. SIPP is a proof system for the following language:

$$\mathcal{L}_{\text{SIPP}} = \{(A \in G_1^m,\ r \in \mathbb{F}^m,\ B \in G_2^m,\ Z \in G_T) :\ Z = A^r * B\}\,.$$

Then, aggregators run $\mathsf{SIPP.Prove}(A_i, r_i, B_i, Z = 1)$ to obtain the SIPP proofs π^1_{SIPP} and π^2_{SIPP}, where A_i, r_i, B_i are defined as follows:

$\mathbf{i = 1}$: $A_1 = ((\overline{\sigma}^{1,1})^{-1} \| H_0(m) \| \{H_1(m, S_j^0, \mathsf{pk}_j^1)\}_{j \in S^1})$,
$B_1 = (g_2 \| \overline{\sigma}^{1,2} \| \{Z_j\}_{j \in S^1})$, and $r_1 = (1 \| 1 \| \{H'(\mathsf{pk}_{S^1}^1, j)\}_{j \in S^1})$.

$\mathbf{i = 2}$: $A_2 = (g_1 \| \{Y_j\}_{j \in S^1})$, $B_2 = ((\overline{\sigma}^{1,2})^{-1} \| \{\prod_{i \in S^0_j} \mathsf{pk}^0_i\}_{j \in S^1})$, and
$r_2 = (1 \| \{H'(\mathsf{pk}^1_{S^1}, j)\}_{j \in S^1})$.

They output π^1_{SIPP} and π^2_{SIPP} along with $\overline{\sigma}^1 := (\overline{\sigma}^{1,1}, \overline{\sigma}^{1,2})$.

Verifiers construct A_i, B_i and r_i for $i = 1, 2$ using $\overline{\sigma}^{1,1}, \overline{\sigma}^{1,2}$, and $\{\mathsf{pk}^1_j, \{\mathsf{pk}^0_i\}_{i \in S^0_j}\}_{j \in S^1}$. They output 1 if the NIZK and the SIPP proofs verify, *i.e.*, $\mathsf{SIPP.Verify}(A_i, r_i, B_i, Z = 1, \pi^i_{\mathsf{SIPP}}) = 1$.

Defining $m = |S^1| + 2 \leq n_1$, to generate the SIPP proofs, the aggregator does a total of $4m$ pairings, and $2m$ exponentiations in each source group. Each SIPP proof then consists of at most $\lceil 2 \log(m) \rceil$ target group elements. To verify the SIPP proofs, the verifier does a total of two pairings, $2\lceil 2 \log(m) \rceil$ target group exponentiations, and two m-sized MSM in each source group. This reduces the overall verifier complexity from "$O(n_0 \cdot n_1)$ $\mathbb{G}_2$ operations and $O(n_1)$ pairings" to "$O(n_0 \cdot n_1)$ $\mathbb{G}_2$ operations, $O(\log(n_1))$ $\mathbb{G}_T$-exponentiations, 2 $\mathsf{MSM}(n_1)$ in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and $O(1)$ pairings", at the expense of more work for the aggregators. This optimization also applies to the scheme in Section 6.3, where A_i, r_i, B_i would be defined as follows:

$\mathbf{i = 1}$: $A_1 = ((\overline{\sigma}^{1,1})^{-1} \| H_0(m) \| \{H_1(m, S^0_j, \mathsf{pk}^1_j)\}_{j \in S^1})$,
$B_1 = (g_2 \| \overline{\sigma}^{1,2} \| \{\hat{Z}_j\}_{j \in S^1})$, and $r_1 = (1 \| 1 \| \ldots \| 1)$.

$\mathbf{i = 2}$: $A_2 = (g_1 \| \{\hat{Y}_j\}_{j \in S^1})$, $B_2 = ((\overline{\sigma}^{1,2})^{-1} \| \{\prod_{i \in S^0_j} \mathsf{pk}^0_i\}_{j \in S^1})$, and
$r_2 = (1 \| 1 \| \ldots \| 1)$.

8.2 Aggregating Signatures on Multiple Messages

Both schemes in Section 5 and Section 6 can support aggregating layer-0 signatures over multiple messages. To illustrate this point, suppose each layer-0 signature is on a different message m_i, $i \in [n_0]$. Then, in Section 5, Verify^1 returns 1 if the NIZK proofs verify and

$$e(\sigma^1, g_2) = \prod_{i \in [n_0]} e(H_0(m_i), Y_{j,i}).$$

Similarly, $\mathsf{AggVerify}^1$ returns 1 if the NIZK proofs verify and

$$e(\overline{\sigma}^1, g_2) = \prod_{i \in [n_0]} e\Big(H_0(m_i), \prod_{j \in S^1} (Y_{j,i})^{k_{i,j} H'(\mathsf{pk}^1_{S^1}, j)}\Big),$$

where $k_{i,j} = 1$ if $i \in S^1_j$, and 0 otherwise. Algorithms Sign^1 and $\mathsf{Aggregate}^1$ remain unchanged, and the optimizations in Section 5.3 remain applicable in the setting of multiple messages.

In Section 6, all observed messages are input to the hash function H_1. The algorithms are modified as follows. For ease of exposition, suppose the union of layer-0 signatures observed by the layer-1 signers is $[n_0]$.

- $\mathsf{Sign}^1((y, z), \{\mathsf{pk}^0_i, m_i, \sigma^0_i\}_{i \in S^0})$: If $\{\mathsf{Verify}^0(\mathsf{pk}^0_i, m_i, \sigma^0_i) = 1\}_{i \in S^0}$, then return $\sigma^1 := (\sigma^{1,1}, \sigma^{1,2})$ with

  $$\sigma^{1,1} := H_1(\{i, m_i\}_{i \in S^0}, \mathsf{pk}^1)^z \cdot \textstyle\prod_{i \in S^0} (\sigma^0_i)^y \quad \text{and} \quad \sigma^{1,2} := ((\mathsf{pk}^0_i)^y)_{i \in [S^0]}$$

  Else, return $\perp$.

- $\mathsf{Verify}^1(\mathsf{pk}^1, \{\mathsf{pk}_i^0, m_i\}_{i\in S^0}, \sigma^1)$:
  - parse $\mathsf{pk}^1 = (Y, Z)$, $\sigma^1 = (\sigma^{1,1}, \sigma^{1,2})$, and $\sigma^{1,2}$ into $\{\sigma_i^{1,2}\}_{i\in[S^0]}$.
  - return 1 iff

$$e(\sigma^{1,1}, g_2) = \prod_{i\in S^0} e(H_0(m_i), \sigma_i^{1,2}) \cdot e(H_1(\{i, m_i\}_{i\in S^0}, \mathsf{pk}^1), Z).$$

- $\mathsf{Aggregate}^1(\{\mathsf{pk}_j^1, \sigma_j^1, \{\{m_i, \mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in S^1})$:
  - Let $\mathsf{pk}_{S^1}^1 := \{\mathsf{pk}_j^1\}_{j\in S^1}$.
  - parse $\sigma_j^1 = (\sigma_j^{1,1}, \sigma_j^{1,2})$ and $\sigma_j^{1,2} = (\sigma_{j,i}^{1,2})_{i\in S_j^0}$
  - compute $\overline{\sigma}^{1,1} := \prod_{j\in S^1} (\sigma_j^{1,1})^{H'(\mathsf{pk}_{S^1}^1, j)}$ and
    $\overline{\sigma}_i^{1,2} := \prod_{j\in S^1, k_{i,j}=1} \sigma_{j,i}^2$ for each $i \in [n_0]$.
  - return $\overline{\sigma}^1 := (\overline{\sigma}^{1,1}, \overline{\sigma}^{1,2} := \{\overline{\sigma}_i^{1,2}\}_{i\in[n_0]})$.
- $\mathsf{AggVerify}^1(\{\mathsf{pk}_j^1, \{m_i, \mathsf{pk}_i^0\}_{i\in S_j^0}\}_{j\in S^1}, m, \overline{\sigma}^1)$:
  - For each $j \in S^1$, parse $\mathsf{pk}_j^1 = (Y_j, Z_j)$, and $\overline{\sigma}^{1,2}$ into $\{\overline{\sigma}_i^{1,2}\}_{i\in[n_0]}$.
  - return 1 iff the NIZKs verify and the following hold:

$$\begin{aligned} e\big(\overline{\sigma}^{1,1}, g_2\big) &= \prod_{i\in[n_0]} e\big(H_0(m_i), \overline{\sigma}_i^{1,2}\big) \\ &\quad \cdot \prod_{j\in S^1} e\Big(H_1(\{i, m_i\}_{i\in S_j^0}, \mathsf{pk}_j^1)^{H'(\mathsf{pk}_{S^1}^1, j)},\ Z_j\Big) \\ e\big(g_1, \overline{\sigma}_i^{1,2}\big) &= \prod_{j\in S^1} e\Big((Y_j)^{H'(\mathsf{pk}_{S^1}^1, j)}, \mathsf{pk}_i^0\Big) \text{ for each } i \in [n_0] \end{aligned}$$

Again, the optimizations in Section 6.3 remain applicable in the setting of multiple messages. Note that the size of layer-1 signatures scales in the number of unique messages, as now the cross-terms derived from the secret keys are broadcast as part of each layer-1 signature, as opposed to the scheme in Section 5 that posts them as a quadratic-sized public key. Indeed, the scheme in Section 6 offers savings in the setting of multiple messages only if the number of unique messages is much lower than the number n_0 of layer-0 signers, in which case the cross-terms corresponding to the layer-0 signatures on the same message can be aggregated.

Acknowledgments. This work was performed in part during a research internship of the fourth author and a research visit of the first author at *a16z crypto*. The first author is supported by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18002] and by the Austrian Science Fund (FWF) [10.55776/F8515-N].

References

[AB19] Shahla Atapoor and Karim Baghery. Simulation extractability in groth's zk-SNARK. In Cristina Pérez-Solà, Guillermo Navarro-Arribas, Alex Biryukov, and Joaquin Garcia-Alfaro, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology (DPM 2019, CBT 2019)*, volume 11737 of *Lecture Notes in Computer Science*, pages 336–354, Cham, 2019. Springer. 3

[ac22] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022. 28

[Anz24] Anza. Timely vote credits, 2024. Solana system design proposal: timely vote credits, https://docs.anza.xyz/proposals/timely-vote-credits. 1, 2

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In *Advances in Cryptology – ASIACRYPT 2023*, volume 14439 of *Lecture Notes in Computer Science*, pages 77–110. Springer, 2023. 3

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 111–120. ACM, 2013. 3

[BCG+24] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, François Garillot, Jonas Lindstrøm, Ben Riva, Arnab Roy, Mahdi Sedaghat, Alberto Sonnino, Pun Waiwitlikhit, and Joy Wang. Subset-optimized BLS multi-signature with key aggregation. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 188–205. Springer, Cham, March 2024. 4, 9, 21, 22, 27

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008. 4

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography Conference (TCC 2020), Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2020. 3

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 649–680. Springer, 2021. 3

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Cham, December 2018. 4, 8, 21

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget, 2017. 1

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. 3

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Berlin, Heidelberg, May 2003. 2, 4, 18, 25

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 276–285. ACM Press, October 2007. 4

[BHK+20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020. 1

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. 1, 2

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001. 2, 3, 4, 6, 17

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Berlin, Heidelberg, September 2003. 28

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Cham, December 2021. 6, 9, 11, 23, 30

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006. 4

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003. 4, 6, 17, 18, 25

[Boy08] Xavier Boyen. The uber-assumption family – a unified complexity framework for bilinear groups. In *2nd International Conference on Pairing-based Cryptography—PAIRING 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 39–56. Berlin: Springer-Verlag, 2008. Available at `http://www.cs.stanford.edu/~xb/pairing08/`. 13

[Bri83] Ernest F. Brickell. Solving low density knapsacks. In David Chaum, editor, *Advances in Cryptology — CRYPTO '83*, pages 25–37. Plenum Press, New York, NY, USA, 1983. 3, 9

[BSCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294. Springer, 2014. 3

[CKM21] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. 4

[CKP+23] Elizabeth C. Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 348–382. Springer, Singapore, December 2023. 4

[CL24] Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin's transform for UC-secure zero knowledge. *CiC*, 1(2):11, 2024. 17

[CLOS91] Matthijs J. Coster, Brian A. LaMacchia, Andrew M. Odlyzko, and Claus-Peter Schnorr. An improved low-density subset sum algorithm. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 54–67. Springer, 1991. 3, 9

[CLPO25] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. Minimmit: Fast finality with even faster blocks, 2025. 2

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology – EUROCRYPT 2020*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793. Springer, 2020. 3

[CP23] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023. 1, 2

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 356–370. ACM Press, November 2023. 9

[Edg23a] Ben Edgington. BLS signatures, 2023. https://eth2book.info/latest/part2/building_blocks/signatures/. 8

[Edg23b] Ben Edgington. Reward of the consensus layer, 2023. https://eth2book.info/capella/part2/incentives/rewards/. 1

[ES16] Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 140–155. Springer, Cham, November 2016. 4

[FH20] Masayuki Fukumitsu and Shingo Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang, editors, *ProvSec 2020*, volume 12505 of *LNCS*, pages 45–64. Springer, Cham, November / December 2020. 4

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005. 17

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. 21

[Fri86] Alan M. Frieze. On the lagarias–odlyzko algorithm for the subset sum problem. *SIAM Journal on Computing*, 15(2):536–539, 1986. 3, 9

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy*, pages 3034–3052. IEEE Computer Society Press, May 2024. 9

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Cham, 2017. Springer. 3

[GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 203–229. Springer, Cham, May 2022. 6, 9

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Berlin, Heidelberg, December 2006. 7

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Berlin, Heidelberg, April 2008. 6

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptol.*, 9(4):199–216, 1996. 13

[Ita83] Kazuharu Itakura. A public-key cryptosystem suitable for digital multisignature. *NEC research and development*, 71:1–8, 1983. 4

[KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. 3

[KS24] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. In *Advances in Cryptology – CRYPTO 2024*, volume 14929 of *Lecture Notes in Computer Science*, pages 345–379. Springer, 2024. 3

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer, 2022. 3

[KSW25] Quentin Kniep, Kobi Sliwinski, and Roger Wattenhofer. Alpenglow: A new consensus for solana. Technical report, Anza Technology, 2025. White paper and SIMD-0326 proposal. 1, 2

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Cham, November 2021. 9

[Lin22] Yehuda Lindell. Simple three-round multiparty Schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. 4

[LO85] J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985. 3, 9

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Berlin, Heidelberg, May / June 2006. 4, 18, 25

[Lyu05] Vadim Lyubashevsky. On random high density subset sums. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 12, 2005. 3, 9

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001. 4

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019. 4

[NKDM03] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*. The Internet Society, February 2003. 4

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Cham. 4

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020. 4

[OO93] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 139–148. Springer, Berlin, Heidelberg, November 1993. 4

[PW23] Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 597–627. Springer, Cham, April 2023. 4

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Berlin, Heidelberg, May 2007. 4

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991. 4

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997. 13

[STM+25] Roozbeh Sarenche, Ertem Nusret Tas, Barnabé Monnot, Caspar Schwarz-Schilling, and Bart Preneel. Commitment attacks on ethereum's reward mechanism. In *EuroS&P*, pages 504–526. IEEE, 2025. 2

[TZ23] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023. 4