Pairing-based Accountable Subgroup Multi-signatures with Verifiable Group Setup

Ahmet Ramazan Ağırtaş[a,b,*], Oğuz Yayla[b]

[a] *AYWARE, 06420, Ankara, Türkiye*
[b] *Institute of Applied Mathematics, Department of Cryptography, Middle East Technical University, 06800, Ankara, Türkiye*

Abstract

In an accountable subgroup multi-signature (ASM), any subgroup $\mathcal{S}$ of a group of signers can sign a message so that each member of $\mathcal{S}$ is accountable for the signature. We propose three pairing-based ASM schemes, unforgeable under the co-CDH assumption in the random oracle and algebraic group models, even when the adversary corrupts all but one member and chooses their keys. The first, vASM, sets up the group with Feldman's verifiable secret sharing and a proof of knowledge of the sharing coefficients; its signature is one $\mathbb{G}_1$ element. Without this proof, a rogue-key attack forges for any target subgroup of size at least two, even with proofs of possession when the group has at least three members. In the second, ASMwSA, the subgroup is fixed before signing; in the third, ASMwCA, a designated combiner authenticates it. In both, keys carry a proof of knowledge, without which a subset-sum attack becomes feasible for large groups. All our schemes verify with one pairing fewer than the ASM scheme of Boneh, Drijvers and Neven (BDN), at the cost of a costlier one-time setup. vASM verifies in 1.3 to 1.6 ms for up to 256 members, faster than BDN for up to 64 members even when its verifier stores per-group hashes. Like BDN, vASM and ASMwCA do not need $\mathcal{S}$ before signing; in ASMwSA the combiner can be eliminated. The aggregated versions output one $\mathbb{G}_1$ element for N signatures and verify with $N + 1$ pairings, against $N + 1$ group elements and $2N + 1$ pairings for BDN.

*Corresponding author.
Email addresses: ahmet.agirtas@alumni.metu.edu.tr (Ahmet Ramazan Ağırtaş), oguz@metu.edu.tr (Oğuz Yayla)

November 23, 2026

Keywords: accountable subgroup multi-signatures, BLS signature, aggregatable signatures, verifiable secret sharing, rogue-key attack, random oracle model, algebraic group model

1. Introduction

Digital signature schemes play a crucial role in modern cryptographic protocols for verifying the authenticity of any message, such as official documents, financial transactions, e-mails, etc. In particular, the increasing popularity of cryptocurrencies [1, 2, 3] in the last decade made the digital signature schemes more significant than before. The structures of signature schemes differ according to many reasons such as use area, system requirements, and users' needs.

A *multi-signature* [4, 5, 6, 7, 8] is a kind of digital signature in which a group of signers signs the same message jointly. In literature, there are some notions related to multi-signatures for different scenarios, such as group signatures [9, 10], threshold signatures [7, 11, 12, 13], aggregate signatures [14], ring signatures [15], accountable subgroup multi-signatures [16, 17], etc. None of them provide sufficient flexibility regarding the number of signers, and accountability of the signers at the same time, except accountable subgroup multi-signatures and accountable threshold signatures [18]. An *accountable subgroup multi-signature (ASM)* [16, 17] is a multi-signature scheme in which any subgroup $\mathcal{S} \subseteq \mathcal{G}$ jointly signs a message m, ensuring that each member of $\mathcal{S}$ is accountable for the resulting signature. This notion was first defined by Micali et al. in [17] by proposing the first ASM scheme. Micali et al. left open the construction of an ASM scheme in which the subgroup of signers $\mathcal{S} \subseteq \mathcal{G}$ is not determined before the signature generation. Boldyreva [7] solved this problem with a BLS-based multi-signature in which any subset of the registered signers can sign without interaction, and the verifier multiplies the public keys of the signers. In a more recent paper [16], Boneh, Drijvers and Neven proposed another ASM scheme which is based on BLS signature [19], in which $\mathcal{S}$ is likewise not determined in advance, and whose aggregate public key has a size independent of the number of members. Accountability has also been added to threshold signatures [18, 20], and accountable multi-signatures have been built with a constant-size verification key [21], for cross-chain bridges [22] and from lattices [23, 24].

In this paper, we focus on accountable subgroup multi-signatures (ASM). We propose three novel accountable subgroup multi-signature schemes based

on pairings. We prove that our proposed schemes are secure against existential forgery under chosen-message attacks, under co-CDH/ψ-co-CDH assumption in the random oracle model and the algebraic group model, in the model of Boneh, Drijvers and Neven [16], where the adversary corrupts $n-1$ of the n members and chooses their public keys. The first one is the vASM (verifiable ASM) scheme which is a modified BLS signature. We give a method of generating a membership key via VSS protocol [25], which transforms the BLS signature scheme into an ASM scheme. For this setup to resist rogue-key attacks, every member must also prove knowledge of all coefficients of its sharing polynomial; we give an explicit attack otherwise. The proposed vASM scheme, which, like the schemes of [7] and [16], solves the open problem in [17], requires fewer bilinear pairings in verification than the ASM schemes proposed in [16], and its signatures aggregate into a single group element. The second one is ASMwSA (ASM with Subgroup Authentication), in which the subgroup of the signers is known before the protocol starts. So the members participating in authentication are decided by the subgroup itself. The third one is ASMwCA (ASM with Combiner Authentication) which also provides a solution to the open problem in [17], in which we construct a scheme such that the subgroup of signers $\mathcal{S}$ is not predetermined. The ASMwSA and ASMwCA schemes also require fewer bilinear pairings in verification than the ASM scheme in [16]. Moreover, we give a method of consecutive and cumulative signing that eliminates the designated combiner of ASMwSA in case the subgroup of signers $\mathcal{S}$ is known before the signature generation. Further, we discuss the aggregated versions of vASM, ASMwSA and ASMwCA schemes for N distinct accountable subgroup multisignatures. The aggregated versions of our schemes, i.e. AvASM, AASMwSA and AASMwCA, output aggregated signatures with the size of a single group element and require $N+1$ pairings for aggregated signature verification, in comparison with the partial aggregated AASM scheme proposed in [16] with the signature size of $N+1$ group elements and verification with $2N+1$ pairings.

Contributions. In summary, this paper makes the following contributions.

- Three pairing-based ASM schemes, vASM, ASMwSA and ASMwCA, with security proofs under co-CDH (ψ-co-CDH for the aggregated versions) in the random oracle model and the algebraic group model, in the model where the adversary corrupts $n-1$ members and chooses their keys.

- Two attacks that show why the proofs of knowledge are needed: a rogue-coefficient attack on a Feldman setup without them, which forges for any target subgroup of size at least two even when keys carry proofs of possession and the group has at least three members, and a subset-sum attack on the subgroup key of ASMwSA and ASMwCA. A third attack shows that the keys of a group must be distinct.

- For vASM, verification with two pairings, using only the n membership public keys of the group, which the verifier computes once from the commitments. In our implementation on an Apple M2 Pro, vASM verifies a signature in 1.3 to 1.6 ms for groups of 4 to 256 members. The ASM scheme of [16] needs 7.8 ms for 64 members and 27.6 ms for 256 when the verifier hashes the member indices in every verification, and 1.5 to 1.6 ms when the verifier stores these hashes for each group. vASM is faster in the first case, and in the second for up to 64 members, with equal times at 256; with the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ exchanged it is faster in both cases for all group sizes. Signing a share costs 0.3 to 0.4 ms in all schemes.

- Full aggregation: N signatures from different groups aggregate into one $\mathbb{G}_1$ element and verify with $N + 1$ pairings, against $N + 1$ group elements and $2N + 1$ pairings for [16]; verifying 100 aggregated vASM signatures takes 49 ms, about 0.5 ms per signature.

- An open-source implementation of all schemes, the attacks and the benchmarks. The price of verification with two pairings is a one-time group setup: 3.1 s per member for 64 members with vASM, which pays off after about 470 verifications against a verifier of [16] that hashes the member indices every time.

Changes with respect to the first version. This report revises its first version (2022) [26]. The schemes were also presented in the first author's PhD thesis [27]. The group setup of vASM given there is open to the attack of Section 5.3; the changes listed below, in particular the proofs of knowledge and the check that the keys are distinct, fix it. The main changes are: (i) the security model now lets the adversary corrupt $n-1$ members and choose their keys (Definition 3.6); in the first version the challenger generated all keys; (ii) in that model the VSS-based setup of vASM is not secure on its own, and we add a proof of knowledge of the sharing coefficients, a group identifier in the

message hash and a check that the keys of a group are distinct (Sections 5.1 and 5.3); (iii) we state the properties of Shamir sharing that the new security proof for vASM uses (Lemma 2); (iv) for ASMwSA and ASMwCA we show a subset-sum attack on the subgroup key when keys carry no proof of knowledge, add such proofs, and give new reductions without forking, including an explicit one for ASMwCA; we also make explicit that the verifier computes the subgroup key itself; (v) all proofs of knowledge are Schnorr proofs made non-interactive with the Fiat–Shamir transform, and the new reductions are in the algebraic group model with random oracles, where the secrets of the corrupt members can be extracted without rewinding; (vi) we define security for the aggregated schemes and prove it by direct reductions; (vii) we survey related work up to 2026 and add an implementation and benchmarks.

The outline of the paper is as follows. Section 2 surveys related work. In Section 3 we give a brief background information, including definitions of bilinear pairings, co-CDH/ψ-co-CDH problems, Feldman's Verifiable Secret Sharing (VSS) protocol, proofs of knowledge in the algebraic group model, multi-signatures and accountable subgroup multi-signatures, and the BLS signature scheme. In Section 4 we summarize the ASM scheme given in [16]. Then, in Section 5.1, we give our vASM scheme, the attacks on a setup without proofs of knowledge, and prove its security in the random oracle model and the algebraic group model. Moreover, in Section 6 we propose ASMwSA and ASMwCA schemes which are based on subgroup authentication instead of global authentication. In the same section we prove their security in the random oracle model and the algebraic group model, without the generalized forking lemma. In Section 7, we summarize the partial aggregated ASM (AASM) scheme given in [16], and discuss the aggregated versions of our proposed schemes and their security. Finally, in Section 8, we compare our new schemes with ASM and AASM schemes in terms of the number of operations required in the phases of the schemes and costs of transmission, broadcasting and storage. Section 9 reports measured running times, and Section 10 concludes the paper.

2. Related work

Multi-signatures and rogue keys. Multi-signatures go back to Itakura and Nakamura [4]. The central difficulty is the rogue-key attack, in which a corrupt signer chooses its public key as a function of the honest keys so that the aggregate key is one it controls [14, 28]. Two approaches are standard.

The first registers every public key together with a proof of possession (PoP) of its secret key. This makes concrete the knowledge-of-secret-key (KOSK) model [7, 8], in which the adversary must reveal the secret key of every public key it uses. Ristenpart and Yilek [29] showed that the form of the proof matters: the standardized PoP, a signature on the public key computed with the same hash function as the signatures on messages, does not prevent rogue-key attacks on BLS multi-signatures, whereas a BLS signature on the public key under a separate hash function does. The second uses key-dependent coefficients, as in MuSig [28] and in the MSP scheme of Boneh, Drijvers and Neven [16]; it needs no registration step and works in the plain public-key model of Bellare and Neven [8]. The IRTF draft [30] specifies a PoP variant of BLS, and Ethereum uses it. Baldimtsi *et al.* [31] gave a BDN-style multi-signature optimized for verification by many different subsets; their proof needs the algebraic group model and, for large groups, a random modular subset-sum assumption.

Accountable subgroup multi-signatures. Micali, Ohta and Reyzin [17] introduced ASMs; in their scheme the signing subgroup must be fixed before signing. Boneh, Drijvers and Neven [16] removed this restriction with a pairing-based ASM whose group setup distributes BLS signatures on member indices, and gave a partially aggregatable version. Boneh, Partap and Waters [21] recently built accountable multi-signatures in which the verifier stores a constant-size key, independent of the number of members, signatures have constant size beyond the description of the signing set, and signers generate their keys locally. Our schemes take the opposite trade-off: the verifier needs $O(n)$ public data per group but verification is two pairings and signatures aggregate across groups into one $\mathbb{G}_1$ element.

Accountable threshold signatures. A threshold signature hides which t of the n parties signed. Accountable threshold signatures make the signers public, and Boneh and Komlo [18] introduced *private* accountability, where only a tracing authority learns the signers. Boneh, Partap and Rotem [20] added proactive refresh to accountable threshold signatures. An ASM gives a threshold-like policy by requiring $|\mathcal{S}| \geq t$ at verification [17].

Applications and recent constructions. Wang *et al.* [22] proposed TMDC, a traceable multi-signature for cross-chain bridges in which signers generate keys independently, combiners are distributed and time-locked, and a tracing algorithm links each signer to the final signature. Chen *et al.* [23] gave a

non-interactive identity-based ASM from lattices. In earlier work we built a lattice-based ASM with verifiable group setup [24] and used vASM as a building block for compartment-based and hierarchical threshold delegation [32]. Both depend on the vASM setup analyzed here.

Two recent works improve the security proofs of BLS multi-signatures. Bacho and Wagner [33] gave a non-interactive BLS multi-signature whose security loss does not depend on the number of signing queries and whose signatures are ordinary BLS signatures. Tan, Ng and Heng [34] proposed a BLS multi-signature with a proof that avoids the forking lemma and that lets signers be added to an existing multi-signature on the same message. Our proofs for ASMwSA and ASMwCA also avoid forking, but for a different reason: in the algebraic group model, the key proofs give the reduction the corrupt secret keys directly. On the threshold side, hinTS by Garg *et al.* [35] is a BLS threshold signature with a *silent* setup: the aggregate key is computed from keys and public hints that parties publish without interacting, and the threshold and the set of signers can be chosen after the setup without further interaction; weighted thresholds are supported natively. hinTS is a threshold signature, whereas an ASM reveals the signing subgroup to the verifier, and vASM pays for the two-pairing verification with an interactive, but verifiable, group setup.

3. Background

In order to provide sufficient background information for readers, we give the definitions of notions that we mainly use throughout this paper. Namely, we give the definitions of bilinear pairings, computationally hard problems, Feldman's VSS protocol, proofs of knowledge in the algebraic group model, multi-signatures and accountable subgroup multi-signatures, and the BLS signature scheme.

3.1. Bilinear pairings and computational hard problems

Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ be cyclic groups of prime order q, all written multiplicatively.

Definition 3.1. A pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which satisfies the bilinearity and non-degeneracy properties:

- Bilinearity: $e(A^\alpha, B^\beta) = e(A, B)^{\alpha\beta}$ for all $\alpha, \beta \in \mathbb{Z}_q$, $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

- Non-degeneracy: $e(g_1, g_2) \neq 1$.

The definitions of underlying hard problems of the schemes in this paper, i.e. computational co-Diffie-Hellman and computational ψ-co-Diffie-Hellman problems, are given below.

Definition 3.2 (Computational co-Diffie-Hellman Problem [14], [16])**.** For groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order q, define $Adv^{\text{co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2}$ of an adversary $\mathcal{A}$ as

$$Pr\left[y = g_1^{\alpha\beta} : (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_q^2, y \leftarrow \mathcal{A}(g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})\right],$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of (α, β). $\mathcal{A}$ (τ, ϵ)-breaks the co-CDH problem if it runs in time at most τ and has $Adv^{\text{co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2} \geq \epsilon$. co-CDH is (τ, ϵ)-hard if no such adversary exists.

Definition 3.3 (Computational ψ-co-Diffie-Hellman Problem [16])**.** For groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order q, let $\mathcal{O}^{\psi}(.)$ be an oracle that returns $g_1^x \in \mathbb{G}_1$ on input $g_2^x \in \mathbb{G}_2$. Define $Adv^{\psi\text{-co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2}$ of an adversary $\mathcal{A}$ as

$$Pr\left[y = g_1^{\alpha\beta} : (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_q^2, y \leftarrow \mathcal{A}^{\mathcal{O}^{\psi}(.)}(g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})\right],$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of (α, β). $\mathcal{A}$ (τ, ϵ)-breaks the ψ-co-CDH problem if it runs in time at most τ and has $Adv^{\psi\text{-co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2} \geq \epsilon$. ψ-co-CDH is (τ, ϵ)-hard if no such adversary exists.

3.2. Feldman's VSS Protocol

Feldman's verifiable secret sharing (VSS) scheme [25] is a protocol that is used for sharing a secret among some predetermined players in a verifiable fashion, in which Shamir's secret sharing scheme [36] was directly used to share and reconstruct the secret. In addition to Shamir's scheme, the shares can be checked for consistency in Feldman's scheme. To this end, the dealer computes commitments with the coefficients of the secret polynomial. By this way, users can verify that they receive consistent shares from the dealer.

Assume that we have n players. Let $g = g_2$ be a generator of $\mathbb{G}_2$ and let all coefficients lie in $\mathbb{Z}_q$; we describe the protocol over $\mathbb{G}_2$, where it is used in Section 5.1. The dealer shares a secret as follows:

- Chooses a polynomial of degree $t-1$, where $t \leq n < q$,

$$f(x) = \alpha_{t-1}x^{t-1} + \ldots + \alpha_1 x + \alpha_0$$

with $\alpha_1, \ldots, \alpha_{t-1} \xleftarrow{\$} \mathbb{Z}_q$ chosen uniformly, where α_0 is the secret to be shared.

- Computes a set of commitments $\text{COM} = \{C_k : C_k = g^{\alpha_k}, k = 0, 1, \ldots, t-1\}$.
- Sends $f(i)$ and COM to the i-th player for $i = 1, 2, \ldots, n$.

After receiving a share and the set of commitments, the i-th player checks

$$g^{f(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} C_k^{i^k}. \tag{1}$$

The received share is consistent with the shared secret only if (1) is satisfied. If at least any t or more players perform Lagrange interpolation with their shares, they can uniquely determine the secret polynomial, and $f(0)$ will yield the secret.

The commitment set contains $C_0 = g^{\alpha_0}$, where g is the generator for the cyclic group, and α_0 is the secret to be shared. This commitment may leak information about the secret α_0. The security of the commitments depends on the Discrete Logarithm Problem (DLP), which is defined over cyclic groups. In some cyclic groups, even with a large order, DLP may not be as hard as it is supposed to be. Therefore the space that we are working in should be chosen carefully.

In Section 5.1 we use this protocol, together with a proof of knowledge of the coefficients (Section 3.3), for the group setup. We use only the sharing, committing and verifying phases of this protocol.

3.3. Proofs of knowledge in the algebraic group model

For $C_0, \ldots, C_d \in \mathbb{G}_2$, a prover who knows α_k with $C_k = g_2^{\alpha_k}$ for every k proves it non-interactively in the random oracle model as follows. It samples $\rho_k \xleftarrow{\$} \mathbb{Z}_q$, sets $R_k = g_2^{\rho_k}$, computes $c = H_{\mathsf{pok}}(\mathsf{ctx}, (C_k)_k, (R_k)_k)$ and $z_k = \rho_k + c\,\alpha_k$, and outputs $\pi = (c, (z_k)_k)$. The verifier accepts if and only if $c = H_{\mathsf{pok}}(\mathsf{ctx}, (C_k)_k, (g_2^{z_k} C_k^{-c})_k)$. This is $d+1$ Schnorr proofs [37] sharing one challenge, made non-interactive with the Fiat–Shamir transform [38]; we call

π a *non-interactive Schnorr proof*. Two accepting transcripts with the same first message and different challenges reveal every α_k. The context string ctx binds the proof to a list of public keys and a member index. Together with the requirement that the keys in a group are distinct, this prevents a proof from being replayed under another identity (Remark 5.5).

The proof consists of $d + 2$ elements of $\mathbb{Z}_q$; for the 64 coefficients of a vASM member in a group of $n = 64$ it takes about 2 KiB on BLS12-381. A simulator that controls H_{pok} produces a proof for given $(C_k)_k$ and ctx without knowing the α_k: it samples $c, z_0, \ldots, z_d \xleftarrow{\$} \mathbb{Z}_q$, sets $R_k = g_2^{z_k} C_k^{-c}$ and programs $H_{\mathsf{pok}}(\mathsf{ctx}, (C_k)_k, (R_k)_k) = c$. Since the R_k are uniform, the simulated proof is distributed exactly as an honest one, unless H_{pok} has already been queried at this point; after at most q_H queries, this happens with probability at most q_H/q.

In the random oracle model alone, the known way to extract the secret from Schnorr signatures, which are non-interactive Schnorr proofs of this kind, is to rewind the prover with the forking lemma [39]; for the α_k this is discussed in Remark 5.7. Our reductions must extract the secrets of all corrupt members, in many group setups that the adversary may run concurrently. We therefore prove security in the algebraic group model (AGM) [40] combined with the random oracle model. In this setting Fuchsbauer, Plouviez and Seurin showed that extraction from Schnorr signatures is straight-line [41, Section 3.2], and Baldimtsi *et al.* [31] also use the algebraic group model for a BLS-type multi-signature. An algorithm is *algebraic* [40] if, for every group element it outputs, it also outputs a representation of this element relative to the group elements it has received so far. In our asymmetric setting, a representation of $U \in \mathbb{G}_2$ is a vector of exponents $(\gamma_j)_j$ with $U = \prod_j V_j^{\gamma_j}$, where $V_1, V_2, \ldots$ are the elements of $\mathbb{G}_2$ received so far, and likewise for $\mathbb{G}_1$. Group elements in queries to random oracles count as outputs. Our proofs use only the representations of elements of $\mathbb{G}_2$. The following lemma adapts the argument of [41] to our proofs.

Lemma 1 (Straight-line extraction)**.** *Let $X = g_2^x$, and let $\mathcal{A}$ be an algebraic algorithm that makes at most q_H queries to H_{pok} and receives elements of $\mathbb{G}_2$ only of the form $g_2^v X^u$, where an algorithm $\mathcal{E}$ knows u and v for each of them but does not know x. Suppose that $\mathcal{A}$ outputs at most p triples $(\mathsf{ctx}, (C_k)_{k=0}^d, \pi)$, and call such a triple valid if π verifies for $(C_k)_k$ and ctx and H_{pok} was not programmed by a simulator at the point at which this verification evaluates it. Then, except with probability at most $(q_H+p)/q$ over the answers*

of H_{pok}, $\mathcal{E}$ outputs, for every valid triple, either x or exponents $(\alpha_k)_k$ with $C_k = g_2^{\alpha_k}$ for every k. The extractor $\mathcal{E}$ sees the queries of $\mathcal{A}$ to H_{pok} and does not rewind $\mathcal{A}$: it only reads the representations that $\mathcal{A}$ outputs, in time linear in their length.

Proof. Let $\pi = (c, (z_k)_k)$ and $R_k = g_2^{z_k} C_k^{-c}$, so that c is the answer of H_{pok} to the query $(\mathsf{ctx}, (C_k)_k, (R_k)_k)$. We may assume that $\mathcal{A}$ queries H_{pok} at this point before it outputs π: if it does not, we modify it to make this query just before the output, which adds at most p queries in total. Consider the first query at this point. With it, $\mathcal{A}$ outputs a representation of every C_k and R_k, which $\mathcal{E}$ turns into $C_k = g_2^{v_k} X^{u_k}$ and $R_k = g_2^{v'_k} X^{u'_k}$ using the known u and v. Then $g_2^{z_k} = R_k C_k^c$ gives

$$z_k = v'_k + c\, v_k + x\,(u'_k + c\, u_k) \qquad \text{for every } k.$$

If $u'_k + c\, u_k \neq 0$ for some k, $\mathcal{E}$ outputs $x = (z_k - v'_k - c\, v_k)/(u'_k + c\, u_k)$. Otherwise $u'_k + c\, u_k = 0$ for every k. If moreover $u_k = 0$ for every k, then $C_k = g_2^{v_k}$ and $\mathcal{E}$ outputs $\alpha_k = v_k$. In the remaining case, $c = -u'_k/u_k$ for the smallest k with $u_k \neq 0$. The answer c is uniform in $\mathbb{Z}_q$ and is chosen after $\mathcal{A}$ has fixed u_k and u'_k, so for each of the at most $q_H + p$ queries this happens with probability at most $1/q$, and by a union bound with probability at most $(q_H + p)/q$ for any of them. □

3.4. Multi-signatures and Accountable Subgroup Multi-signatures

Definition 3.4. Following Bellare and Neven [8] and Boneh, Drijvers and Neven [16, Section 2.4], a multi-signature scheme consists of the algorithms **Pg**, **Kg**, **Sign**, **KAg** and **Vf**.

- $\mathbf{Pg}(1^\lambda) \to par$: It is run by a trusted party, takes the security parameter λ as input and outputs the public system parameters par, such as the groups, generators and hash functions.
- $\mathbf{Kg}(par) \to (sk, pk)$: It is run by every signer independently, takes the public parameters par as input and outputs a key pair (sk, pk), consisting of a secret key sk and a public key pk.
- $\mathbf{Sign}(par, \mathcal{PK}, sk_i, m) \to \sigma$: It is an interactive protocol run by the signers whose public keys form the set $\mathcal{PK}$. Signer i takes as input the public parameters par, the set $\mathcal{PK}$, its secret key sk_i and the message

m; at the end, every signer outputs the multi-signature σ on m. In the schemes of this paper, each signer computes an individual signature and a combiner aggregates the individual signatures into σ.

- **KAg**$(\mathcal{PK}) \to apk$: It is run by any party, takes the set $\mathcal{PK}$ of public keys of the signers as input and outputs an aggregate public key apk.

- **Vf**$(par, apk, m, \sigma) \to \{0, 1\}$: It is run by a verifier, takes as input the public parameters par, an aggregate public key apk, a message m and a multi-signature σ, and outputs 1 if σ is a valid multi-signature on m under apk, and 0 otherwise.

Boneh, Drijvers and Neven [16, Section 4.1] adapt the syntax of accountable subgroup multi-signatures of [17] by adding an interactive group setup **GSetup**, which is a one-time protocol run by all members of a group. We use their syntax with one change, explained after the definition.

Definition 3.5. An accountable subgroup multi-signature scheme consists of the algorithms **Pg**, **Kg**, **GSetup**, **Sign** and **Vf**. Let $\mathcal{G}$ be a group of n players with public keys $\mathcal{PK} = (pk_1, \ldots, pk_n)$, an ordered list in which pk_i is at position i, where each player has a publicly computable index $i \in \{1, \ldots, n\}$.

- **Pg** and **Kg** are as in Definition 3.4.

- **GSetup**$(par, sk_i, \mathcal{PK}) \to (mk_i, gpk)$: It is an interactive protocol run by all players in $\mathcal{G}$. Player i takes as input the public parameters par, its secret key sk_i and the list $\mathcal{PK}$ of public keys of the group; at the end, player i obtains its membership key mk_i, and the protocol outputs the public data gpk of the group.

- **Sign**$(par, \mathcal{PK}, \mathcal{S}, sk_i, mk_i, m) \to \sigma$: It is an interactive protocol run by the members of a subgroup $\mathcal{S} \subseteq \{1, \ldots, n\}$. Member $i \in \mathcal{S}$ takes as input the public parameters par, the list $\mathcal{PK}$, the subgroup $\mathcal{S}$, its secret key sk_i, its membership key mk_i and the message m; at the end, the protocol outputs the accountable subgroup multi-signature σ on m. As above, each member of $\mathcal{S}$ computes an individual signature and a combiner aggregates them.

- **Vf**$(par, gpk, \mathcal{S}, m, \sigma) \to \{0, 1\}$: It is run by a verifier, takes as input the public parameters par, the public data gpk of a group, a subgroup $\mathcal{S}$,

a message m and a signature σ, and outputs 1 if σ is a valid signature on m by the subgroup $\mathcal{S}$ of the group with public data gpk, and 0 otherwise.

In [16] the verifier receives an aggregate public key $apk \leftarrow \mathbf{KAg}(\mathcal{PK})$. We give it the public data gpk of the group instead, because the vASM verifier needs the membership public keys rather than a single aggregate key. For vASM, $gpk = (\mathcal{PK}, (\mathrm{COM}_j, \pi_j)_{j \in \mathcal{G}})$ consists of the keys and of the dealings that the members broadcast in the group setup, from which the verifier computes the set COM of commitments and the set MPK of membership public keys; for ASMwSA and ASMwCA, $gpk = \mathcal{PK}$, and the verifier computes apk and the subgroup key itself from it. In every scheme, $\mathbf{Vf}$ first checks gpk itself, once per group, and outputs 0 for every signature if this check fails.

Correctness and *unforgeability* are two properties that every accountable subgroup multi-signature scheme should meet. Correctness means that for any subgroup of signers $\mathcal{S} \subseteq \mathcal{G}$ and message m, if the signers $i \in \mathcal{S}$ run the **Sign** protocol with their membership keys mk_i, and follow the protocol honestly, then all of the signers in $\mathcal{S}$ outputs exactly the same valid accountable subgroup multi-signature σ, such that $\mathbf{Vf}(par, gpk, \mathcal{S}, m, \sigma) = 1$. Unforgeability means that it is infeasible for an adversary to forge a valid multi-signature where at least one honest user follows the protocol properly. Unforgeability can be described by the following game.

We follow the model of Boneh, Drijvers and Neven [16]. There is a single honest key pk^*. The adversary controls every other member, may choose their public keys as any function of pk^*, and may set up any number of groups that contain pk^*. With $n - 1$ corrupted members out of n, every rogue-key strategy is available to the adversary.

Unforgeability game. The game is given in Figure 1. In every group setup and signing session that the adversary starts, the challenger plays the honest member at a position ι chosen by the adversary, with $pk_\iota = pk^*$, and $\mathcal{A}$ plays all other positions, including any other position that holds pk^*; $\mathcal{A}$ may schedule the messages arbitrarily (rushing). Every scheme fixes an identifier gid of a group, computed from the public data gpk that **GSetup** outputs. We say that a group gid is *set up* with honest position ι if the honest member at position ι accepted its group setup, that is, if $(\mathsf{gid}, \iota, mk_\iota) \in \mathcal{T}$, and that (gid, m) is *queried* if $(\mathsf{gid}, m) \in \mathcal{Q}$. As in [16], the forgery may be for any public data gpk chosen by the adversary, including data that no group setup

with the honest member produced: the adversary wins if pk^* is in the key list of gpk at a position in $\mathcal{S}$ and **Vf** accepts.

Game $\mathrm{Exp}^{\mathrm{uf}}_{\Pi}(\mathcal{A})$

1. $par \leftarrow \mathbf{Pg}(1^\lambda)$; $(sk^*, pk^*) \leftarrow \mathbf{Kg}(par)$; $\mathcal{T} \leftarrow \emptyset$; $\mathcal{Q} \leftarrow \emptyset$
2. $(gpk, \mathcal{S}, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{GSetup}}, \mathcal{O}_{\mathrm{Sign}}, H}(par, pk^*)$; let gid be the identifier and $\mathcal{PK}$ the key list of gpk
3. **if** $(\mathsf{gid}, m) \in \mathcal{Q}$ **then return** 0
4. **if** there is no position $\iota \in \mathcal{S}$ with $pk_\iota = pk^*$ in $\mathcal{PK}$ **then return** 0
5. **return** $\mathbf{Vf}(par, gpk, \mathcal{S}, m, \sigma)$

Oracle $\mathcal{O}_{\mathrm{GSetup}}(\mathcal{PK}, \iota)$

1. **if** $pk_\iota \neq pk^*$ **then return** $\perp$
2. run **GSetup** with the honest member ι on input $(par, sk^*, \mathcal{PK})$ and with $\mathcal{A}$ at all other positions
3. **if** member ι aborts **then return** $\perp$
4. let gid be the identifier of the group; $\mathcal{T} \leftarrow \mathcal{T} \cup \{(\mathsf{gid}, \iota, mk_\iota)\}$
5. **return** gid

Oracle $\mathcal{O}_{\mathrm{Sign}}(\mathsf{gid}, \iota, m, \mathcal{S})$

1. **if** $\mathcal{T}$ has no entry $(\mathsf{gid}, \iota, mk_\iota)$ **or** $\iota \notin \mathcal{S}$ **then return** $\perp$
2. run **Sign** with the honest member ι on input $(par, \mathcal{PK}, \mathcal{S}, sk^*, mk_\iota, m)$ and with $\mathcal{A}$ as the other members of $\mathcal{S}$
3. $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{gid}, m)\}$
4. **return** the view of member ι in this session

Figure 1: The unforgeability game of accountable subgroup multi-signatures (Definition 3.6). H denotes the random oracles of the scheme, and $\mathcal{PK}$ in $\mathcal{O}_{\mathrm{Sign}}$ is the key set of the group gid.

The freshness condition is on (gid, m) and not on $(\mathsf{gid}, m, \mathcal{S})$: as in [16], signatures are homomorphic in the subgroup. In [16], vASM and ASMwCA one signing query on $(\mathsf{gid}, m, \mathcal{S})$ gives signatures on m for other subgroups

containing ι; in ASMwSA two signing queries on one message for two subgroups suffice to move later signatures from one of them to the other (Remark 6.2). This is not a break, because the honest member did sign m for this group. Requiring only that (gid, m) be fresh, rather than m, also lets the adversary obtain signatures on m in other groups, so the notion is at least as strong as that of [16, Definition 6], which forbids any signing query on m. Otherwise our game follows [16, Section 4.1]; in particular, as there, a forgery counts for every key list that contains pk^*, whether or not the honest member took part in a group setup for it.

Definition 3.6. Let $\Pi = (\mathbf{Pg}, \mathbf{Kg}, \mathbf{GSetup}, \mathbf{Sign}, \mathbf{Vf})$ be an accountable subgroup multi-signature scheme. We say that an adversary $\mathcal{A}$ is a $(\tau, q_G, q_H, q_S, \epsilon)$-forger if it runs in time τ, makes at most q_G group setup queries, q_H random oracle queries and q_S signing queries, and wins the game of Figure 1 with probability at least ϵ. We say that the accountable subgroup multi-signature scheme is $(\tau, q_G, q_H, q_S, \epsilon)$-secure if no such adversary exists.

Remark 3.7 (The game for the schemes of this paper)**.** For vASM the identifier of a group is $\mathsf{gid} = (\mathcal{PK}, \mathrm{COM})$, and for ASMwSA and ASMwCA it is $\mathsf{gid} = apk$. All three schemes abort unless the keys in $\mathcal{PK}$ are distinct (Remark 5.5 shows why vASM needs this check), so in a group that is set up the honest position ι is unique. Since the adversary may output any public data, **Vf** must reject group data to which the honest member did not contribute. For ASMwSA and ASMwCA the honest member receives no message in the group setup, and the setup succeeds exactly when the keys in $\mathcal{PK}$ are distinct and their proofs of knowledge verify; a verifier that checks this once per group accepts exactly the groups that the adversary could set up with one more group setup query. For vASM, gpk contains the dealings (COM_j, π_j) of all members, and the verifier checks each of them once per group (Section 5.1). The aggregate COM alone is not enough, even if the verifier checks $C_0 = \prod_{j \in \mathcal{G}} pk_j$: given only $\mathcal{PK}$, anyone can take this C_0, a random x, $C_1 = \left(g_2^x C_0^{-|\mathcal{S}|}\right)^{1/\sum_{i \in \mathcal{S}} i}$ and $C_k = 1$ for $k \geq 2$; then $\prod_{i \in \mathcal{S}} mpk_i = C_0^{|\mathcal{S}|} C_1^{\sum_{i \in \mathcal{S}} i} = g_2^x$, and $H_0(\mathsf{gid}, m)^x$ is a valid signature for $\mathcal{S}$ on every message m. With the dealings, the contribution at the position of pk^* must carry a valid proof of knowledge for $C_0^{(\iota)} = pk^*$, which only the honest member can produce. In a deployment where many verifiers use the same groups, for example a smart contract, a registration authority may run this check once and register the group; the verifiers then trust this authority,

which is an assumption outside our model. In the first version of this report (2022) [26] the challenger generated all n keys and ran the group setup itself, which rules out rogue keys by construction.

3.5. BLS Signature Scheme

Let e be an efficient, non-degenerate bilinear map, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, in groups ($\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$) and (g_1, g_2) be generators of the group pair $(\mathbb{G}_1, \mathbb{G}_2)$, respectively. Let $H : \{0,1\}^* \to \mathbb{G}_1$ be a hash function which maps any arbitrary binary string onto the group $\mathbb{G}_1$, modeled as a random oracle. BLS signature scheme has three phases, which we give below shortly.

1. Key Generation:

   Pick a random secret key $sk \xleftarrow{\$} \mathbb{Z}_q$, and compute the public key $pk \leftarrow g_2^{sk}$.

2. Signature Generation:

   Compute the signature $\sigma = H(m)^{sk}$, where m is the message.

3. Verification:

   Accept if and only if $e(H(m), pk) = e(\sigma, g_2)$ holds.

The BLS signature was proved to be secure against existential forgery under adaptive chosen message attacks in the random oracle model in [19], under the co-CDH assumption (Definition 3.2); our security reductions follow its proof.

Although the BLS signature scheme was proved in [19] to be secure in the random oracle model, it is not safe to use it as a multi-signature scheme directly because of the "rogue-key" attack [14]. In order to avoid this attack, there are some standard measures, such as either using *proof-of-possession (PoP)* or ensuring that the messages are distinct. Both methods have some advantages and disadvantages. Signing distinct messages hinders users from performing efficient verification [16], and using PoP requires additional verification operations. Maxwell *et al.* [28] consider this inconvenient in Bitcoin, where keys are not registered, whereas Ethereum uses the PoP variant of BLS [30]. In order to eliminate these disadvantages, Boneh, Drijvers and Neven proposed in [16] a multi-signature scheme, called MSP, which is a modified version of the BLS scheme. Moreover, they proposed an accountable subgroup multi-signature (ASM) scheme, a composition of the BLS and MSP schemes.

4. Boneh-Drijvers-Neven ASM Scheme

This section follows the notation given in both [16] and the previous sections. Let $\mathcal{PK} := \{pk_1, \ldots, pk_n\}$ be the set of public keys of the group members of the group $\mathcal{G}$, and let $H_0, H_2 : \{0,1\}^* \to \mathbb{G}_1$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ be the hash functions. The ASM scheme given in [16] can be stated as follows.

1. Key Generation: Each user $i \in \mathcal{G}$ picks a secret key $sk_i \xleftarrow{\$} \mathbb{Z}_q$, and computes the corresponding public key $pk_i \leftarrow g_2^{sk_i}$, where g_2 is a generator of $\mathbb{G}_2$.

2. Group Setup: Each member $i \in \mathcal{G}$ performs group setup by participating in 1-round interactive protocol for $i = 1, 2, \ldots, n$.

   - Computes aggregated public key apk of the group as $apk = \prod_{i=1}^{n} pk_i^{a_i}$, where $a_i = H_1(pk_i, \mathcal{PK})$.
   - Sends $\mu_{ij} = H_2(apk, j)^{a_i sk_i}$ to j-th user for $j = 1, 2, \ldots, n$ and $j \neq i$.
   - After receiving μ_{ji}, computes $\mu_{ii} = H_2(apk, i)^{a_i sk_i}$.
   - The membership key of user i is $mk_i = \prod_{j=1}^{n} \mu_{ji}$.

3. Signature Generation: A signer $i \in \mathcal{G}$ computes its individual signature on the message m
$$s_i = H_0(apk, m)^{sk_i} \cdot mk_i, \tag{2}$$
and sends s_i to the combiner.

4. Signature Aggregation: After receiving the individual signatures of the signers, the combiner first forms the set of signers $\mathcal{S} \subseteq \mathcal{G}$. Then, it computes the aggregated subgroup multi-signature $\sigma = (s, pk)$, where $s = \prod_{i \in \mathcal{S}} s_i$ and $pk = \prod_{i \in \mathcal{S}} pk_i$.

5. Verification: Any verifier who is given $\{par, apk, \mathcal{S}, m, \sigma\}$ can verify the signature $\sigma = (s, pk)$ by checking
$$e\Big(H_0(apk, m), pk\Big) \cdot e\Big(\prod_{j \in \mathcal{S}} H_2(apk, j), apk\Big) \stackrel{?}{=} e(s, g_2). \tag{3}$$

Theorem 4.1 ([16, Theorem 3])**.** *ASM scheme is unforgeable under the ψ-co-CDH problem (Definition 3.3) in the random oracle model. More precisely, ASM scheme is $(\tau, q_G, q_H, q_S, \epsilon)$-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if ψ-co-CDH problem is $(\tau + q_H \cdot \max(\tau_{exp_2^l}, \tau_{exp_1^2}) + q_G(l-1)\tau_{exp_1} + q_S(\tau_{exp_2^l} + \tau_{exp_1}) + 2\tau_{pair} + \tau_{exp_1^3}) \cdot 8q_H^2/((1-(q_S+q_H)/q) \cdot \epsilon) \cdot \ln(8q_H/((1-(q_S+q_H)/q)\epsilon)), (1-(q_S+q_H)/q) \cdot \epsilon/(8q_H))$-hard, where l is the maximum number of signers involved in any group setup, τ_{exp_1} and τ_{exp_2} denote the time required to compute exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $\tau_{exp_1^i}$ and $\tau_{exp_2^i}$ denote the time required to compute i-multi-exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and τ_{pair} denotes the time required to compute a pairing operation.*

ASM scheme of Boneh, Drijvers and Neven was proved to be secure in Theorem 4.1 by [16]. This scheme is a composition of a BLS signature and a group-specific membership key mk_i of the signer $i \in \mathcal{G}$. Namely, the first part $H_0(apk, m)^{sk_i}$ of (2) is a BLS signature on (apk, m) by $|\mathcal{S}|$ signers; on the other hand, the second part mk_i is an MSP signature on (apk, i) by all the members $j \in \mathcal{G}$ for $i = 1, 2, \ldots, n$.

The *proof of possession* (PoP) of the secret keys is also discussed in [16]. The ASM scheme with PoP includes each user's signature on their public keys. The i-th user first chooses a secret key $sk_i \in \mathbb{Z}_q$, then computes $pk_i = g_2^{sk_i}$, and constructs the PoP by $pop_i = H_3(pk_i)^{sk_i}$, where $H_3 : \{0,1\}^* \to \mathbb{G}_1$ for $i = 1, 2, \ldots, n$. Then each user has a secret key sk_i and the public key pair (pk_i, pop_i). In order to compute the aggregated public key (apk) of the group, they first check $e(H_3(pk_i), pk_i) \stackrel{?}{=} e(pop_i, g_2)$, then they compute $Y = \prod_{i \in \mathcal{G}} pk_i$ and $h = H_4(\mathcal{PK})$, where $H_4 : \{0,1\}^* \to \mathbb{Z}_q$ is another hash function. And then, the aggregated public key is $apk = (Y, h)$. The signature generation, aggregation and verification phases are the same as the original ASM scheme.

It is known that using PoP brings additional costs, such as the growth in public key size and extra checks in the verification. In the PoP variant of ASM scheme [16], each user's public key consists of two group elements, and each user computes two extra pairings before computing the aggregated public key apk.

5. ASM scheme with verifiable group setup

In this section, we set a special signing key and its public companion for a multi-signature. First, each user generates their secret and public key

pair independently. Then all users jointly perform a group setup in which they participate in a VSS protocol. At the end of this procedure, each user obtains his membership key and membership public key, which satisfy a common public commitment set generated in the group setup phase.

5.1. vASM: An ASM scheme with VSS based group setup

We give the steps of the vASM scheme below. Let $H_0 : \{0,1\}^* \to \mathbb{G}_1$ and $H_{\mathsf{pok}} : \{0,1\}^* \to \mathbb{Z}_q$ be hash functions modeled as random oracles. H_0 is used only by vASM, with its own domain-separation tag; in particular it is distinct from the hash used for plain BLS signatures under the same keys (Remark 5.4). We assume authenticated private channels between the members and an authenticated broadcast channel for the group setup. Every received public key and commitment must be checked to lie in the prime-order subgroup of $\mathbb{G}_2$ and to differ from the identity element, and every received signature to lie in the prime-order subgroup of $\mathbb{G}_1$, as in the key and signature validation of [30]; we do not repeat these checks below.

1. Key Generation: Each user $i \in \mathcal{G}$ picks a secret key $sk_i \xleftarrow{\$} \mathbb{Z}_q$, and computes the public key $pk_i \leftarrow g_2^{sk_i}$, where g_2 is a generator of $\mathbb{G}_2$.

2. Group Setup: Each user $i \in \mathcal{G}$ proceeds as follows:

   - Checks that the keys in $\mathcal{PK}$ are distinct and that $pk_i = g_2^{sk_i}$, and aborts otherwise, before sending anything (Remark 5.5).
   - Chooses a polynomial $f_i(x) = \alpha_{n-1}^{(i)} x^{n-1} + \ldots + \alpha_1^{(i)} x + \alpha_0^{(i)} \in \mathbb{Z}_q[x]$, where $\alpha_0^{(i)} = sk_i$ and $\alpha_k^{(i)} \xleftarrow{\$} \mathbb{Z}_q$ are chosen uniformly, for $k = 1, \ldots, n-1$.
   - Computes the set of commitments $\mathrm{COM}_i := \{C_k^{(i)} = g_2^{\alpha_k^{(i)}} | k = 0, \ldots, n-1\}$.
   - Computes a proof π_i of knowledge of all $\alpha_k^{(i)}$ (Section 3.3) with context $\mathsf{ctx}_i = (\mathcal{PK}, i)$.
   - Broadcasts (COM_i, π_i) and sends $f_i(j)$ privately to j-th user in $\mathcal{G}$, for $j = 1, \ldots, n$.
   - After receiving $(f_j(i), \mathrm{COM}_j, \pi_j)$,
     - computes the membership key $mk_i = \sum_{j \in \mathcal{G}} f_j(i)$.

– computes $\mathrm{COM} := \{C_k = \prod_{j \in \mathcal{G}} C_k^{(j)} | k = 0, \ldots, n-1\}$.

- Checks, for every $j \in \mathcal{G}$:
  (a) $C_0^{(j)} \stackrel{?}{=} pk_j$
  (b) $g_2^{f_j(i)} \stackrel{?}{=} \prod_{k=0}^{n-1} \left(C_k^{(j)}\right)^{i^k}$
  (c) π_j verifies for $(\mathrm{COM}_j, \mathsf{ctx}_j)$
- If any of (a), (b) or (c) fails for some j, then they abort and blame j. Else, they make COM and set of membership public keys mpk_i's public. Define $\mathrm{MPK} = \{mpk_i\}_{i \in \mathcal{G}}$. Note that these public keys can also be computed by the verifier, i.e. $mpk_i = g_2^{mk_i} = \prod_{k=0}^{n-1} C_k^{i^k}$. The public data of the group is $gpk = (\mathcal{PK}, (\mathrm{COM}_j, \pi_j)_{j \in \mathcal{G}})$: the members publish the dealings that were broadcast, so that a verifier can check them. The group is identified by $\mathsf{gid} = (\mathcal{PK}, \mathrm{COM})$. In an implementation, a collision-resistant digest of $(\mathcal{PK}, \mathrm{COM})$ can serve as gid, so that H_0 does not hash $2n$ group elements for every message; the bound of Theorem 5.6 then increases by the probability of finding a collision of the digest.

3. Signature Generation: A signer $i \in \mathcal{G}$ computes its individual signature $s_i = H_0(\mathsf{gid}, m)^{mk_i}$ on the message m and sends s_i to the combiner. The combiner may check each s_i with $e(s_i, g_2) = e(H_0(\mathsf{gid}, m), mpk_i)$.

4. Signature Aggregation: After receiving the individual signatures of the signers, the combiner first forms the set of signers $\mathcal{S} \subseteq \mathcal{G}$. Then, it computes the aggregated subgroup multi-signature $\sigma = \prod_{i \in \mathcal{S}} s_i$.

5. Verification: Anyone, who is given $\{par, gpk, \mathcal{S}, m, \sigma\}$, can verify the signature σ as follows. Once per group, the verifier checks $gpk = (\mathcal{PK}, (\mathrm{COM}_j, \pi_j)_{j \in \mathcal{G}})$: the keys in $\mathcal{PK}$ are distinct and, for every $j \in \mathcal{G}$, COM_j has n elements, $C_0^{(j)} = pk_j$ and π_j verifies for $(\mathrm{COM}_j, \mathsf{ctx}_j)$ with $\mathsf{ctx}_j = (\mathcal{PK}, j)$. If a check fails, it rejects every signature for this group (Remark 3.7 shows why checking COM alone is not enough). Otherwise it computes COM as in the group setup and $mpk_i = \prod_{k=0}^{n-1} C_k^{i^k}$ for $i = 1, \ldots, n$, and stores them; if MPK is given, it checks it against COM in the same way. It never uses an MPK that is not checked:

otherwise anyone could claim $mpk_i = g_2^{x_i}$ for $i \in \mathcal{S}$ and output $\sigma = H_0(\mathsf{gid}, m)^{\sum_{i\in\mathcal{S}} x_i}$. For each signature it checks

$$e\big(H_0(\mathsf{gid}, m), \prod_{i\in\mathcal{S}} mpk_i\big) \stackrel{?}{=} e(\sigma, g_2). \tag{4}$$

Correctness of the vASM scheme follows from the following equation array.

$$\begin{aligned} e\big(H_0(\mathsf{gid}, m), \prod_{i\in\mathcal{S}} mpk_i\big) &= e(H_0(\mathsf{gid}, m), \prod_{i\in\mathcal{S}} g_2^{mk_i}) \\ &= e(H_0(\mathsf{gid}, m), g_2^{\sum\limits_{i\in\mathcal{S}} mk_i}) \\ &= e(H_0(\mathsf{gid}, m)^{\sum\limits_{i\in\mathcal{S}} mk_i}, g_2) \\ &= e(\prod_{i\in\mathcal{S}} H_0(\mathsf{gid}, m)^{mk_i}, g_2) \\ &= e(\prod_{i\in\mathcal{S}} s_i, g_2) \\ &= e(\sigma, g_2) \end{aligned}$$

5.2. Remarks on vASM

1. Unlike the threshold multi-signatures [7, 11, 12, 13], ASM schemes [16, 17] provide accountability. Further, in ASM schemes, any subgroup $\mathcal{S} \subseteq \mathcal{G}$ can sign a message on behalf of the whole group $\mathcal{G}$, whereas in threshold schemes, only subgroups with a sufficient cardinality can sign. Moreover, one can easily transform an ASM scheme into a threshold scheme by setting the threshold as $|\mathcal{S}|$ [17].

2. Since the membership key mk_i of each group member consists of the shares $f_j(i)$ of the secret key of all group members for $i, j = 1, 2, \ldots, n$, the signature σ authenticates the subgroup $\mathcal{S} \subseteq \mathcal{G}$ and shows that each member of $\mathcal{S}$ is authenticated by the other members of $\mathcal{G}$. Hence, the signature is created by $\mathcal{S}$ on behalf of the whole group $\mathcal{G}$.

3. In the first version of this report (2022) [26] we claimed that the consistency checks (a) and (b) provide proof of possession for each user and force all users to be honest, so that no user can set a special rogue key.

This is not true when the adversary chooses the keys of $n-1$ members: Proposition 5.1 gives a rogue-key attack that passes checks (a) and (b), and Remark 5.2 shows that even a proof of possession of each sk_j does not stop it for $n \geq 3$. Check (c), the proof of knowledge of all coefficients, is what rules out rogue keys. The per-dealer form of checks (a) and (b) costs the same as the aggregate form of the first version, up to a constant, and lets an honest member identify a cheating dealer.

4. We consider the case that the membership public keys, i.e., mpk_i for $i = 1, 2, \ldots, n$, are published. We could also write the verification equation (4) as

$$e\big(H_0(\mathsf{gid}, m), \prod_{k=0}^{n-1} C_k^{\sum_{i \in \mathcal{S}} i^k}\big) \stackrel{?}{=} e(\sigma, g_2).$$

   However, in this case, the verifier would incur the cost of n exponentiations in $\mathbb{G}_2$.

5. If the members in the group $\mathcal{G}$ change, the group setup phase must be reset with new random polynomials f_i for $i \in \mathcal{G}$. Otherwise, any n corrupted users can obtain any user's secret key since the secret polynomials are of degree $n-1$ (see Section 3.2). In order to avoid this vulnerability, the polynomials f_i in the group setup phase of the vASM scheme can be set to degree $d \geq n-1$. In this way, at most $d-n+1$ newcomers can be registered to the group $\mathcal{G}$ without resetting the group setup. On the other hand, this costs extra computational complexity at the group setup phase. The degree cannot be smaller than $n-1$: otherwise the $n-1$ shares received by the other members would already determine f_i and hence $sk_i = f_i(0)$. Here we mean that a member keeps its polynomial f_i: a newcomer then receives one more evaluation of f_i, and together with the $n-1$ evaluations held by the others this gives n points of a polynomial of degree $n-1$, which determine f_i and hence $sk_i = f_i(0)$. This remark is an informal observation outside our security model: a newcomer changes $\mathcal{PK}$ and hence gid, and Theorem 5.6 does not cover groups extended in this way.

6. The membership key mk_i in the vASM scheme is used to sign the message m instead of the secret key sk_i by each member $i \in \mathcal{G}$ so that one can easily check the accountability of the signer at the verification

step. For example, consider the case that Bob has two distinct identities, i.e. his individual identity "*Bob*", and his corporate identity "*CFO of Company X*". Assume that sk_B and mk_B are Bob's secret and the membership keys, respectively. In this case, Bob uses sk_B for spending his own money; besides, he signs by mk_B for spending on behalf of Company X. As this example shows, user i uses their secret key sk_i to sign messages and to participate in any multi-signatures; on the other hand, they participate in the vASM scheme with their membership key mk_i. This separation of roles is safe only if H_0 takes gid as input and is separated from the hash of plain BLS signatures (Proposition 5.3 and Remark 5.4). The separation is one of authority, not of security level: mk_i must be protected like sk_i. If mk_ι leaks, the other $n-1$ members together know $f_j(\iota)$ for every $j \neq \iota$ and the shares $f_\iota(j)$, so they compute $f_\iota(\iota) = mk_\iota - \sum_{j \neq \iota} f_j(\iota)$ and, by Lagrange interpolation, $sk_\iota = f_\iota(0)$ (see (5) in Section 5.3). Leakage of membership keys is outside our security model.

7. The group setup is the joint-Feldman protocol of Pedersen [42] with n-of-n sharing, in which every member deals, extended with the proofs π_i; the key generation of FROST [43] also attaches a proof of knowledge to each dealing, there for the constant term only. Checks (a) and (c) use only broadcast data, so all honest members reach the same result for them. Check (b) uses a private share, so a blamed member cannot be told apart from a false accusation by the others. As in [44], member i then broadcasts a complaint against j, and j answers by broadcasting $f_j(i)$, which every member checks with (b). If j does not answer or the broadcast share fails the check, all members abort and j is excluded from the group; otherwise i uses the broadcast share. A corrupt member can always force an abort of an n-of-n setup, so the complaint round only ensures that the abort is attributed to it. Since every dealing is broadcast, the members that accept compute the same COM and the same gid. After an abort, every member chooses new coefficients $\alpha_k^{(i)}$, $k \geq 1$, for the next run; the proof of Theorem 5.6 also covers the dealings of aborted runs. In a deployment on a blockchain, the chain can serve as the broadcast channel.

5.3. Attacks on the setup without proofs of knowledge

Fix a group of $n \geq 2$ members with a single honest member ι holding sk^*, and write $f^* = f_\iota$ and $\theta = f^*(\iota)$. Let $\Phi = \sum_{j \in \mathcal{G}} f_j$, so that $mk_i = \Phi(i)$,

and for a subgroup $\mathcal{S}$ let $T_{\mathcal{S}} = \sum_{i\in\mathcal{S}} \Phi(i)$; by (4), $H_0(\mathsf{gid}, m)^{T_{\mathcal{S}}}$ is a valid signature on m by $\mathcal{S}$. The adversary sees COM_ι and the $n-1$ shares $f^*(j)$, $j \neq \iota$. Since f^* has degree $n-1$, it is determined by its values at the n points of $[n] = \{1, \ldots, n\}$, and Lagrange interpolation at 0 gives

$$sk^* = L_\iota(0)\,\theta + \sum_{j\neq\iota} L_j(0) f^*(j), \qquad L_\iota(0) = \prod_{j\neq\iota} \frac{j}{j-\iota} \neq 0. \tag{5}$$

Thus, from the adversary's point of view, everything about f^* is a known affine function of the single unknown θ, and g_2^θ is computable from COM_ι.

Proposition 5.1 (Rogue-coefficient attack)**.** *Consider vASM without the proofs π_i (and with either the aggregate or the per-dealer version of checks (a) and (b)). For every $n \geq 2$, every honest position ι and every $\mathcal{S}$ with $\iota \in \mathcal{S}$ and $|\mathcal{S}| \geq 2$, a rushing adversary controlling the other $n-1$ members can complete the setup with the honest member accepting and then output valid signatures for $(\mathsf{gid}, \mathcal{S}, m)$ on any message m of its choice.*

Proof. Fix a corrupt member $j_0 \neq \iota$. Choose a polynomial φ of degree at most $n-1$ with $\varphi(\iota) = 0$ and $\sum_{i\in\mathcal{S}} \varphi(i) = -1$. One exists: take $\varphi(x) = \zeta\, x^\ell (x-\iota)$ with $\ell \leq n-2$ chosen so that $\nu_\ell = \sum_{i\in\mathcal{S}} i^\ell (i-\iota) \neq 0$, which is possible because $\mathcal{S}$ contains a point other than ι: the $|\mathcal{S}|-1$ vectors $(i^\ell)_{i\in\mathcal{S}\setminus\{\iota\}}$, $\ell = 0, \ldots, |\mathcal{S}|-2$, form an invertible Vandermonde matrix, so they span all vectors indexed by $\mathcal{S}\setminus\{\iota\}$, and their inner products with the non-zero vector $(i-\iota)_{i\in\mathcal{S}\setminus\{\iota\}}$ cannot all vanish; then set $\zeta = -\nu_\ell^{-1}$. By (5) the constant term of $\varphi\cdot\theta$ is $\varphi(0)L_\iota(0)^{-1} sk^*$ plus a value known after the honest shares arrive, so the adversary registers $pk_{j_0} = (pk^*)^{\varphi(0)/L_\iota(0)} g_2^w$ for a random w before the setup, and during the setup publishes commitments to $f_{j_0} = \xi + \varphi\cdot\theta$, where ξ is a polynomial with known coefficients whose constant term is adjusted so that $C_0^{(j_0)} = pk_{j_0}$. All commitments are computable from g_2^θ and known scalars. The share $f_{j_0}(\iota) = \xi(\iota)$ does not depend on θ, so the adversary can send it; checks (a) and (b) hold by construction. The other corrupt members

behave honestly. Then

$$T_{\mathcal{S}} = \overbrace{\sum_{i\in\mathcal{S}} f^*(i)}^{\text{honest member } \iota} + \overbrace{\sum_{i\in\mathcal{S}} f_{j_0}(i)}^{\text{corrupt member } j_0} + \overbrace{\sum_{j\neq\iota,j_0}\sum_{i\in\mathcal{S}} f_j(i)}^{\text{other corrupt members}}$$

$$= \Big(\theta + \sum_{i\in\mathcal{S}\setminus\{\iota\}} f^*(i)\Big) + \Big(\sum_{i\in\mathcal{S}} \xi(i) + \theta\sum_{i\in\mathcal{S}} \varphi(i)\Big) + \sum_{j\neq\iota,j_0}\sum_{i\in\mathcal{S}} f_j(i)$$

$$= \sum_{i\in\mathcal{S}\setminus\{\iota\}} f^*(i) + \sum_{i\in\mathcal{S}} \xi(i) + \sum_{j\neq\iota,j_0}\sum_{i\in\mathcal{S}} f_j(i).$$

The second line uses $f^*(\iota) = \theta$ and $f_{j_0} = \xi + \varphi\cdot\theta$. The third line uses $\sum_{i\in\mathcal{S}} \varphi(i) = -1$, so the two terms in θ cancel. Every term in the last line is a scalar the adversary knows: the honest shares $f^*(i)$ for $i \neq \iota$ were sent to corrupt members, and ξ and the f_j with $j \neq \iota, j_0$ were chosen by the adversary. So it computes $T_{\mathcal{S}}$ and outputs $H_0(\mathsf{gid}, m)^{T_{\mathcal{S}}}$, which satisfies (4). □

For $n = 2$, $\iota = 1$ and $\mathcal{S} = \{1, 2\}$ the attack is short. Here $\varphi(x) = 1 - x$ and $L_1(0) = 2$, so the adversary registers $pk_2 = (pk^*)^{1/2} g_2^w$ and uses $f_2(x) = \xi(x) + \theta(1 - x)$. Then $f_2(1) = \xi(1)$ and $T_{\mathcal{S}} = f^*(2) + \xi(1) + \xi(2)$, which it knows.

Remark 5.2 (A proof of possession of sk_j is not enough)**.** If every corrupt key must carry a proof of possession, the adversary can still run the attack for $n \geq 3$ by requiring $\varphi(0) = 0$ as well; then $pk_{j_0} = g_2^w$ is an honestly generated key. The conditions $\varphi(0) = \varphi(\iota) = 0$ and $\sum_{i\in\mathcal{S}} \varphi(i) = -1$ are three linear conditions on the n coefficients of φ; they are solvable whenever $|\mathcal{S}| \leq n - 1$, because evaluations at at most n distinct points are linearly independent on polynomials of degree at most $n - 1$. They are also solvable for $\mathcal{S} = [n]$ when $n \geq 3$. On polynomials χ of degree at most $n-1$, the evaluations at $0, 1, \ldots, n$ satisfy, up to a scalar factor, only the relation $\sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} \chi(i) = 0$, the vanishing of the n-th finite difference. A relation $\eta_0\,\chi(0) + \eta_1\,\chi(\iota) + \eta_2 \sum_{i=1}^{n} \chi(i) = 0$ is therefore either trivial or a non-zero multiple of it, and in the second case the coefficients $(-1)^{n-i}\binom{n}{i}$ would be equal for all $i \in [n]\setminus\{\iota\}$. This set contains two consecutive integers $i, i+1$, unless $n = 3$ and $\iota = 2$; the two coefficients are then equal only if q divides $\binom{n}{i} + \binom{n}{i+1} = \binom{n+1}{i+1}$, which is impossible for a prime $q > n + 1$. For $n = 3$ and $\iota = 2$ the coefficients

at 1 and 3 are 3 and 1. Hence $\eta_2 = 0$, then $\eta_0 = \eta_1 = 0$, and the three conditions are independent. For example, for $n = 3$, $\iota = 1$ and $\mathcal{S} = \{1, 2, 3\}$ one can take $\varphi(x) = (x - x^2)/8$: then $\varphi(0) = \varphi(1) = 0$, $\varphi(2) = -1/4$ and $\varphi(3) = -3/4$, so $\sum_{i \in \mathcal{S}} \varphi(i) = -1$. Only for $n = 2$ does a PoP of sk_j rule out this strategy. What the security proof needs, and what π_i provides, is knowledge of *all* coefficients of f_j. This is why the claim of the first version that checks (a) and (b) "force all users to be honest" is false.

Proposition 5.3 (Shared-hash attack)**.** *Suppose vASM hashes messages with $H_0(m)$ instead of $H_0(\mathsf{gid}, m)$, or uses for H_0 the same hash as plain BLS signing under sk^*. If the adversary controls the other $n - 1$ members of a group* gid *and obtains $H_0(m)^{sk^*}$ (a plain BLS signature) or the honest member's share signature on m in any other group in which it controls the other members, then it forges on m for* gid *and every $\mathcal{S}$ with $\iota \in \mathcal{S}$, even when all corrupt members know their coefficients.*

Proof. From a share signature $H_0(m)^{\theta' + \kappa'}$ in another group, with κ' the sum of the corrupt shares sent there, the adversary obtains $H_0(m)^{\theta'}$ and, by (5) for that group, $H_0(m)^{sk^*}$. Applying (5) for gid in reverse gives $H_0(m)^{\theta}$. Since the adversary knows its own coefficients and the shares $f^*(i)$, $i \neq \iota$, it knows $T_{\mathcal{S}} - \theta$ and outputs $H_0(m)^{\theta} \cdot H_0(m)^{T_{\mathcal{S}} - \theta}$. In the second case, replace $H_0(m)$ by $H_0(\mathsf{gid}, m)$ throughout: a plain BLS signature under sk^* on the input (gid, m) is then $H_0(\mathsf{gid}, m)^{sk^*}$, and the last two steps apply unchanged. □

Remark 5.4 (Domain separation)**.** Proposition 5.3 is why H_0 takes gid as input and must be separated from the hash of any other BLS-type scheme used with the same key. The earlier version suggested that a user may sign personally with sk_i and on behalf of the group with mk_i; this is safe only with such separation.

Both attacks were checked symbolically for $n \leq 5$, numerically in the exponent for $n \leq 7$, and end to end on BLS12-381 with our implementation (Section 9); the scripts are available at `https://github.com/ahmetramazan/vasm-pairing`.

Remark 5.5 (Repeated keys)**.** The context $\mathsf{ctx}_i = (\mathcal{PK}, i)$ binds a proof to a list of keys and a position, not to one run of the setup. If $\mathcal{PK}$ could contain pk^* twice, the following attack would work even with the proofs π_i. Let $n = 3$ and $\mathcal{PK} = (pk^*, pk^*, pk_3)$, where the adversary knows the secret key of pk_3. In a first setup the honest member sits at position 2; it sends its

dealing (COM', π'), with context $(\mathcal{PK}, 2)$, and the shares $f'(1)$ and $f'(3)$ to the corrupt positions. This setup then fails, because the adversary cannot prove knowledge of sk^* at position 1. In a second setup with the same $\mathcal{PK}$ the honest member sits at position 1 and deals f'', and the adversary replays (COM', π') at position 2, together with the share $f'(1)$. Checks (a), (b) and (c) pass, since $C'_0 = pk^* = pk_2$. For $\mathcal{S} = \{1, 2\}$ the only unknown part of $T_\mathcal{S}$ is $f''(1) + f'(2)$, and by (5) its coefficient of sk^* is $L_1(0)^{-1} + L_2(0)^{-1} = 1/3 - 1/3 = 0$. The adversary therefore computes $T_\mathcal{S}$ and forges on every message without a signing query. We ran this attack on BLS12-381 with our implementation. To prevent it, every member of vASM first checks that the keys in $\mathcal{PK}$ are distinct; ASMwSA and ASMwCA include the same check.

5.4. Shamir sharing with a fixed secret

The security proof of vASM uses two standard properties of Shamir secret sharing [36], which we state for the honest member's polynomial.

Lemma 2 (Shamir sharing with a fixed secret)**.** *Let $\iota \in [n]$, let $sk^* \in \mathbb{Z}_q$ be fixed and let f^* have constant term sk^* and uniform coefficients $\alpha_1, \ldots, \alpha_{n-1} \xleftarrow{\$} \mathbb{Z}_q$. Then:*

(i) the shares $(f^*(j))_{j \neq \iota}$ are uniform in $\mathbb{Z}_q^{n-1}$ and independent of sk^*;

(ii) given these shares, f^* is determined by sk^*, and every coefficient of f^*, as well as $\theta = f^*(\iota)$, is an affine function of sk^* whose coefficients are computable from the shares.

Proof. The points $P = \{0\} \cup ([n] \setminus \{\iota\})$ are n distinct elements of $\mathbb{Z}_q$ (as $q > n$), so evaluation on P is a bijection between polynomials of degree at most $n - 1$ and $\mathbb{Z}_q^n$ (the Vandermonde matrix on P is invertible). Fixing the value at 0 to sk^*, the map $(\alpha_1, \ldots, \alpha_{n-1}) \mapsto (f^*(j))_{j \neq \iota}$ is therefore a bijection of $\mathbb{Z}_q^{n-1}$, which gives (i). Lagrange interpolation on P expresses every coefficient of f^*, and $f^*(\iota)$, as $L_0^P(\cdot)\, sk^*$ plus a combination of the shares, which gives (ii). □

5.5. Security

We follow the security reduction of the BLS signature scheme given in [19] with a few modifications to prove the security of the proposed scheme in Section 5.1, in the model of Definition 3.6. First, we give Theorem 5.6 which

states the security reduction of our proposed vASM scheme. We prove security with the non-interactive Schnorr proofs π_i of Section 3.3, for algebraic forgers in the algebraic group model with random oracles. By Lemma 1, the reduction obtains every corrupt member's coefficients as soon as the corrupt member sends its proof, without rewinding, which matters because the adversary may run many group setups.

Theorem 5.6. *Let the proofs π_i be the non-interactive Schnorr proofs of Section 3.3. If co-CDH is (τ', ϵ')-hard, then vASM is $(\tau, q_G, q_H, q_S, \epsilon)$-secure (see Definition 3.6) against existential forgery under adaptive chosen message attacks by algebraic forgers, in the algebraic group model and the random oracle model, for all τ and ϵ satisfying*

$$\begin{aligned} \epsilon &\geq \mathrm{e}(q_S+1)\,\epsilon' + \frac{(q_G+1)\,q_H + n\,(q_G+1)}{q}, \\ \tau &\leq \tau' - O\big(q_H+q_S\big)\tau_{exp_1} - O\big(n^2(q_G+1)\big)\tau_{exp_2} - O\big(n^3 q_G^2\big)\tau_{mul}, \end{aligned}$$

where e *is the base of the natural logarithm, τ_{exp_1} and τ_{exp_2} denote the time required to compute an exponentiation in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and τ_{mul} denotes the time required to compute a multiplication in $\mathbb{Z}_q$.*

As for BLS signatures [19, 45], the reduction loses the factor $\mathrm{e}(q_S+1)$, about $2^{31.4}$ for $q_S = 2^{30}$ signing queries, and the same holds for the other theorems of this paper.

Proof. Let $\mathcal{F}$ be a forger. We build $\mathcal{B}$ that, on a co-CDH instance $(A = g_1^{\alpha}, B_1 = g_1^{\beta}, B_2 = g_2^{\beta})$, outputs $g_1^{\alpha\beta}$. $\mathcal{B}$ simulates the game for $\mathcal{F}$ without knowing β; when $\mathcal{F}$ forges on a pair (gid, m) into whose hash $\mathcal{B}$ embedded A, $\mathcal{B}$ reads off $g_1^{\alpha\beta}$ from the forgery.

Keys. $\mathcal{B}$ sets $pk^* = B_2$, implicitly $sk^* = \beta$.

Random oracles. On a new query (gid, m) to H_0, $\mathcal{B}$ samples $r \xleftarrow{\$} \mathbb{Z}_q$ and a bit ω with $\Pr[\omega = 0] = 1/(q_S+1)$, and answers $H_0(\mathsf{gid}, m) = A^{1-\omega} g_1^r$; this is the biased coin of Coron [45], as in the proof for BLS [19]. Since g_1 generates $\mathbb{G}_1$, which has prime order q, and r is uniform in $\mathbb{Z}_q$, g_1^r is uniform in $\mathbb{G}_1$; multiplying by the fixed element A is a bijection of $\mathbb{G}_1$, so $H_0(\mathsf{gid}, m)$ is uniform in $\mathbb{G}_1$ for either value of ω, so the answers are distributed as in the real game and $\mathcal{F}$'s view is independent of all the bits ω. H_{pok} is answered at random, except at the points that $\mathcal{B}$ programs below.

Group setup. When $\mathcal{F}$ starts a setup for $\mathcal{PK}$ with $pk_\iota = pk^*$, $\mathcal{B}$ samples the honest shares $f^*(j) \xleftarrow{\$} \mathbb{Z}_q$ for $j \neq \iota$; by Lemma 2(i) they are distributed as in the real game. By Lemma 2(ii), each honest coefficient is $\alpha_k^{(\iota)} = u_k\beta + v_k$, and $\theta = u_\theta\beta + v_\theta$, where u_k, v_k are computable from $(f^*(j))_{j\neq\iota}$ and, solving (5) for θ with $sk^* = \beta$, $u_\theta = L_\iota(0)^{-1} \neq 0$ and $v_\theta = -L_\iota(0)^{-1}\sum_{j\neq\iota} L_j(0)f^*(j)$. So $\mathcal{B}$ computes $C_k^{(\iota)} = B_2^{u_k} g_2^{v_k}$, simulates π_ι as in Section 3.3 (aborting if H_{pok} has already been queried at the point it programs), sends $f^*(j)$ to member j and publishes $(\mathrm{COM}_\iota, \pi_\iota)$. It records this dealing, the shares $f^*(j)$ and (u_θ, v_θ) at once, also for setups that later abort. Hence every element of $\mathbb{G}_2$ that $\mathcal{F}$ receives in the game (g_2, B_2 and the honest commitments of all setups) has the form $g_2^v B_2^u$ with u, v known to $\mathcal{B}$, as Lemma 1 requires with $X = B_2$.

When $\mathcal{F}$ sends (COM_j, π_j) for a corrupt j and π_j verifies, $\mathcal{B}$ applies Lemma 1. The proof π_j is valid in the sense of the lemma: every point at which $\mathcal{B}$ programs H_{pok} in a setup for $\mathcal{PK}$ has context $(\mathcal{PK}, \iota)$; because the keys in $\mathcal{PK}$ are distinct, ι is the position of pk^* in $\mathcal{PK}$ and is the same in every setup for $\mathcal{PK}$ (Remark 5.5 shows what goes wrong otherwise); and the verification of π_j evaluates H_{pok} with context $(\mathcal{PK}, j)$, $j \neq \iota$, so never at a programmed point. The lemma then gives either β, in which case $\mathcal{B}$ outputs $A^\beta = g_1^{\alpha\beta}$ and stops, or the coefficients of f_j. $\mathcal{B}$ then performs the honest member's checks. If they pass, it records gid, the shares $f_j(\iota)$ received, all extracted polynomials f_j, and (u_θ, v_θ) for this setup.

Signing. On a query for $(\mathsf{gid}, m, \mathcal{S})$, $\mathcal{B}$ obtains (r, ω) for $H_0(\mathsf{gid}, m)$. If $\omega = 0$ it aborts. Otherwise $H_0(\mathsf{gid}, m) = g_1^r$, and the honest share is $g_1^{r\,mk_\iota}$ with $mk_\iota = \theta + \kappa$, where $\kappa = \sum_{j\neq\iota} f_j(\iota)$ is known; so $\mathcal{B}$ returns $B_1^{ru_\theta} g_1^{r(v_\theta+\kappa)}$, which is exactly the honest share, since $g_1^{r\,mk_\iota} = g_1^{r(u_\theta\beta+v_\theta+\kappa)} = (g_1^\beta)^{ru_\theta} g_1^{r(v_\theta+\kappa)}$ and $B_1 = g_1^\beta$.

Output. Let $\mathcal{F}$ output $(gpk, \mathcal{S}, m, \sigma)$ satisfying the winning condition, with $gpk = (\mathcal{PK}, (\mathrm{COM}_j, \pi_j)_j)$, identifier gid and $pk_\iota = pk^*$ for some $\iota \in \mathcal{S}$, and let ω be the bit for (gid, m); $\mathcal{B}$ aborts unless $\omega = 0$, so that $H_0(\mathsf{gid}, m) = A\, g_1^r$. Since $\mathbf{Vf}$ accepts, the keys in $\mathcal{PK}$ are distinct and every dealing in gpk passes the checks of the verifier. For $j \neq \iota$, the proof π_j has context $(\mathcal{PK}, j)$, which as above is never a programmed point, and Lemma 1 gives β, in which case $\mathcal{B}$ outputs A^β, or f_j. For $j = \iota$, consider the point at which the verification of π_ι evaluates H_{pok}. If $\mathcal{B}$ programmed H_{pok} at this point when it simulated an honest dealing, then the challenge c of π_ι is the programmed one, its first message $(R_k)_k$ is part of the point, and each z_k is

determined by $R_k = g_2^{z_k}(C_k^{(\iota)})^{-c}$; so $(\mathrm{COM}_\iota, \pi_\iota)$ is that honest dealing, and $\mathcal{B}$ uses the shares and (u_θ, v_θ) that it recorded for it, and writes f^* for its polynomial and $\theta = f^*(\iota)$. Otherwise π_ι is valid in the sense of Lemma 1, which gives β or $\alpha_0^{(\iota)}$ with $g_2^{\alpha_0^{(\iota)}} = C_0^{(\iota)} = pk^* = B_2$, that is, β in both cases, and $\mathcal{B}$ outputs A^β. So, unless $\mathcal{B}$ has already succeeded, every f_j with $j \neq \iota$ is known to $\mathcal{B}$ and matches its commitments, so $mpk_i = g_2^{\Phi(i)}$ and $\prod_{i\in\mathcal{S}} mpk_i = g_2^{T_\mathcal{S}}$ with $T_\mathcal{S} = \sum_{i\in\mathcal{S}} \Phi(i)$. The verification equation (4) then reads $e(H_0(\mathsf{gid}, m), g_2)^{T_\mathcal{S}} = e(\sigma, g_2)$, and since e is non-degenerate, $\sigma = (A\, g_1^r)^{T_\mathcal{S}}$. Since $\iota \in \mathcal{S}$, $\sum_{i\in\mathcal{S}} f^*(i) = \theta + \sum_{i\in\mathcal{S}\setminus\{\iota\}} f^*(i)$, and every corrupt f_j is known, so $T_\mathcal{S} = \theta + K$ for a scalar K that $\mathcal{B}$ computes, and $\theta = u_\theta\beta + v_\theta$. So

$$\begin{aligned}\sigma = (A\, g_1^r)^{u_\theta\beta + v_\theta + K} &= A^{u_\theta\beta} \cdot A^{v_\theta+K} \cdot g_1^{r u_\theta \beta} \cdot g_1^{r(v_\theta+K)} \\ &= \left(g_1^{\alpha\beta}\right)^{u_\theta} \cdot A^{v_\theta+K} \cdot B_1^{r u_\theta} \cdot g_1^{r(v_\theta+K)},\end{aligned}$$

using $A^\beta = g_1^{\alpha\beta}$ and $g_1^\beta = B_1$. The last three factors are computable by $\mathcal{B}$, and $u_\theta \neq 0$, so $\mathcal{B}$ outputs

$$g_1^{\alpha\beta} = \left(\sigma \cdot A^{-(v_\theta+K)} \cdot B_1^{-r u_\theta} \cdot g_1^{-r(v_\theta+K)}\right)^{u_\theta^{-1}}.$$

Analysis. Up to the events below, $\mathcal{F}$'s view is distributed exactly as in the real game: keys, shares and commitments by Lemma 2, the answers of H_0 as shown above, and signing answers because they equal the honest shares. Two events can make the simulation fail. First, the simulation of π_ι fails with probability at most q_H/q in each setup (Section 3.3), so with probability at most $q_G q_H/q$ in total. Second, by Lemma 1, applied to at most $(n-1)q_G$ corrupt proofs in the setups and at most n proofs in the output, extraction fails with probability at most $(q_H + n(q_G+1))/q$. Outside these events, if extraction ever gives β, $\mathcal{B}$ succeeds directly. Otherwise $\mathcal{B}$ succeeds if $\mathcal{F}$ wins and $\mathcal{B}$ does not abort. By the freshness condition the forgery is on a pair (gid, m) on which no signature was queried, so $\mathcal{B}$ does not abort if $\omega = 0$ for that pair and $\omega = 1$ for the at most q_S pairs queried to signing. These bits are independent of $\mathcal{F}$'s view, so this happens with probability at least $\frac{1}{q_S+1}\left(1 - \frac{1}{q_S+1}\right)^{q_S} \geq \frac{1}{\mathrm{e}(q_S+1)}$. Hence

$$\epsilon' \;\geq\; \frac{1}{\mathrm{e}(q_S+1)}\left(\epsilon - \frac{(q_G+1)\,q_H + n\,(q_G+1)}{q}\right),$$

and rearranging gives the bound on ϵ in the theorem. $\mathcal{B}$'s overhead is one exponentiation in $\mathbb{G}_1$ per H_0 query, $O(1)$ in $\mathbb{G}_1$ per signing query and for the output, $O(n^2)$ in $\mathbb{G}_2$ per setup for the commitments, the proofs and the checks, and $O(nq_G)$ multiplications in $\mathbb{Z}_q$ to convert each of the $O(n^2)$ representations per setup that it uses. □

Remark 5.7 (Why the algebraic group model)**.** In the random oracle model alone, the known way to extract the coefficients from the non-interactive Schnorr proofs is to fork $\mathcal{F}$ at each corrupt member's H_{pok} query. For a single group setup this works with the generalized forking lemma of Bagherzandi, Cheon and Jarecki [46] applied to the $n-1$ corrupt proofs. For many adaptively scheduled setups, however, rewinding one extraction can invalidate others, and the running time of naive sequential extraction can grow exponentially in the nesting depth. In the algebraic group model the representations given by $\mathcal{F}$ make extraction straight-line (Lemma 1), so $\mathcal{B}$ runs $\mathcal{F}$ once and the bound loses only the factor $\mathrm{e}(q_S+1)$ of the BLS reduction, up to negligible terms. A proof in the random oracle model alone with these proofs remains an open question.

6. Accountable Subgroup Multi-signature Scenarios with Subgroup Authentication

In the previous section, we proposed the vASM scheme with a verifiable group setup. In this section, we propose two more ASM schemes whose group setups are different from the vASM scheme. We slightly modify the group setup method of Boneh et al.'s ASM scheme. In the first one, we use components of a membership key to create a subgroup-specific membership key. In the second one, the users keep a single component secret and send other components to the combiner.

In some cases, a signer $i \in \mathcal{S}$ wants to know other signers $\mathcal{S} \subseteq \mathcal{G}$ in advance. In the ASM scheme in Section 4, any subgroup $\mathcal{S} \subseteq \mathcal{G}$ of signers are authorized to sign any message on behalf of the whole group. Consider that two subgroups make two opposite decisions. Since either of the subgroups signs on behalf of the entire group $\mathcal{G}$, this causes a conflict. In order to avoid such a case, the legal entities could presume a unique authorized signer (CEO, CFO, etc.) in $\mathcal{S}$.

In this section, we consider to replace sk_i with $a_i sk_i$ and mk_i with smk_i in Equation (2) for an identifier a_i and a subgroup-specific membership key

smk_i of the signer $i \in \mathcal{S} \subseteq \mathcal{G}$. Then, we can combine two pairings on the left-hand side of Equation (3). In the following, we describe two scenarios.

6.1. ASMwSA: Accountable Subgroup Multi-signature with Subgroup Authentication

In ASMwSA, we consider the case that the subgroup $\mathcal{S}$ is known before the protocol starts. We discard the interactive protocol in the group setup phase and make simple modifications to the ASM scheme given in Section 4. The ASMwSA is as follows:

1. Key Generation: Identical to the Key Generation in Section 4. In addition, user i publishes with pk_i a non-interactive Schnorr proof π_i of knowledge of sk_i, that is, the proof of Section 3.3 with $d = 0$, $C_0 = pk_i$ and $\alpha_0 = sk_i$, and context $\mathtt{ctx}_i = (\texttt{"ASM-key"}, pk_i)$.

2. Group Setup: Each group member $i \in \mathcal{G}$ computes

   - Checks that the keys in $\mathcal{PK}$ are distinct and that π_j verifies for every $j \in \mathcal{G}$, and aborts otherwise.
   - Aggregated public key $apk = \prod_{i \in \mathcal{G}} pk_i^{a_i}$, where $a_i = H_1(pk_i, \mathcal{PK})$.
   - Components of the membership keys $\mu_{ij} = H_2(apk, j)^{a_i sk_i}$ for $j = 1, \ldots, n$, and stores them.

3. Signature Generation: A signer $i \in \mathcal{G}$ computes its subgroup-specific membership key $smk_i = \prod_{j \in \mathcal{S}} \mu_{ij}$ and individual signature $s_i = H_0(apk, m)^{a_i sk_i} \cdot smk_i$ on the message m and sends s_i to the combiner.

4. Signature Aggregation: After receiving the individual signatures, the combiner computes the aggregated subgroup multi-signature $\sigma = \prod_{i \in \mathcal{S}} s_i$. The combiner does not output spk; the verifier computes it (Remark 6.1).

5. Verification: Anyone who is given $\{par, \mathcal{PK}, \mathcal{S}, m, \sigma\}$ can verify the signature σ as follows. Once per group, the verifier checks that the keys in $\mathcal{PK}$ are distinct and that every proof π_j verifies, and computes and stores $a_j = H_1(pk_j, \mathcal{PK})$, the values $pk_j^{a_j}$ and $apk = \prod_{j \in \mathcal{G}} pk_j^{a_j}$.

For each signature it computes $spk = \prod_{j\in\mathcal{S}} pk_j^{a_j}$, which takes $l-1$ multiplications in $\mathbb{G}_2$, and checks

$$e\left(H_0(apk,m)\cdot\prod_{j\in\mathcal{S}}H_2(apk,j),spk\right)\stackrel{?}{=}e(\sigma,g_2).$$

Note that the correctness property follows from the below equation array.

$$\begin{aligned}
e(\sigma,g_2) &= e\left(\prod_{i\in\mathcal{S}}\left(H_0(apk,m)^{a_i sk_i}\cdot smk_i\right),g_2\right)\\
&= e\left(H_0(apk,m)^{\sum_{i\in\mathcal{S}}a_i sk_i}\cdot\left(\prod_{j\in\mathcal{S}}H_2(apk,j)\right)^{\sum_{i\in\mathcal{S}}a_i sk_i},g_2\right)\\
&= e\left(H_0(apk,m)\cdot\prod_{j\in\mathcal{S}}H_2(apk,j),g_2^{\sum_{i\in\mathcal{S}}a_i sk_i}\right)\\
&= e\left(H_0(apk,m)\cdot\prod_{j\in\mathcal{S}}H_2(apk,j),spk\right)
\end{aligned}$$

6.1.1. Remarks on ASMwSA

1. In the group setup phase of ASMwSA, the signers only compute μ_{ij} and store them, but they do not send those μ_{ij} to anyone. In ASMwSA, the i-th signer multiplies their signature $s_i = H_0(apk,m)^{a_i sk_i}$ with $smk_i = \prod_{j\in\mathcal{S}}\mu_{ij}$, instead of mk_i as in Section 4, which also results in a legitimate aggregated signature. We note that this eliminates 1 round of transmission cost.

2. If the subgroup $\mathcal{S}$ is known in advance, the signers can do without the combiner, by passing a partial aggregated signature from one signer to the next; Appendix A describes this variant and its costs.

3. We note that the user $i\in\mathcal{G}$ may compute their individual signature as

$$s_i = \left(H_0(apk,m)\cdot\prod_{j\in\mathcal{S}}H_2(apk,j)\right)^{a_i sk_i}$$

instead of computing and storing μ_{ij}'s in the group setup phase. Although this reduces the storage costs of the signers, computational cost increases by the extra computations of the hash H_2 at the signature generation of each message.

Remark 6.1 (The verifier computes spk)**.** In the first version of this report (2022) [26] the verifier was given spk. It must instead compute spk itself from $\mathcal{PK}$ and $\mathcal{S}$. Otherwise anyone can pick $x \xleftarrow{\$} \mathbb{Z}_q$, claim $spk = g_2^x$ and output $\sigma = \big(H_0(apk, m) \prod_{j \in \mathcal{S}} H_2(apk, j)\big)^x$, which passes the verification equation for every $\mathcal{S}$ and every m. The signature (s, pk) of the scheme in Section 4 also contains a key pk formed by the combiner, and its verifier does not check pk against $\mathcal{S}$. This is safe there because Equation (3) pairs $\prod_{j \in \mathcal{S}} H_2(apk, j)$ with apk, which the verifier computes, so a valid s must contain the membership keys of the members of $\mathcal{S}$, including that of the honest member. ASMwSA pairs this product with spk instead. This saves one pairing, but makes spk the only link between σ and the keys of the group. The verifier therefore keeps the n values $pk_j^{a_j}$ of a group instead of apk alone, and spends $l - 1$ multiplications in $\mathbb{G}_2$ per verification on spk. The same holds for ASMwCA.

Remark 6.2 (ASMwSA does not bind a signature to $\mathcal{S}$)**.** In ASMwSA the members fix $\mathcal{S}$ before signing, but $H_0(apk, m)$ does not take $\mathcal{S}$ as input, and an individual signature is not bound to $\mathcal{S}$. Write $W_{\mathcal{S}}(m) = H_0(apk, m) \prod_{j \in \mathcal{S}} H_2(apk, j)$, so that $s_i = W_{\mathcal{S}}(m)^{a_i sk_i}$. If the honest member signs some m_0 for two subgroups $\mathcal{S}$ and $\mathcal{S}'$ that contain it, then $\delta = s_\iota(m_0, \mathcal{S}') \cdot s_\iota(m_0, \mathcal{S})^{-1} = \big(\prod_{j \in \mathcal{S}'} H_2(apk, j) \prod_{j \in \mathcal{S}} H_2(apk, j)^{-1}\big)^{a_\iota sk_\iota}$ does not depend on m_0, and for every later message m that the honest member signs for $\mathcal{S}$ only, $s_\iota(m, \mathcal{S}) \cdot \delta$ is its signature for $\mathcal{S}'$. We ran this on BLS12-381 with our implementation. It does not contradict Theorem 6.4, since the honest member did sign m for the group apk, but a signature by $\mathcal{S}'$ does not show that the honest member approved the co-signers $\mathcal{S}'$. Binding $\mathcal{S}$ would require $\mathcal{S}$ in the input of H_0 and a notion of subgroup binding, which we do not define or analyze here.

6.1.2. Security

We follow the security reduction of the MSP signature scheme given in [16] with a few modifications to prove the security of the proposed scheme in Section 6.1. Below, we give Theorem 6.4 which states the security reduction of our proposed ASMwSA scheme. In the model of Definition 3.6 the adversary also chooses the subgroup $\mathcal{S}$ of the forgery. Without the proofs π_i this allows a rogue-key attack on the subgroup key spk. The coefficients a_i do not rule it out: they turn it into a modular subset-sum problem, which becomes feasible for large groups (Proposition 6.3).

Proposition 6.3 (Subset-sum attack without proofs of knowledge)**.** *Consider ASMwSA without the proofs π_i, and a group whose honest member is at position ι, with $pk_\iota = pk^*$. Let the corrupt members choose $pk_j = pk^* \cdot g_2^{y_j}$ with $y_j \xleftarrow{\$} \mathbb{Z}_q$, and let $a_j = H_1(pk_j, \mathcal{PK})$ for every j. If the adversary finds $\mathcal{S}$ with $\iota \in \mathcal{S}$ and $a_\iota + \sum_{j \in \mathcal{S}\setminus\{\iota\}} a_j \equiv 0 \pmod q$, then $spk_\mathcal{S} = g_2^y$ with $y = \sum_{j \in \mathcal{S}\setminus\{\iota\}} a_j y_j$, and $\sigma = \big(H_0(apk, m) \prod_{j \in \mathcal{S}} H_2(apk, j)\big)^y$ is a valid signature for $(\mathcal{S}, m)$ on every message m.*

Proof. The keys are distinct with overwhelming probability, and the group setup of ASMwSA is non-interactive, so the honest member completes it. Then $spk_\mathcal{S} = (pk^*)^{a_\iota} \prod_{j \in \mathcal{S}\setminus\{\iota\}} \big(pk^* g_2^{y_j}\big)^{a_j} = (pk^*)^{a_\iota + \sum_{j \in \mathcal{S}\setminus\{\iota\}} a_j}\, g_2^y = g_2^y$, and the verification equation holds for σ by bilinearity: $e\big(W, g_2^y\big) = e\big(W^y, g_2\big)$ with $W = H_0(apk, m) \prod_{j \in \mathcal{S}} H_2(apk, j)$. □

Finding $\mathcal{S}$ is a modular subset-sum problem on $n-1$ random elements of $\mathbb{Z}_q$, and the adversary may also retry with new keys. Wagner's generalized birthday algorithm [47] with 2^k lists solves it in time about $2^{k+\log_2 q/(k+1)}$ once $n - 1 \geq 2^k \log_2 q/(k+1)$; for $\log_2 q \approx 255$ this is about 2^{67} for $n \approx 510$ ($k = 3$) and about 2^{39} for $n \approx 4\,100$ ($k = 7$). For the same reason, the subset version of the key aggregation of [16] needs a random modular subset-sum assumption and the algebraic group model in [31]. The same issue is the gap in the proof of the first version: after the fork, *apk* changes and the forger may output a different subgroup, so the contribution of the corrupt members does not cancel. The proofs π_i remove both problems: in the algebraic group model the reduction extracts every corrupt secret key without rewinding (Lemma 1) and needs no forking. With the proofs, the coefficients a_i are no longer needed for security; we keep them so that the scheme stays close to [16], and setting $a_i = 1$ saves n exponentiations in $\mathbb{G}_2$. Boneh, Drijvers and Neven take the same route in their ASM-pop scheme, which replaces the coefficients by BLS proofs of possession [48, Section 6.2]; they prove ASM-pop under the ψ-co-CDH assumption [48, Theorem 6]. A variant of ASMwSA with BLS proofs of possession instead of π_i would need a separate analysis, which we do not give here. A group is identified by $\mathsf{gid} = apk$, and H_0, H_1, H_2 and H_{pok} use distinct domain-separation tags.

Theorem 6.4. *Let the proofs π_i be the non-interactive Schnorr proofs of Section 3.3. If co-CDH is (τ', ϵ')-hard, then ASMwSA is $(\tau, q_G, q_H, q_S, \epsilon)$-secure (see Definition 3.6) against algebraic forgers, in the algebraic group*

model and the random oracle model, for all τ and ϵ satisfying

$$\epsilon \;\geq\; \mathrm{e}(q_S+1)\,\epsilon' + \frac{2q_H + n\,(q_G+1)}{q},$$
$$\tau \;\leq\; \tau' - O\big(q_H+q_S\big)\tau_{exp_1} - O\big(nq_G\big)\tau_{exp_2} - O\big(nq_G\big)\tau_{mul},$$

where n is the maximum number of members of a group and τ_{exp_1}, τ_{exp_2} and τ_{mul} are as in Theorem 5.6.

Proof. Let $\mathcal{F}$ be a forger. We build $\mathcal{B}$ that, on a co-CDH instance $(A = g_1^\alpha, B_1 = g_1^\beta, B_2 = g_2^\beta)$, outputs $g_1^{\alpha\beta}$. As in the proof of Theorem 5.6, $\mathcal{B}$ simulates the game for $\mathcal{F}$ without knowing β, and reads off $g_1^{\alpha\beta}$ from a forgery on a pair (apk, m) into whose hash it embedded A.

Keys and random oracles. $\mathcal{B}$ sets $pk^* = B_2$, implicitly $sk^* = \beta$. Before it runs $\mathcal{F}$, it simulates the proof π^* for pk^* as in Section 3.3; since no query has been made yet, programming H_{pok} cannot fail, and the simulated proof is distributed exactly as an honest one. It answers H_1 at random, $H_2(apk, j) = g_1^{r_j}$ with $r_j \xleftarrow{\$} \mathbb{Z}_q$, and $H_0(apk, m) = A^{1-\omega} g_1^r$ with $r \xleftarrow{\$} \mathbb{Z}_q$ and a coin ω with $\Pr[\omega = 0] = 1/(q_S+1)$, as in the proof of Theorem 5.6. All these answers are uniform, as in the real game, and $\mathcal{F}$'s view is independent of the coins ω.

Group setup. When $\mathcal{F}$ starts a group setup for $\mathcal{PK}$ with $pk_\iota = pk^*$, $\mathcal{B}$ applies Lemma 1 with $X = B_2$ to the proof π_j of every corrupt key pk_j in $\mathcal{PK}$. The lemma applies: the only elements of $\mathbb{G}_2$ that $\mathcal{F}$ receives are g_2 and B_2, and π_j is valid in the sense of the lemma, because its verification evaluates H_{pok} with context $(\texttt{"ASM-key"}, pk_j)$, the keys in $\mathcal{PK}$ are distinct, so $pk_j \neq pk^*$, and the only programmed point has context $(\texttt{"ASM-key"}, pk^*)$. The lemma gives either β, in which case $\mathcal{B}$ outputs $A^\beta = g_1^{\alpha\beta}$ and stops, or sk_j. The honest member's setup sends nothing to $\mathcal{F}$.

Signing. On a query for $(apk, m, \mathcal{S})$, $\mathcal{B}$ obtains (r, ω) for $H_0(apk, m)$. If $\omega = 0$ it aborts. Otherwise it returns $s_\iota = B_1^{a_\iota(r+\bar{r}_\mathcal{S})}$, where $\bar{r}_\mathcal{S} = \sum_{j\in\mathcal{S}} r_j$. This is the honest individual signature, since $s_\iota = H_0(apk, m)^{a_\iota\beta} \prod_{j\in\mathcal{S}} \mu_{\iota j} = \big(H_0(apk, m) \prod_{j\in\mathcal{S}} H_2(apk, j)\big)^{a_\iota\beta} = \big(g_1^{r+\bar{r}_\mathcal{S}}\big)^{a_\iota\beta}$.

Output. Let $\mathcal{F}$ output $(\mathsf{gid}, \mathcal{S}, m, \sigma)$ satisfying the winning condition: pk^* is at a position $\iota \in \mathcal{S}$ of a key list $\mathcal{PK}$ with $\mathsf{gid} = apk$ whose keys are distinct and whose proofs verify, (apk, m) is fresh, and σ is valid for the subgroup key $spk = \prod_{j\in\mathcal{S}} pk_j^{a_j}$ that the verifier computes from $\mathcal{PK}$. $\mathcal{B}$ aborts unless $\omega = 0$ for (apk, m), so that $W = H_0(apk, m) \prod_{j\in\mathcal{S}} H_2(apk, j) = A\, g_1^{r+\bar{r}_\mathcal{S}}$. If $\mathcal{F}$ made no setup query for $\mathcal{PK}$, $\mathcal{B}$ now applies Lemma 1 to the proofs of $\mathcal{PK}$

as in a setup query, which gives β or every corrupt sk_j. Since $\mathcal{B}$ thus knows every corrupt sk_j of $\mathcal{PK}$, $spk = g_2^{a_\iota\beta+K_\mathcal{S}}$ with $K_\mathcal{S} = \sum_{j\in\mathcal{S}\setminus\{\iota\}} a_j sk_j$ known to $\mathcal{B}$. The verification equation reads $e(W, g_2)^{a_\iota\beta+K_\mathcal{S}} = e(\sigma, g_2)$, and since e is non-degenerate, $\sigma = W^{a_\iota\beta+K_\mathcal{S}}$. Expanding,

$$\sigma = A^{a_\iota\beta} \cdot g_1^{(r+\bar{r}_\mathcal{S})a_\iota\beta} \cdot A^{K_\mathcal{S}} \cdot g_1^{(r+\bar{r}_\mathcal{S})K_\mathcal{S}} = \left(g_1^{\alpha\beta}\right)^{a_\iota} \cdot B_1^{a_\iota(r+\bar{r}_\mathcal{S})} \cdot A^{K_\mathcal{S}} \cdot g_1^{(r+\bar{r}_\mathcal{S})K_\mathcal{S}},$$

so, if $a_\iota \neq 0$, $\mathcal{B}$ outputs

$$g_1^{\alpha\beta} = \left(\sigma \cdot A^{-K_\mathcal{S}}\, g_1^{-(r+\bar{r}_\mathcal{S})K_\mathcal{S}} \cdot B_1^{-a_\iota(r+\bar{r}_\mathcal{S})}\right)^{a_\iota^{-1}}.$$

Analysis. Up to the events below, $\mathcal{F}$'s view is distributed exactly as in the real game. By Lemma 1, applied to at most $(n-1)q_G$ corrupt proofs in the setups and at most n proofs in the output, extraction fails with probability at most $(q_H + n(q_G+1))/q$; if it ever gives β, $\mathcal{B}$ succeeds directly. Some answer of H_1 is 0, so that possibly $a_\iota = 0$, with probability at most q_H/q. Otherwise $\mathcal{B}$ succeeds if $\mathcal{F}$ wins and $\mathcal{B}$ does not abort, that is, if $\omega = 0$ for the pair of the forgery, which is fresh, and $\omega = 1$ for the at most q_S pairs queried to signing. As in the proof of Theorem 5.6, this happens with probability at least $1/(\mathrm{e}(q_S+1))$, independently of $\mathcal{F}$'s view. Hence

$$\epsilon' \;\geq\; \frac{1}{\mathrm{e}(q_S+1)}\left(\epsilon - \frac{2q_H + n\,(q_G+1)}{q}\right),$$

and rearranging gives the bound on ϵ in the theorem. $\mathcal{B}$'s overhead is one exponentiation in $\mathbb{G}_1$ per hash query and per signing query, $O(1)$ in $\mathbb{G}_1$ for the output, $O(n)$ in $\mathbb{G}_2$ per group setup to compute apk and verify the proofs, and $O(n)$ multiplications in $\mathbb{Z}_q$ per setup to read the representations, which have length 2. □

6.2. ASMwCA: Accountable Subgroup Multi-signature with Combiner Authentication

In ASMwCA, each user $i \in \mathcal{G}$ sends the membership key components, i.e. μ_{ij} for $j \neq i$, to the combiner so that the signers perform fewer computations, and the main workload passes to the combiner. The ASMwCA is as follows:

1. Key Generation: Identical to Key Generation in Section 4. In addition, user i publishes with pk_i a non-interactive Schnorr proof π_i of knowledge of sk_i, that is, the proof of Section 3.3 with $d = 0$, $C_0 = pk_i$ and $\alpha_0 = sk_i$, and context $\mathsf{ctx}_i = (\texttt{"ASM-key"}, pk_i)$.

2. Group Setup: Each group member $i \in \mathcal{G}$ computes:

   - Checks that the keys in $\mathcal{PK}$ are distinct and that π_j verifies for every $j \in \mathcal{G}$, and aborts otherwise.
   - The aggregated public key apk of $\mathcal{G}$ as $apk = \prod_{i \in \mathcal{G}} pk_i^{a_i}$, where $a_i = H_1(pk_i, \mathcal{PK})$.
   - $\mu_{ii} = H_2(apk, i)^{a_i sk_i}$ and stores it.
   - $\mu_{ij} = H_2(apk, j)^{a_i sk_i}$ for $j = 1, \ldots, n$ and $j \neq i$, and sends them to the combiner.

3. Signature Generation: A signer $i \in \mathcal{G}$ computes individual signature $s_i = H_0(apk, m)^{a_i sk_i} \mu_{ii}$ on the message m and sends s_i to the combiner.

4. Signature Aggregation: After receiving the individual signatures of the signers, the combiner first forms $\mathcal{S} \subseteq \mathcal{G}$. Then, it computes the aggregated subgroup multi-signature $\sigma = \prod_{i \in \mathcal{S}} s_i \cdot \prod_{i \in \mathcal{S}} \prod_{j \in \mathcal{S} \setminus \{i\}} \mu_{ij}$. As in ASMwSA, the combiner does not output spk; the verifier computes it (Remark 6.1).

5. Verification: Anyone who is given $\{par, \mathcal{PK}, \mathcal{S}, m, \sigma\}$ can verify signature σ as in ASMwSA. Once per group it checks the keys in $\mathcal{PK}$ and their proofs π_j and computes apk and the values $pk_j^{a_j}$; for each signature it computes $spk = \prod_{j \in \mathcal{S}} pk_j^{a_j}$ and checks

$$e\left(H_0(apk, m) \cdot \prod_{j \in \mathcal{S}} H_2(apk, j), spk\right) \stackrel{?}{=} e(\sigma, g_2).$$

Since ASMwCA is a modified version of ASMwSA, its correctness property follows from the correctness of the ASMwSA schemes as shown in Section 6.1.

6.2.1. Remarks on ASMwCA

1. The subgroup $\mathcal{S} \subseteq \mathcal{G}$ of the signers is determined by the combiner from the set of received individual signatures. Hence, no signer knows their co-signers.

2. In the signature generation phase, $H_0(apk, m)$ may be replaced by $H_0(spk, m)$ so that the signer could set the subgroup $\mathcal{S}$ in advance. This also eliminates the combiner's corruption at the signature aggregation phase. Namely, the combiner cannot discard the signature s_i of any user $i \in \mathcal{S}$ from σ. However, in this case, computing spk brings additional cost, i.e. one l-term multi-exponentiation in $\mathbb{G}_2$ for each signer. Theorem 6.5 covers only the scheme with $H_0(apk, m)$; we do not analyze this variant.

3. Each user $i \in \mathcal{G}$ sends μ_{ij} to the combiner for $i, j = 1, 2, \ldots, n$ and $j \neq i$. Unlike the scenarios in Sections 4 and 6.1, each signer $i \in \mathcal{G}$ does not compute the membership key, but uses only μ_{ii} for the signature generation. The other components, μ_{ij} for $i, j = 1, 2, \ldots, n$ and $j \neq i$, are taken into account by the combiner as it forms the subgroup $\mathcal{S} \subseteq \mathcal{G}$. Therefore, each user's computational cost is reduced, but the workload of the combiner is increased by the number of signers.

4. We note that the user $i \in \mathcal{G}$ may compute their individual signature as

$$s_i = \Big(H_0(apk, m) H_2(apk, i) \Big)^{a_i sk_i}$$

instead of computing and storing μ_{ii} in the group setup phase. However, this costs extra computation of the hash H_2 at the signature generation of each message.

6.2.2. Security

Here the combiner is controlled by the adversary. The security reduction of the following theorem is that of Theorem 6.4 with two changes, given in the proof. The proofs π_i are needed here for the same reason: ASMwCA computes spk and verifies signatures as ASMwSA does, so the attack of Proposition 6.3 applies to ASMwCA without the proofs π_i unchanged.

Theorem 6.5. *Let the proofs π_i be as in Theorem 6.4. If co-CDH is (τ', ϵ')-hard, then ASMwCA, with the combiner controlled by the adversary, is $(\tau, q_G, q_H, q_S, \epsilon)$-secure (see Definition 3.6) against algebraic forgers, in the algebraic group model and the random oracle model, for all τ and ϵ*

satisfying

$$\epsilon \;\geq\; \mathrm{e}(q_S+1)\,\epsilon' + \frac{2q_H + n\,(q_G+1)}{q},$$
$$\tau \;\leq\; \tau' - O\big(q_H+q_S+nq_G\big)\tau_{exp_1} - O\big(nq_G\big)\tau_{exp_2} - O\big(nq_G\big)\tau_{mul},$$

where n is the maximum number of members of a group and τ_{exp_1}, τ_{exp_2} and τ_{mul} are as in Theorem 5.6.

Proof. In the first version we assumed without loss of generality that the reduction is the combiner. This is not enough, because the combiner receives the honest member's components $\mu_{\iota j}$, which the forger can use. We therefore give the reduction. It is the reduction $\mathcal{B}$ of Theorem 6.4, with two changes, both in what the honest member sends to the adversarial combiner.

Group setup. The honest member must send $\mu_{\iota j} = H_2(apk, j)^{a_\iota \beta}$ for every $j \neq \iota$. Since $\mathcal{B}$ answers $H_2(apk, j) = g_1^{r_j}$ with r_j known, it sends $B_1^{a_\iota r_j} = (g_1^{r_j})^{a_\iota \beta}$, which is exactly this value. The extraction of the corrupt secret keys is as in Theorem 6.4.

Signing. On a query with $\omega = 1$, the honest individual signature is $s_\iota = H_0(apk, m)^{a_\iota \beta}\mu_{\iota\iota} = \big(H_0(apk, m)H_2(apk, \iota)\big)^{a_\iota \beta} = \big(g_1^{r+r_\iota}\big)^{a_\iota \beta}$, so $\mathcal{B}$ returns $B_1^{a_\iota(r+r_\iota)}$. On $\omega = 0$ it aborts.

The values that the forger receives and that depend on β are therefore powers of B_1 with exponents known to $\mathcal{B}$, and the only elements of $\mathbb{G}_2$ it receives are still g_2 and B_2, so Lemma 1 applies as before. A forgery is checked with the same verification equation and the same subgroup key *spk* as in ASMwSA, so the output step of Theorem 6.4 computes $g_1^{\alpha\beta}$ from it unchanged. The probability analysis is therefore that of Theorem 6.4; the additional components cost $n-1$ exponentiations in $\mathbb{G}_1$ per setup. □

6.3. Advantages and costs of subgroup-specific membership keys

1. Group-specific membership keys in Boneh-Drijvers-Neven ASM scheme given in Section 4 are generated by one round of interactive protocol, in which all members participate. But in case of using subgroup-specific ones given in Sections 6.1 and 6.2, no interactive protocol is needed: in ASMwSA the members send nothing, and in ASMwCA each member sends one message to the combiner. The proofs π_i are attached to the public keys once and are reused in every group.

2. Even if one user registers/unregisters to/from the group, the apk and the membership key mk need to be recomputed. This means a new interactive protocol to be held by all the group members again, which is costly for large $|\mathcal{G}|$. Consider the case of using the subgroup-specific membership key smk instead of group-specific one, i.e. mk_i. If we use $\mu_{ij} = H_2(pk_j)^{sk_i}$, that is, $H_2(pk_j)$ instead of $H_2(apk, j)$ and $a_i = 1$, registering/unregistering a member to/from the group $\mathcal{G}$ does not require a new group setup. Both changes are needed, since $a_i = H_1(pk_i, \mathcal{PK})$ changes whenever $\mathcal{PK}$ does. Setting $a_i = 1$ is possible because the proofs π_i, not the coefficients, rule out rogue keys (see the discussion after Proposition 6.3). The group must then be identified by a collision-resistant digest of $\mathcal{PK}$ rather than by $apk = \prod_{j \in \mathcal{G}} pk_j$, because an adversary who knows the secret keys of its members easily finds two key sets with the same product. With these changes the proof of Theorem 6.4 still applies, because $\mathcal{B}$ answers $H_2(pk_j) = g_1^{r_j}$ with known r_j and H_0 takes the group identifier as input.

3. The membership key mk_i of the ASM scheme in Section 4 is computed once per group, whereas smk_i is computed for every signature, with $|\mathcal{S}| - 1$ multiplications in $\mathbb{G}_1$, which are negligible next to the exponentiations of signing. The advantage of smk is therefore the non-interactive setup, not a lower signing cost. Similarly, apk is an n-term multi-exponentiation computed once per group, and once the values $pk_j^{a_j}$ are stored, spk takes only $|\mathcal{S}| - 1$ multiplications in $\mathbb{G}_2$ per signature.

4. The costs are the following. In ASMwSA the subgroup $\mathcal{S}$ must be fixed before signing, and in ASMwCA the combiner holds the components μ_{ij} and chooses $\mathcal{S}$. Each member of ASMwSA stores n components μ_{ij} instead of one membership key, and the combiner of ASMwCA stores $n(n-1)$ of them. The setup computation is the same as in the ASM scheme of Section 4, where each member also computes n components but sends $n - 1$ of them to the other members.

7. Aggregated versions of ASM schemes

In this section, we extend the ASM scheme to a partial aggregated version of ASM, i.e. AASM scheme. Remember that the ASM signature scheme described in Section 4 outputs a signature $\sigma = (s, pk)$. Consider N many ASM

signatures $(apk_1, \mathcal{S}_1, m_1, \sigma_1), \ldots, (apk_N, \mathcal{S}_N, m_N, \sigma_N)$. The AASM scheme outputs the partial aggregated signature $\Sigma = (pk_1, \ldots, pk_N, s)$, where s is the aggregation of $s_1, \ldots, s_N$. In Section 7.1, we describe the AASM scheme that is given in [16, Section 4.4].

7.1. Partially Aggregated ASM Scheme (AASM)

The aggregation of several ASM signatures is discussed in [16], in which they aggregate only the second component of the signature. The first components are used in the verification phase. On the other hand, it is noted that the ASM scheme cannot be partially aggregated directly because the membership keys do not depend on the messages [16]. In addition to the phases of the ASM scheme, two more phases are defined in [16]. The signature aggregation and aggregated signature verification phases are as follows. Let $H_5 : \{0,1\}^* \to \mathbb{Z}_q$ be another hash function.

- Signature Aggregation: Given $(par, \{(apk_i, \mathcal{S}_i, m_i, \sigma_i)_{i=1}^N\})$.
  - Parse σ_i as (s_i, pk_i).
  - For $i = 1, \ldots, N$, compute
$$b_i \leftarrow H_5\big((apk_i, \mathcal{S}_i, m_i, pk_i), \{(apk_j, \mathcal{S}_j, m_j, pk_j)_{j=1}^N\}\big).$$
  - Compute $s \leftarrow \prod_{i=1}^{N} s_i^{b_i}$ and output $\Sigma \leftarrow (pk_1, \ldots, pk_N, s)$.
- Aggregate Signature Verification: Given $(par, \{(apk_i, \mathcal{S}_i, m_i)_{i=1}^N\}, \Sigma)$.
  - Parse Σ as $(pk_1, \ldots, pk_N, s)$.
  - For $i = 1, \ldots, N$, compute
$$b_i \leftarrow H_5\big((apk_i, \mathcal{S}_i, m_i, pk_i), \{(apk_j, \mathcal{S}_j, m_j, pk_j)_{j=1}^N\}\big).$$
  - Accept if and only if
$$\prod_{i=1}^{N} \left(e\big(H_0(apk_i, m_i), pk_i^{b_i}\big) \cdot e\Big(\prod_{j \in \mathcal{S}_i} H_2(apk_i, j), apk_i^{b_i}\Big) \right) \stackrel{?}{=} e(s, g_2). \tag{6}$$

Since the AASM scheme cannot be fully aggregated, the size of the resulting partial aggregated signature grows linearly by the number of ASM signatures. Besides, the verification (6) of the AASM scheme requires $(2N + 1)$ pairings.

7.2. Aggregated versions of proposed schemes

Our schemes defined in Sections 5.1, 6.1 and 6.2 can also be further turned into aggregated schemes. Unlike the partial aggregation in the AASM scheme, our proposed schemes can be fully aggregated, resulting in a single signature size.

For the aggregated schemes we extend Definition 3.6 as follows. As in [16, Section 2.5], an aggregated scheme adds two algorithms.

- $\mathbf{SAg}(par, \{(gpk_k, \mathcal{S}_k, m_k, \sigma_k)\}_{k=1}^N) \to \Sigma$: It is run by any party, takes as input the public parameters par and N tuples, each consisting of the public data gpk_k of a group, a subgroup $\mathcal{S}_k$, a message m_k and a signature σ_k on m_k by $\mathcal{S}_k$, and outputs one aggregate signature Σ.

- $\mathbf{AVf}(par, \{(gpk_k, \mathcal{S}_k, m_k)\}_{k=1}^N, \Sigma) \to \{0, 1\}$: It is run by a verifier, takes as input the public parameters par, N tuples $(gpk_k, \mathcal{S}_k, m_k)$ and an aggregate signature Σ, and outputs 1 if Σ is a valid aggregate signature on these tuples, and 0 otherwise.

Definition 7.1. The game is that of Figure 1, with the same oracles, except for the output: the adversary outputs N tuples $(gpk_k, \mathcal{S}_k, m_k)$, where gpk_k is the public data of a group gid_k, and an aggregate signature Σ. It wins if $\mathbf{AVf}(par, \{(gpk_k, \mathcal{S}_k, m_k)\}_{k=1}^N, \Sigma) = 1$ and there is a k^* such that pk^* is in the key list of gpk_{k^*} at a position $\iota \in \mathcal{S}_{k^*}$ and $(\mathsf{gid}_{k^*}, m_{k^*})$ has not been queried. As in Figure 1, all public data are chosen by the adversary, and **AVf** checks each gpk_k as **Vf** does. An adversary is a $(\tau, q_G, q_H, q_S, N, \epsilon)$-forger if it is a $(\tau, q_G, q_H, q_S, \epsilon)$-forger in this game and outputs at most N tuples, and the scheme is $(\tau, q_G, q_H, q_S, N, \epsilon)$-secure if no such forger exists.

7.2.1. Aggregated vASM Scheme (AvASM)

The vASM scheme described in Section 5.1 can be extended to an aggregated version AvASM, with the algorithms SAg and AVf defined above and without the exponents b_i of the AASM scheme (Remark 7.2). In AvASM the public data of a group is $gpk_k = (\mathcal{PK}_k, (\mathrm{COM}_j^{(k)}, \pi_j^{(k)})_j)$, as in vASM, with identifier $\mathsf{gid}_k = (\mathcal{PK}_k, \mathrm{COM}_k)$, and $mpk_j^{(k)}$ and $mk_j^{(k)}$ denote the membership public key and the membership key of member j of the group gid_k.

- $\mathbf{SAg}(par, \{(gpk_k, \mathcal{S}_k, m_k, \sigma_k)\}_{k=1}^N) \to \Sigma$: It is run by any party, takes as input the public parameters par and N tuples, each consisting of the public data gpk_k of a group, a subgroup $\mathcal{S}_k$, a message m_k and a

vASM signature σ_k on m_k by $\mathcal{S}_k$, and outputs the aggregate signature $\Sigma = \prod_{k=1}^{N} \sigma_k$.

- $\mathbf{AVf}(par, \{(gpk_k, \mathcal{S}_k, m_k)\}_{k=1}^{N}, \Sigma) \to b \in \{0,1\}$: It is run by a verifier, takes as input the public parameters par, N tuples $(gpk_k, \mathcal{S}_k, m_k)$ and an aggregate signature Σ, checks every gpk_k and computes MPK_k from it once per group, as the vASM verifier does, outputs $b = 0$ if a check fails, and outputs $b = 1$ if and only if

$$\prod_{k=1}^{N} e\left(H_0(\mathsf{gid}_k, m_k), \prod_{j \in \mathcal{S}_k} mpk_j^{(k)} \right) = e(\Sigma, g_2). \tag{7}$$

If MPK_k were taken from the input, a forger could add a tuple with the same $\mathcal{PK}_k$ and message but a fake MPK that cancels the contribution of the honest member.

Correctness of the AvASM scheme follows from the following equation array.

$$\begin{aligned}
\prod_{k=1}^{N} e\left(H_0(\mathsf{gid}_k, m_k), \prod_{j \in \mathcal{S}_k} mpk_j^{(k)} \right) &= \prod_{k=1}^{N} e\left(H_0(\mathsf{gid}_k, m_k), \prod_{j \in \mathcal{S}_k} g_2^{mk_j^{(k)}} \right) \\
&= \prod_{k=1}^{N} e\left(H_0(\mathsf{gid}_k, m_k), g_2^{\sum_{j \in \mathcal{S}_k} mk_j^{(k)}} \right) \\
&= \prod_{k=1}^{N} e\left(H_0(\mathsf{gid}_k, m_k)^{\sum_{j \in \mathcal{S}_k} mk_j^{(k)}}, g_2 \right) \\
&= \prod_{k=1}^{N} e\big(\sigma_k, g_2\big) \\
&= e\left(\prod_{k=1}^{N} \sigma_k, g_2 \right) \\
&= e(\Sigma, g_2).
\end{aligned}$$

Remark 7.2 (No randomization exponents)**.** The AASM scheme of [16] randomizes the aggregation with exponents b_i; AvASM does not. Without them, an adversary could try to make two tuples cancel in (7). Cancellation needs two tuples with the same hash value in $\mathbb{G}_1$, that is, the same (gid, m).

The exponents of such tuples add up, so the honest member's contribution appears with multiplicity at least one, from the target tuple, and cannot cancel. Tuples with different (gid, m) have independent random hash values, and relating them is a discrete-logarithm problem. This is also why the attack that forces [16, Section 4.4] to use the exponents b_i does not apply. There, an ASM signature consists of a part on the message under the key of $\mathcal{S}$ and a part on the membership keys under apk, which are not linked, so an aggregate for $\{(apk, \mathcal{S}_1, m_1), (apk, \mathcal{S}_2, m_2)\}$ is also valid for $\{(apk, \mathcal{S}_1, m_2), (apk, \mathcal{S}_2, m_1)\}$. In AvASM the contribution of a tuple is $H_0(\mathsf{gid}, m)^{T_\mathcal{S}}$, and in AASMwSA and AASMwCA (Section 7.2.2) it is $\big(H_0(apk, m) \prod_{j \in \mathcal{S}} H_2(apk, j)\big)^{x_\mathcal{S}}$ with $spk = g_2^{x_\mathcal{S}}$; in both, the message and the subgroup are bound by a single exponent, and swapping them changes the aggregate unless the two exponents are equal. The proof of the next theorem makes this precise.

Next, we prove the security of AvASM by a direct reduction from the ψ-co-CDH problem, which extends the proof of Theorem 5.6.

Theorem 7.3. *Let the proofs π_i be as in Theorem 5.6. If the ψ-co-CDH problem is (τ', ϵ')-hard, then AvASM is $(\tau, q_G, q_H, q_S, N, \epsilon)$-secure (Definition 7.1) against algebraic forgers in the algebraic group model and the random oracle model, for all τ and ϵ satisfying*

$$\epsilon \;\geq\; \mathrm{e}(q_S + N)\,\epsilon' + \frac{(q_G + 1)\,q_H + n\,(q_G + 1)}{q},$$
$$\tau \;\leq\; \tau' - O\big(q_H + q_S + N\big)\tau_{exp_1} - O\big(n^2 q_G + nN\big)\tau_{exp_2} - O\big(n^3 q_G^2\big)\tau_{mul},$$

where n is the maximum number of members of a group, the reduction makes at most $N-1$ queries to $\mathcal{O}^\psi$, and τ_{exp_1}, τ_{exp_2} and τ_{mul} are as in Theorem 5.6.

Proof. Let $\mathcal{F}$ be a forger. We build $\mathcal{B}$ that, on a ψ-co-CDH instance $(A = g_1^\alpha, B_1 = g_1^\beta, B_2 = g_2^\beta)$ with access to $\mathcal{O}^\psi$, outputs $g_1^{\alpha\beta}$. $\mathcal{B}$ runs the reduction of Theorem 5.6 with $\Pr[\omega = 0] = 1/(q_S + N)$; keys, random oracles, group setups and signing queries are simulated as there.

Output. Let $\mathcal{F}$ output tuples $(gpk_k, \mathcal{S}_k, m_k)$, $k = 1, \ldots, N$, and Σ. $\mathcal{B}$ aborts if **AVf** rejects; otherwise let k^* be the first tuple that satisfies the winning condition. For every tuple, $\mathcal{B}$ computes $M_k = \prod_{j \in \mathcal{S}_k} mpk_j^{(k)}$ from COM_k, as the verifier does, and queries $H_0(\mathsf{gid}_k, m_k)$ if $\mathcal{F}$ has not. Let I^* be the set of tuples with $(\mathsf{gid}_k, m_k) = (\mathsf{gid}_{k^*}, m_{k^*})$; they all have the same $\mathcal{PK}$ and COM, hence the same Φ, and $\mathcal{B}$ obtains the polynomials of gpk_{k^*}

as in the output step of the proof of Theorem 5.6, or β. $\mathcal{B}$ aborts unless the bit ω of $(\mathsf{gid}_{k^*}, m_{k^*})$ is 0 and the bit of every other pair (gid_k, m_k) in the output is 1. For $k \notin I^*$ we then have $H_0(\mathsf{gid}_k, m_k) = g_1^{r_k}$ and $e(g_1^{r_k}, M_k) = e\big(\mathcal{O}^\psi(M_k)^{r_k}, g_2\big)$. These tuples may be on groups whose keys $\mathcal{B}$ does not know, and $\mathcal{B}$ removes them with one query to $\mathcal{O}^\psi$ each; this is the only place where the oracle is needed. For $k \in I^*$, $M_k = g_2^{T_{\mathcal{S}_k}}$, and as in the proof of Theorem 5.6, $T_{\mathcal{S}_k} = \delta_k\theta + K_k$, where $\delta_k = 1$ if $\iota \in \mathcal{S}_k$ and $\delta_k = 0$ otherwise, and K_k is known to $\mathcal{B}$. Let $H_0(\mathsf{gid}_{k^*}, m_{k^*}) = A\, g_1^r$, $\Delta = \sum_{k\in I^*}\delta_k$ and $K = \sum_{k\in I^*} K_k$. The verification equation (7) then reads $e(\Sigma, g_2) = \prod_{k\notin I^*} e\big(\mathcal{O}^\psi(M_k)^{r_k}, g_2\big) \cdot e\big((A\, g_1^r)^{\sum_{k\in I^*} T_{\mathcal{S}_k}}, g_2\big)$, and since $e(\cdot, g_2)$ is injective on $\mathbb{G}_1$, with $\theta = u_\theta\beta + v_\theta$,

$$\sigma^* = \Sigma \cdot \prod_{k\notin I^*} \mathcal{O}^\psi(M_k)^{-r_k} = (A\, g_1^r)^{\Delta u_\theta\beta + \Delta v_\theta + K},$$

hence

$$g_1^{\alpha\beta} = \Big(\sigma^* \cdot A^{-(\Delta v_\theta + K)} \cdot B_1^{-r\Delta u_\theta} \cdot g_1^{-r(\Delta v_\theta + K)}\Big)^{(\Delta u_\theta)^{-1}}.$$

Here $u_\theta \neq 0$, and $1 \le \Delta \le N < q$ because $k^* \in I^*$, so Δu_θ is invertible.

Analysis. Signing queries are on pairs other than $(\mathsf{gid}_{k^*}, m_{k^*})$, and the output contains at most $N-1$ other pairs. $\mathcal{B}$ therefore needs $\omega = 0$ for one pair and $\omega = 1$ for at most $q_S + N - 1$ others, which happens with probability at least $\frac{1}{q_S+N}\big(1 - \frac{1}{q_S+N}\big)^{q_S+N-1} \ge \frac{1}{\mathrm{e}(q_S+N)}$, independently of $\mathcal{F}$'s view. The other losses are those of Theorem 5.6, so $\epsilon' \ge \big(\epsilon - ((q_G+1)q_H + nq_G)/q\big)/\big(\mathrm{e}(q_S+N)\big)$, which gives the bound on ϵ. In addition, $\mathcal{B}$ spends $O(n)$ exponentiations in $\mathbb{G}_2$ per output tuple to compute M_k from the n commitments in COM_k, $O(1)$ exponentiations in $\mathbb{G}_1$ per output tuple, and at most $N-1$ queries to $\mathcal{O}^\psi$. Merging the tuples with the same (gid, m) is what makes randomization exponents unnecessary (Remark 7.2): repeating the target tuple only increases Δ. □

7.2.2. Aggregated versions of ASMwSA/ASMwCA Schemes

The ASMwSA and ASMwCA schemes described in Section 6 can be extended to aggregated versions AASMwSA and AASMwCA, with the algorithms SAg and AVf defined above and without the exponents b_i of the AASM scheme. Here the public data of a group is its key list, $gpk_k = \mathcal{PK}_k$, and for a subgroup $\mathcal{S}_k$ we write $spk_k = \prod_{j\in\mathcal{S}_k} pk_j^{a_j} = g_2^{x_{\mathcal{S}_k}}$ with $x_{\mathcal{S}_k} = \sum_{j\in\mathcal{S}_k} a_j sk_j$, where $a_j = H_1(pk_j, \mathcal{PK}_k)$ and pk_j, sk_j are the keys of member j of $\mathcal{PK}_k$.

In addition to the phases of the described schemes in Section 6, the two algorithms are as follows.

- $\mathbf{SAg}(par, \{(\mathcal{PK}_k, \mathcal{S}_k, m_k, \sigma_k)\}_{k=1}^N) \to \Sigma$: It is run by any party, takes as input the public parameters par and N tuples, each consisting of a key list $\mathcal{PK}_k$, a subgroup $\mathcal{S}_k$, a message m_k and an ASMwSA (or ASMwCA) signature σ_k on m_k by $\mathcal{S}_k$, and outputs the aggregate signature $\Sigma = \prod_{k=1}^N \sigma_k$.
- $\mathbf{AVf}(par, \{(\mathcal{PK}_k, \mathcal{S}_k, m_k)\}_{k=1}^N, \Sigma) \to b \in \{0,1\}$: It is run by a verifier and takes as input the public parameters par, N tuples $(\mathcal{PK}_k, \mathcal{S}_k, m_k)$ and an aggregate signature Σ. Once per group, as in ASMwSA, it checks that the keys in $\mathcal{PK}_k$ are distinct and that every proof π_j verifies, and computes apk_k and the values $pk_j^{a_j}$; for each tuple it computes spk_k. It outputs $b = 0$ if one of these checks fails or two tuples have the same apk_k but different $\mathcal{PK}_k$, and otherwise outputs $b = 1$ if and only if

$$\prod_{k=1}^{N} e\Bigg(H_0(apk_k, m_k) \cdot \prod_{j \in \mathcal{S}_k} H_2(apk_k, j), spk_k\Bigg) = e(\Sigma, g_2). \tag{8}$$

Remark 7.2 applies here as well, with (apk_k, m_k) in place of (gid_k, m_k).

Note that the correctness property follows from the below equation array.

$$\begin{aligned}
&\prod_{k=1}^{N} e\Bigg(H_0(apk_k, m_k) \cdot \prod_{j \in \mathcal{S}_k} H_2(apk_k, j), spk_k\Bigg) \\
&\quad= \prod_{k=1}^{N} e\Bigg(H_0(apk_k, m_k)^{x_{\mathcal{S}_k}} \cdot \Bigg(\prod_{j \in \mathcal{S}_k} H_2(apk_k, j)\Bigg)^{x_{\mathcal{S}_k}}, g_2\Bigg) \\
&\quad= \prod_{k=1}^{N} e\big(\sigma_k, g_2\big) \\
&\quad= e\Bigg(\prod_{k=1}^{N} \sigma_k, g_2\Bigg) \\
&\quad= e\big(\Sigma, g_2\big).
\end{aligned}$$

As for AvASM, we prove the security of AASMwSA and AASMwCA by a direct reduction from the ψ-co-CDH problem, which extends the proofs of Theorems 6.4 and 6.5.

Theorem 7.4. *Let the proofs π_i be as in Theorem 6.4. If the ψ-co-CDH problem is (τ', ϵ')-hard, then AASMwSA and AASMwCA, the latter with the combiner controlled by the adversary, are $(\tau, q_G, q_H, q_S, N, \epsilon)$-secure (Definition 7.1) against algebraic forgers, in the algebraic group model and the random oracle model, for all τ and ϵ satisfying*

$$\begin{aligned} \epsilon \;&\geq\; \mathrm{e}(q_S + N)\,\epsilon' + \frac{2q_H + n\,(q_G + 1)}{q}, \\ \tau \;&\leq\; \tau' - O\big(q_H + q_S + nq_G + N\big)\tau_{exp_1} - O\big(nq_G\big)\tau_{exp_2} \\ &\qquad -2N\tau_{exp_2^n} - O\big(nq_G\big)\tau_{mul}, \end{aligned}$$

where n is the maximum number of members of a group, the reduction makes at most $N - 1$ queries to $\mathcal{O}^\psi$, $\tau_{exp_2^n}$ is the time required to compute an n-multi-exponentiation in $\mathbb{G}_2$, and τ_{exp_1}, τ_{exp_2} and τ_{mul} are as in Theorem 5.6.

Proof. Let $\mathcal{F}$ be a forger. We build $\mathcal{B}$ that, on a ψ-co-CDH instance $(A = g_1^\alpha, B_1 = g_1^\beta, B_2 = g_2^\beta)$ with access to $\mathcal{O}^\psi$, outputs $g_1^{\alpha\beta}$. $\mathcal{B}$ is the reduction of Theorem 6.4, with the addition of Theorem 6.5 for AASMwCA, and with $\Pr[\omega = 0] = 1/(q_S + N)$. In particular, $\mathcal{B}$ answers $H_2(apk, j) = g_1^{r_{apk,j}}$ with known $r_{apk,j}$ for every input, and knows the secret keys of the corrupt members of every group that is set up.

Let $\mathcal{F}$ output tuples $(\mathcal{PK}_k, \mathcal{S}_k, m_k)$, $k = 1, \ldots, N$, and Σ, and let k^* be the first tuple that satisfies the winning condition, on a key list $\mathcal{PK}^*$ with aggregated key apk^*; as in the proof of Theorem 6.4, $\mathcal{B}$ knows or now extracts every corrupt secret key of $\mathcal{PK}^*$, or obtains β. For every tuple, $\mathcal{B}$ computes apk_k, spk_k and $\bar{r}_k = \sum_{j \in \mathcal{S}_k} r_{apk_k,j}$, so that $\prod_{j \in \mathcal{S}_k} H_2(apk_k, j) = g_1^{\bar{r}_k}$, and queries $H_0(apk_k, m_k)$ if $\mathcal{F}$ has not. Let I^* be the set of tuples with $(apk_k, m_k) = (apk^*, m_{k^*})$. Since the verifier rejects tuples with the same apk and different key lists, every $k \in I^*$ has $\mathcal{PK}_k = \mathcal{PK}^*$. $\mathcal{B}$ aborts unless the bit ω of (apk^*, m_{k^*}) is 0 and the bit of every other pair (apk_k, m_k) in the output is 1. For $k \notin I^*$ we then have $H_0(apk_k, m_k) = g_1^{r_k}$ and $e\big(g_1^{r_k + \bar{r}_k}, spk_k\big) = e\big(\mathcal{O}^\psi(spk_k)^{r_k + \bar{r}_k}, g_2\big)$, so $\mathcal{B}$ removes these tuples, whose keys it may not know, with one query to $\mathcal{O}^\psi$ each. For $k \in I^*$, $H_0(apk^*, m_{k^*}) = A\, g_1^r$ and $spk_k = g_2^{\delta_k a_\iota \beta + K_k}$, where $\delta_k = 1$ if $\iota \in \mathcal{S}_k$ and $\delta_k = 0$ otherwise, and $K_k = \sum_{j \in \mathcal{S}_k \setminus \{\iota\}} a_j sk_j$ is known to $\mathcal{B}$. Hence

$$\sigma^* = \Sigma \cdot \prod_{k \notin I^*} \mathcal{O}^\psi(spk_k)^{-(r_k + \bar{r}_k)} \cdot \prod_{k \in I^*} A^{-K_k}\, B_1^{-\delta_k a_\iota (r + \bar{r}_k)}\, g_1^{-K_k (r + \bar{r}_k)} = g_1^{\Delta a_\iota \alpha\beta},$$

where $\Delta = \sum_{k \in I^*} \delta_k$ satisfies $1 \le \Delta \le N < q$ because $k^* \in I^*$. $\mathcal{B}$ outputs $(\sigma^*)^{(\Delta a_\iota)^{-1}} = g_1^{\alpha\beta}$, which fails only if $a_\iota = 0$; some answer of H_1 is 0 with probability at most q_H/q. As in the proof of Theorem 6.4, extraction fails with probability at most $(q_H + n(q_G + 1))/q$, and the no-abort probability is at least $1/(\mathrm{e}(q_S + N))$ as in the proof of Theorem 7.3. Hence $\epsilon' \ge \big(\epsilon - (2q_H + n(q_G + 1))/q\big)/\big(\mathrm{e}(q_S + N)\big)$, which gives the bound on ϵ; the overhead adds to that of Theorem 6.4 at most N hash queries, $2N$ multi-exponentiations in $\mathbb{G}_2$ for apk_k and spk_k, and $O(N)$ exponentiations in $\mathbb{G}_1$. The proof uses no forking, so, unlike the bound in the first version of this report (2022) [26], the bound loses only the factor $\mathrm{e}(q_S + N)$. □

8. Comparison

In Table 1, we compare ASM, ASM-PoP, and the proposed schemes in this paper. Our comparison contains the number of operations required in each phase of those schemes. For example, consider a comparison of group setup phases of the schemes ASM and vASM. It can be seen in Table 1 that the ASM scheme costs of n H_2 hashes (onto $\mathbb{G}_1$), n exponentiations and $(n-1)$ multiplications in $\mathbb{G}_1$ in the group setup phase. On the other hand, in the group setup of vASM each member computes $2n-1$ exponentiations and $n(n-1)$ multiplications in $\mathbb{G}_2$ for its commitments, its mpk_i, the combined set COM and the share checks (b), $n-1$ Horner evaluations for these checks (note (e) of Table 1), and $n(n-1)$ multiplications in $\mathbb{Z}_q$ for the shares; the proofs of knowledge add n exponentiations in $\mathbb{G}_2$ to create π_i and $n(n-1)$ two-term multi-exponentiations in $\mathbb{G}_2$ to verify the $n-1$ received proofs.

It can also be seen from Table 1 that the vASM scheme requires fewer operations than ASM and ASM-PoP in the verification phase. The group setup of vASM is run once per group and costs $O(n^2)$ operations per member; a verifier also checks the n dealings once per group, at $O(n^2)$ cost (note (h)). Each verification saves one pairing over ASM, and also the l hashes to $\mathbb{G}_1$ if the ASM verifier computes $H_2(apk, j)$ for every signature. Against such a verifier, the number of verifications after which vASM's total cost drops below that of ASM is about 110 for $n = 16$, 470 for $n = 64$ and 2 000 for $n = 256$ (Section 9). An ASM verifier can instead store the n values $H_2(apk, j)$ of a group, just as the vASM verifier stores the n values mpk_i (note (g) of Table 1). With equal storage on both sides, the hashes disappear but ASM still computes three pairings against two for vASM, so vASM is faster by 0.20 to 0.23 ms for $n \le 64$ and equally fast at $n = 256$, and faster by 0.25 to

0.40 ms for every n when the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ are exchanged (Section 9); in addition, the vASM signature is shorter and fully aggregatable. We checked the columns for ASM and ASM with PoP against [16, Section 4.2] and [48, Section 6.2]; two cells of the PoP column are corrected, see note (f). On the other hand, since the set of membership public keys MPK and the set of commitments COM are published, the broadcasted data size in vASM is linear in the number of users in $\mathcal{G}$.

The ASMwSA scheme provides efficient signature aggregation and verification phases; however, its signature generation requires more multiplications in $\mathbb{G}_1$ and extra storage for $(n-1)$ $\mathbb{G}_1$ elements, and its group setup also includes checking the $n-1$ key proofs (Section 9). Its verifier also computes spk, with $l-1$ multiplications in $\mathbb{G}_2$ for each signature (Remark 6.1).

The ASMwCA scheme has an efficient verification phase and the same group setup cost as ASMwSA, but it requires more computations in the signature aggregation phase. On the other hand, it requires extra storage for $n(n-1)$ $\mathbb{G}_1$ elements at the combiner.

Comparison with the multi-signature of Boldyreva. The multi-signature scheme MGS of Boldyreva [7], used with the list of signers, also verifies with two pairings: the verifier multiplies the public keys of the l signers and checks one pairing equation, as the vASM verifier does with the membership public keys, so the cost per signature is the same. MGS has no group setup, and its verifier also needs the n public keys of the group. Its security is proven in a model in which the adversary must know the secret keys of the corrupt signers, which in practice is realised with proofs of possession [29]. What vASM adds at the price of its setup is the group itself. In MGS, a subgroup of registered signers can sign for any set of keys that contains it, and nothing shows that the other members agreed to form the group. In vASM, the signature is bound to gid through $H_0(\mathsf{gid}, m)$, and the verifier accepts a group only if the dealing of every member, with its proof of knowledge, is in gpk (Section 5.1). This corresponds to the setting of Micali et al. [17], in which all members of $\mathcal{G}$ take part in generating the keys of the group. When this property is not needed, MGS gives the same verification cost without a group setup.

Consider N distinct accountable subgroup multi-signatures. The ASM scheme proposed by Boneh et al. supports a partial aggregation. Therefore the AASM scheme outputs an aggregated signature consisting of N many $\mathbb{G}_2$ elements and one $\mathbb{G}_1$ element. On the other hand, the AvASM outputs

an aggregated signature with only one $\mathbb{G}_1$ element. In terms of verification efficiency, we only compare the number of pairings required in verification equations (since the pairing operations dominate the cost of verification). All the proposed schemes, i.e. AvASM, AASMwSA, and AASMwCA, require $N + 1$ pairings for verification whereas the AASM requires $2N + 1$.

9. Implementation and evaluation

We implemented vASM (with and without the proofs of knowledge), the ASM scheme of [16], ASMwSA, ASMwCA and their aggregated versions in Python on top of `py_arkworks_bls12381` [49], a binding to the Rust arkworks library for BLS12-381 that provides hashing to $\mathbb{G}_1$ and $\mathbb{G}_2$ (RFC 9380 [50]), multi-exponentiation and multi-pairing. The implementation also runs the attacks of Section 5.3 and Remark 5.5 end to end: without the corresponding checks the forged signatures verify, and with them the honest member aborts. As suggested in Section 5.1, it uses the SHA-256 digest of $(\mathcal{PK}, \mathrm{COM})$ as gid, computed once per group, so that H_0 hashes 32 bytes and the message. A verifier obtains gid and MPK only after checking the published dealings, as in the verification step of Section 5.1. The code and the benchmark script are available at `https://github.com/ahmetramazan/vasm-pairing`.

All timings were taken on an Apple M2 Pro (macOS, Python 3.12, library version 0.5.0) in a single run of the benchmark script; the files with the results in the repository also give the minimum and maximum of every entry. Each entry is a median after one untimed warm-up run: over 21 runs for signing, verification and single operations, and over 5 runs for the group setups and the aggregate verification. The only exceptions are the setups for $n = 256$, where the ASM, ASMwSA and ASMwCA entries are medians of 3 runs and the vASM entry comes from a single run. For signing and verification, the membership keys of vASM and the values mpk_i were computed directly from the members' polynomials and from COM, as a verifier computes them; signing and verification use only the output of the setup, which is the same either way.

Table 2 lists the per-member cost of the group setup, and the cost of signing and verification for a subgroup of size $l = n/2$. The setup cost of a member is the work it performs itself. In ASM [16], this is the computation of the member's contributions μ_{ij} to all membership keys and of its own membership key from the n values it receives. In vASM, it is creating its own dealing with its proof of knowledge and checking the dealings of the other

Table 1: Comparison of the schemes$^{(i)}$

Phases	ASM [16]	ASM with PoP [16]	vASM	ASMwSA	ASMwCA
Key Generation	1 $\text{Exp}_{\mathbb{G}_2}$	pk: 1 $\text{Exp}_{\mathbb{G}_2}$ Proof: 1 H_3, 1 $\text{Exp}_{\mathbb{G}_1}{}^{(f)}$	1 $\text{Exp}_{\mathbb{G}_2}$	pk: 1 $\text{Exp}_{\mathbb{G}_2}$ $\pi_i^{(d)}$: 1 H_{pok}, 1 $\text{Exp}_{\mathbb{G}_2}$	pk: 1 $\text{Exp}_{\mathbb{G}_2}$ $\pi_i^{(d)}$: 1 H_{pok}, 1 $\text{Exp}_{\mathbb{G}_2}$
Key Aggregation	n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$	Proofs: n H_3, $2n$ Pairings apk: 1 H_4, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$	–	n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$	n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$
Group Setup	n H_2, n $\text{Exp}_{\mathbb{G}_1}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_1}$	n H_2, n $\text{Exp}_{\mathbb{G}_1}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_1}$	f_i, COM_i, mpk_i: n $\text{Exp}_{\mathbb{G}_2}$, $n(n-1)$ $\text{Mul}_{\mathbb{Z}_q}$ COM: $n(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ $\pi_i^{(d)}$: 1 H_{pok}, n $\text{Exp}_{\mathbb{G}_2}$ Checks (b)$^{(e)}$: $(n-1)$ $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ Horner eval. Checks (c)$^{(d)}$: $(n-1)$ H_{pok}, $n(n-1)$ $\text{MExp}^2_{\mathbb{G}_2}$	n H_2, n $\text{Exp}_{\mathbb{G}_1}$ Key proofs$^{(d)}$: $(n-1)$ H_{pok}, $(n-1)$ $\text{MExp}^2_{\mathbb{G}_2}$	n H_2, n $\text{Exp}_{\mathbb{G}_1}$ Key proofs$^{(d)}$: $(n-1)$ H_{pok}, $(n-1)$ $\text{MExp}^2_{\mathbb{G}_2}$
Signature Generation	1 H_0, 1 $\text{Exp}_{\mathbb{G}_1}$, 1 $\text{Mul}_{\mathbb{G}_1}$	1 H_0, 1 $\text{Exp}_{\mathbb{G}_1}$, 1 $\text{Mul}_{\mathbb{G}_1}$	1 H_0, 1 $\text{Exp}_{\mathbb{G}_1}$	1 H_0, 1 $\text{Exp}_{\mathbb{G}_1}$, l $\text{Mul}_{\mathbb{G}_1}$	1 H_0, 1 $\text{Exp}_{\mathbb{G}_1}$, 1 $\text{Mul}_{\mathbb{G}_1}$
Signature Aggregation	$(l-1)$ $\text{Mul}_{\mathbb{G}_1}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$	$(l-1)$ $\text{Mul}_{\mathbb{G}_1}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$	$(l-1)$ $\text{Mul}_{\mathbb{G}_1}$	$(l-1)$ $\text{Mul}_{\mathbb{G}_1}$	(l^2-1) $\text{Mul}_{\mathbb{G}_1}$
Verification	$apk^{(c)}$: n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ Per signature: 1 H_0, l $H_2{}^{(g)}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_1}$, 1 $\text{Mul}_{\mathbb{G}_T}$, 3 Pairings	Proofs: n H_3, $2n$ Pairings$^{(f)}$ $apk^{(c)}$: 1 H_4, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ Per signature: 1 H_0, l $H_2{}^{(g)}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_1}$, 1 $\text{Mul}_{\mathbb{G}_T}$, 3 Pairings	MPK$^{(h)}$: n Horner eval.$^{(e)}$ Per signature: 1 H_0, $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$, 2 Pairings	Key proofs$^{(d)}$: n H_{pok}, n $\text{MExp}^2_{\mathbb{G}_2}$ $apk^{(a)}$: n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ Per signature: 1 H_0, l $H_2{}^{(g)}$, l $\text{Mul}_{\mathbb{G}_1}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$, 2 Pairings	Key proofs$^{(d)}$: n H_{pok}, n $\text{MExp}^2_{\mathbb{G}_2}$ $apk^{(a)}$: n H_1, n $\text{Exp}_{\mathbb{G}_2}$, $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ Per signature: 1 H_0, l $H_2{}^{(g)}$, l $\text{Mul}_{\mathbb{G}_1}$, $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$, 2 Pairings
Transmission	Group setup: $(n-1)$ $\mathbb{G}_1$ Elt. Signature: 1 $\mathbb{G}_1$ Elt., 1 $\mathbb{G}_2$ Elt.	Group setup: $(n-1)$ $\mathbb{G}_1$ Elt. Signature: 1 $\mathbb{G}_1$ Elt., 1 $\mathbb{G}_2$ Elt.	Group setup: $(n-1)$ $\mathbb{Z}_q$ Elt. Signature: 1 $\mathbb{G}_1$ Elt.	Signature: 1 $\mathbb{G}_1$ Elt.	Group setup: $(n-1)$ $\mathbb{G}_1$ Elt. Signature: 1 $\mathbb{G}_1$ Elt.
Broadcasting	pk: 1 $\mathbb{G}_2$ Elt.	pk: 1 $\mathbb{G}_2$ Elt. Proof: 1 $\mathbb{G}_1$ Elt.	pk, mpk_i: 2 $\mathbb{G}_2$ Elt. COM_i: $(n-1)$ $\mathbb{G}_2$ Elt.$^{(j)}$ $\text{COM}^{(b)}$: n $\mathbb{G}_2$ Elt. $\pi_i^{(d)}$: $(n+1)$ $\mathbb{Z}_q$ Elt.	pk: 1 $\mathbb{G}_2$ Elt. $\pi_i^{(d)}$: 2 $\mathbb{Z}_q$ Elt.	pk: 1 $\mathbb{G}_2$ Elt. $\pi_i^{(d)}$: 2 $\mathbb{Z}_q$ Elt.
Storage	sk: 1 integer ($\mathcal{O}(\lambda)$-bit) mk: 1 $\mathbb{G}_1$ Elt.	sk: 1 integer ($\mathcal{O}(\lambda)$-bit) mk: 1 $\mathbb{G}_1$ Elt.	sk: 1 integer ($\mathcal{O}(\lambda)$-bit) mk: 1 integer ($\mathcal{O}(\lambda)$-bit)	sk: 1 integer ($\mathcal{O}(\lambda)$-bit) mk: n $\mathbb{G}_1$ Elt.	sk: 1 integer ($\mathcal{O}(\lambda)$-bit) mk: 1 $\mathbb{G}_1$ Elt. Combiner: (n^2-n) $\mathbb{G}_1$ Elt.

Notes to Table 1

(a) Once per group the verifier checks the key proofs and computes apk and the $pk_j^{a_j}$; for each signature it computes $spk = \prod_{j \in \mathcal{S}} pk_j^{a_j}$ itself (Remark 6.1).

(b) Since all the users compute the same commitment set COM, it is enough to be broadcast by only one user. A verifier, however, needs the dealings (COM_j, π_j) of all members (note (h)).

(c) apk is computed by the verifier once per group, never taken from the combiner.

(d) Non-interactive Schnorr proofs of Section 3.3: for $d+1$ exponents, creating a proof costs $d+1$ $\mathrm{Exp}_{\mathbb{G}_2}$ and one hash, verifying it $d+1$ $\mathrm{MExp}^2_{\mathbb{G}_2}$ (two-term multi-exponentiations in $\mathbb{G}_2$) and one hash, and it consists of $d+2$ elements of $\mathbb{Z}_q$; $d = n-1$ in vASM and $d = 0$ for the key proofs of ASMwSA and ASMwCA. In ASMwSA and ASMwCA the proofs are attached to the keys, so each is verified once per key, not once per group.

(e) Horner eval.: a product $\prod_{k=0}^{n-1} C_k^{i^k}$ computed with Horner's rule, as $n-1$ exponentiations by the small integer i and $n-1$ $\mathrm{Mul}_{\mathbb{G}_2}$, which is much cheaper than a general multi-exponentiation. Check (b) of the group setup evaluates it for every other dealer; these checks replace the single aggregate check of the first version and let a member blame a cheating dealer.

(f) In [48, Section 6.1] the PoP is $H_3(pk)^{sk} \in \mathbb{G}_1$: one $\mathrm{Exp}_{\mathbb{G}_1}$ to create it, two pairings to check it. The first version listed $\mathrm{Exp}_{\mathbb{G}_2}$ in both cells.

(g) A verifier can store the n values $H_2(apk, j)$ of a group, as the vASM verifier stores the mpk_i; the l hashes per verification then disappear.

(h) Once per group, the verifier checks the n dealings (COM_j, π_j), that is, n proofs of Section 3.3 with $d = n-1$, about as costly as the checks of one member in the group setup; it then computes COM and the n values $mpk_i = \prod_{k=0}^{n-1} C_k^{i^k}$, or checks given values against COM at the same cost, and stores them.

(i) Per member: key generation, key aggregation, group setup, signature generation, transmission, broadcasting and storage; for the combiner: signature aggregation; for the verifier: verification. Operations in $\mathbb{Z}_q$ are omitted except for the vASM shares.

(j) $C_0^{(i)} = pk_i$ is already public.

$\mathrm{Exp}_{\mathbb{G}_i}$, $\mathrm{Mul}_{\mathbb{G}_i}$ Exponentiation in $\mathbb{G}_i$, $i \in \{1, 2\}$; multiplication in $\mathbb{G}_i$, $i \in \{1, 2, T\}$; $\mathrm{Mul}_{\mathbb{Z}_q}$: multiplication in $\mathbb{Z}_q$.
$H_0, \ldots, H_4, H_{\mathsf{pok}}$ One evaluation of the hash function of the schemes. $\mathbb{G}_i$ Elt.: element of $\mathbb{G}_i$. λ: security parameter.

$n-1$ members. The share checks of member i consist of exponentiations by i (note (e) of Table 1), so their cost depends on i; we report member $n-1$, the most expensive one for the values of n we use. In ASMwSA and ASMwCA, it is the check of the key proofs of the other $n-1$ members followed by the computation of the μ_{ij}; the two schemes differ only in where the received values are stored, so their costs are the same. For the verification of ASM, ASMwSA and ASMwCA we measure two verifiers: one computes the l values $H_2(apk, j)$ for every signature ("H_2 hashed"), the other computes all n of them once per group and stores them ("H_2 stored"), just as the vASM verifier stores the n values mpk_i. The verifier of ASMwSA and ASMwCA always computes spk from the stored values $pk_j^{a_j}$ (Remark 6.1). All schemes are measured with the same library, the same hash functions and the same storage options, so that no scheme benefits from an optimization that is withheld from another.

Group setup. In ASM [16] the setup costs a member 2.5 ms at $n = 4$ and 0.61 s at $n = 256$. All three proposed schemes have a more expensive setup, but to very different degrees. In vASM the cost per member grows quadratically in n, since each member checks $n-1$ dealings of n commitments each, together with their proofs of knowledge; it grows by a factor of about 16 whenever n is multiplied by 4, from 0.2 s at $n = 16$ to 3.1 s at $n = 64$ and 52 s at $n = 256$. In ASMwSA and ASMwCA the setup costs 1.3 to 2.1 times as much as in ASM, because each member also checks the key proofs of the other members: 105 ms per member at $n = 64$ and 0.79 s at $n = 256$. In all schemes the setup is run once per group.

Signing. Signing a share costs 0.30 to 0.32 ms in ASM and vASM and 0.30 to 0.39 ms in ASMwSA and ASMwCA; in all schemes it is dominated by one hash to $\mathbb{G}_1$ and one exponentiation in $\mathbb{G}_1$. Only in ASMwSA does it grow with the subgroup size, because the signer multiplies the l values μ_{ij}, $j \in \mathcal{S}$; at $n = 256$ these 128 multiplications in $\mathbb{G}_1$ add about 0.08 ms, the cost of a product of 128 elements in Table 3.

Verification. ASM [16] verifies with a multi-pairing with three pairs. When its verifier hashes the l values $H_2(apk, j)$ for every signature, verification grows linearly with l, to 7.8 ms at $n = 64$ and 27.6 ms at $n = 256$. When the verifier stores these values for each group, the hashes disappear and verification takes 1.50 to 1.62 ms; the verifier then keeps n group elements per group and multiplies l of them in $\mathbb{G}_1$, and it receives pk as part of

Table 2: Measured running times in milliseconds on an Apple M2 Pro. Signing and verification use a subgroup of size $l = n/2$; "H_2 hashed" and "H_2 stored" are the two verifiers described in the text. Setup and signing are measured with the keys in $\mathbb{G}_2$, as in the description of the schemes; verification is measured in both orientations: with public keys, mpk_i and spk in $\mathbb{G}_1$ and H_0, H_2 and signatures in $\mathbb{G}_2$, and the other way round. In each part of the table, the lowest value in each column is in bold.

n	4	16	64	256
Setup, per member				
ASM [16]	**2.5**	**9.6**	**58.2**	**610**
vASM	13.4	203	3 126	52 382
ASMwSA / ASMwCA	4.7	20.2	105	794
Signing (one share)				
ASM [16]	**0.30**	**0.30**	0.32	**0.31**
vASM	0.31	0.31	**0.30**	**0.31**
ASMwSA	0.31	0.31	0.33	0.39
ASMwCA	**0.30**	0.32	0.31	**0.31**
Verification, keys in $\mathbb{G}_1$				
ASM [16], H_2 hashed	2.54	4.94	14.4	54.3
ASM [16], H_2 stored	1.71	1.71	1.75	2.01
vASM	**1.45**	**1.46**	**1.47**	**1.61**
ASMwSA / ASMwCA, H_2 hashed	2.29	4.72	14.2	54.4
ASMwSA / ASMwCA, H_2 stored	1.46	1.48	1.54	1.84
Verification, keys in $\mathbb{G}_2$				
ASM [16], H_2 hashed	1.92	3.11	7.81	27.6
ASM [16], H_2 stored	1.50	1.50	1.51	1.62
vASM	**1.27**	**1.29**	**1.31**	**1.59**
ASMwSA / ASMwCA, H_2 hashed	1.69	2.90	7.61	27.5
ASMwSA / ASMwCA, H_2 stored	**1.27**	**1.29**	1.34	1.64

the signature, which is therefore one $\mathbb{G}_2$ element longer. vASM verification consists of one hash to $\mathbb{G}_1$, a product of l stored elements of $\mathbb{G}_2$ and a multi-pairing with two pairs, with the same storage of n group elements per group. Its time is therefore almost independent of the group size: 1.27 to 1.31 ms for $n \leq 64$ and 1.59 ms at $n = 256$, where the product of 128 elements of $\mathbb{G}_2$ accounts for the increase. vASM is 6.0 times faster than ASM with hashed H_2 at $n = 64$ and 17 times faster at $n = 256$. Against ASM with stored H_2, vASM saves the third pairing, 0.22 ms (Table 3), but forms its product of l elements in $\mathbb{G}_2$ instead of $\mathbb{G}_1$, which costs about 0.15 ms more for $l = 128$. Accordingly, vASM is faster by 0.20 to 0.23 ms for $n \leq 64$, while at $n = 256$

Table 3: Cost of single operations in milliseconds (median of 21 runs).

Operation	Time
Product of 128 elements of $\mathbb{G}_1$	0.079
Product of 128 elements of $\mathbb{G}_2$	0.231
Hash to $\mathbb{G}_1$	0.178
Hash to $\mathbb{G}_2$	0.369
Multi-pairing with 2 pairs	1.133
Multi-pairing with 3 pairs	1.352

the difference of 0.03 ms is within the variation between runs. ASMwSA and ASMwCA also verify with two pairings, but compute *spk* from l elements of $\mathbb{G}_2$ in addition to the product of the l values $H_2(apk, j)$ in $\mathbb{G}_1$. With hashed H_2 they take 7.6 ms at $n = 64$ and 27.5 ms at $n = 256$; with stored H_2 they are as fast as vASM for $n \leq 64$ and save 0.17 to 0.24 ms over ASM there, while at $n = 256$ they are 0.01 ms slower than ASM.

Exchanging $\mathbb{G}_1$ and $\mathbb{G}_2$. The comparison does not depend on which group holds the keys. The upper part of the verification rows of Table 2 repeats the measurements with the roles of $\mathbb{G}_1$ and $\mathbb{G}_2$ exchanged, for all schemes alike. Since hashing to $\mathbb{G}_2$ costs about twice as much as hashing to $\mathbb{G}_1$, the verifiers that hash H_2 for every signature become about twice as slow, with 14.4 ms at $n = 64$ and 54.3 ms at $n = 256$. With stored H_2, the products of keys move to the cheaper group $\mathbb{G}_1$ and the products of stored hashes to $\mathbb{G}_2$. ASM then verifies in 1.71 to 2.01 ms and vASM in 1.45 to 1.61 ms, so vASM is faster by 0.25 to 0.40 ms for every n, and the difference grows with n. ASMwSA and ASMwCA take 1.46 to 1.84 ms. Comparing each scheme in its faster orientation, vASM, ASMwSA and ASMwCA verify faster than ASM by about 0.2 ms for $n \leq 64$, and at $n = 256$ all of them take 1.59 to 1.64 ms, which is within the variation between runs.

Cost of the setup over many verifications. Against an ASM verifier that hashes H_2 for every signature, the extra setup work of one vASM member over ASM is recovered after about 110 verifications for $n = 16$, 470 for $n = 64$ and 2 000 for $n = 256$. Against a verifier that stores these hashes, the saving per verification is small: the extra setup work is recovered only after about 910 verifications for $n = 16$ and 15 000 for $n = 64$, and at $n = 256$ the measured saving is too small to recover it. Against such a verifier, vASM pays for its setup mainly with signatures that are one $\mathbb{G}_2$ element shorter and

fully aggregatable. In the same way, the extra setup work of ASMwSA and ASMwCA over ASM is recovered after about 50 verifications for $n = 16$ and 270 for $n = 64$ with stored H_2, but not at $n = 256$. These figures compare the work of one member with the saving of one verifier. If the setup work of all n members is counted, a vASM group needs about 1 700, 30 000 and 510 000 verifications for $n = 16$, 64 and 256 against a verifier that hashes H_2, and about 15 000 for $n = 16$ and 980 000 for $n = 64$ against one that stores them, before the total work of the system is lower than with ASM; this does not include the check of the dealings by the verifier, which costs $O(n^2)$ exponentiations in $\mathbb{G}_2$, as the checks (c) of one member. vASM therefore suits groups that are set up once and verified very often, or in which the setup is required anyway, as discussed below.

Aggregate verification. Verifying an AvASM signature that aggregates N signatures from N different groups of 4 members took 1.7 ms for $N = 1$, 5.6 ms for $N = 10$ and 49 ms for $N = 100$, that is, about 0.5 ms per signature for $N = 100$. Each additional signature costs one hash to $\mathbb{G}_1$ and one more pair in the multi-pairing, whose final exponentiation is shared by all pairs. We did not measure the aggregated version of ASM [16]; Section 8 compares the two by the number of pairings only.

Communication and verifier state. With compressed points (48 bytes in $\mathbb{G}_1$, 96 bytes in $\mathbb{G}_2$) and 32-byte scalars, the dealing (COM_i, π_i) of a member has $96n + 32(n + 1)$ bytes, and a member sends $32(n - 1)$ bytes of shares on private channels. For $n = 256$, each member broadcasts about 32 KiB, so the setup broadcasts about 8 MiB in total, and about 2 MiB of shares are sent on private channels. A verifier that checks the group itself receives all dealings, also about 8 MiB for $n = 256$, and then keeps the n values mpk_i, 24 KiB for $n = 256$. The verifier of ASM [16] keeps only apk, 96 bytes, or also the n values $H_2(apk, j)$, 12 KiB for $n = 256$, if it stores them.

On-chain verification. To estimate the cost of verification in a smart contract, we implemented the verifiers of all schemes in Solidity on top of the BLS12-381 precompiles of EIP-2537 [51], which are available on Ethereum since the Pectra upgrade, with hashing to $\mathbb{G}_1$ and $\mathbb{G}_2$ as in RFC 9380 [50], checked against its test vectors. The public data of a group, MPK for vASM and apk, the values $pk_j^{a_j}$ and optionally the values $H_2(apk, j)$ for ASM, ASMwSA and ASMwCA, is written to contract storage once per group. Every verification reads the stored values it needs, checks that $\mathcal{S}$

is a strictly increasing list of indices in $\{1, \ldots, n\}$ and calls the pairing precompile. Table 4 lists the gas used by one transaction that verifies a signature with $l = n/2$ on a local chain with the current rules of Ethereum (Hardhat 2.29 [52], Osaka). Since the precompiles for $\mathbb{G}_1$ and $\mathbb{G}_2$ have different prices and a stored element of $\mathbb{G}_2$ takes twice the space of one of $\mathbb{G}_1$, we measured every scheme in both orientations. With the keys in $\mathbb{G}_1$, vASM uses the least gas for every n, for instance 447 000 against 705 000 for ASM with stored H_2 at $n = 64$. With the keys in $\mathbb{G}_2$, vASM reads l stored elements of $\mathbb{G}_2$, while ASM with stored H_2 reads its stored hashes in $\mathbb{G}_1$; vASM is then the cheapest for $n \leq 16$ but more expensive than ASM for $n \geq 64$. Comparing each scheme in its cheaper orientation, vASM uses the least gas for every n: 184 000 at $n = 4$, against 205 000 for ASMwSA and ASMwCA and 225 000 for ASM with stored H_2. For larger n, reading the l stored group elements dominates the cost, and vASM and ASM with stored H_2 come close: 447 000 against 458 000 gas at $n = 64$, and 1.19 against 1.21 million at $n = 256$. ASMwSA and ASMwCA read both the l values $pk_j^{a_j}$ and the l values $H_2(apk, j)$ and therefore use more gas than both for $n \geq 16$. Hashing H_2 in the contract is expensive: ASM then needs 880 000 gas at $n = 64$ and 3.0 million at $n = 256$. The gas of a transaction includes its base cost of 21 000 and the cost of its calldata, which is 3 600 to 9 600 gas here: the calldata holds the signature, the message and $\mathcal{S}$ as a list of 2-byte indices, and in ASM the signature also carries pk, which makes its calldata one group element longer than that of vASM. The rest, from 159 000 gas for vASM at $n = 4$ upwards, is execution, dominated by the precompiles and the storage reads. All verification transactions stay well below the limit of 2^{24} gas per transaction of EIP-7825. Registering a group costs more. Writing a new 32-byte storage slot costs 22 100 gas, and an element of $\mathbb{G}_1$ in the uncompressed form that the precompiles take needs at least 3 slots and one of $\mathbb{G}_2$ at least 6, so storing the n values mpk_i costs at least 66 300 gas per member with the keys in $\mathbb{G}_1$ and 132 600 with the keys in $\mathbb{G}_2$. For $n = 256$ this is at least 17.0 and 33.9 million gas, above the limit of one transaction, so the registration must be split over several transactions; ASM with stored H_2 writes n group elements as well. The check of the dealings, with $O(n^2)$ exponentiations in $\mathbb{G}_2$, is not practical in a contract; there it is done off-chain by a registration authority, as described in Remark 3.7. We repeated the measurement for $n = 4$ and $n = 16$ on the Sepolia test network: every transaction used exactly 6 000 gas less than on the local chain, for every scheme and both orientations, so the comparison

Table 4: Gas used by one verification transaction on Ethereum (EIP-2537), in thousands, subgroup size $l = n/2$, in both orientations. In each part of the table, the lowest value in each column is in bold.

n	4	16	64	256
Keys in $\mathbb{G}_1$				
ASM [16], H_2 hashed	365	739	2 249	8 517
ASM [16], H_2 stored	270	357	705	2 094
vASM	**214**	**260**	**447**	**1 195**
ASMwSA / ASMwCA, H_2 hashed	340	761	2 457	9 471
ASMwSA / ASMwCA, H_2 stored	246	379	913	3 048
Keys in $\mathbb{G}_2$				
ASM [16], H_2 hashed	250	375	880	2 993
ASM [16], H_2 stored	225	272	**458**	**1 206**
vASM	**184**	**271**	618	2 008
ASMwSA / ASMwCA, H_2 hashed	231	442	1 294	4 795
ASMwSA / ASMwCA, H_2 stored	205	339	872	3 008

is unchanged. The contract is deployed at `https://sepolia.etherscan.io/address/0x1Fb5826EbF075CE2502BfA9a71cC92fB88989116`, where all transactions can be inspected; their hashes are also given with the code.

Where vASM fits. The measurements suggest the following picture. Where the verifier is given the list of signers and stores n keys per group anyway, a BLS multi-signature with proofs of possession, as specified in [30] and used in Ethereum, or MGS [7], verifies as fast as vASM without any group setup (Section 8). vASM is meant for a group with fixed membership whose existence is itself a statement, such as the board of a company or the committee of a bridge that approves transfers by subgroups: an outsider, or a coalition of the other members, cannot form such a group in the name of an honest member, because the honest member's dealing and proof are part of *gpk* and every signature is bound to gid. The price is an n-of-n setup that any member can abort, whose cost grows quadratically in n and is paid again whenever the membership changes, and a registration on a blockchain that needs several transactions for large n. For groups of up to a few tens of members whose membership rarely changes, these costs are a few seconds of computation per member and a one-time registration.

10. Conclusion

In this work, we propose a novel BLS-based ASM scheme (vASM) that is more efficient than the ones in [16] in terms of signature size, signature aggregation and the number of pairings in verification, and as efficient in signature generation. On the other hand, our vASM scheme requires a one-time group setup with more multiplications: on an Apple M2 Pro, vASM verification takes 1.3 to 1.6 ms for groups of up to 256 members, while the setup costs 3.1 s per member at $n = 64$. Our analysis, in the model where the adversary controls $n - 1$ members and their keys, shows that a Feldman setup alone is not enough: the members must also prove knowledge of all sharing coefficients, and with these proofs vASM reduces to co-CDH in the random oracle model and the algebraic group model. We propose two more accountable subgroup multi-signature schemes with subgroup authentication (ASMwSA) and combiner authentication (ASMwCA), which verify with one pairing fewer but compute the subgroup key in every verification, and which have a more expensive group setup and storage and transmission disadvantages in comparison with the ones in [16]. A multi-signature with an explicit list of signers, such as MGS [7], verifies as fast as vASM without a group setup (Section 8); what vASM adds is a group setup to which every member contributes and which every verifier can check, at the price of an n-of-n setup whose cost grows quadratically in n.

Moreover, we compare the aggregated versions of the proposed schemes with the aggregated version of Boneh et al.'s ASM [16]. We see that aggregated versions of our schemes, i.e. AvASM, AASMwSA, and AASMwCA, are more efficient regarding aggregated signature size and the number of pairings in verification. According to the requirements of the system involving an ASM scheme, our schemes could be good alternatives, especially when short signatures, full aggregation or fewer pairings in verification are desired. It would be a good future work to add accountability to known multi-signature schemes by the methods given in this paper. Further open questions are a proof without the algebraic group model, tight reductions, dynamic membership without a new setup for vASM, constant-size verifier data as in [21], and instantiations beyond pairings, in particular with lattices as in [24], which need a statistical version of Lemma 2 and an explicit binding between the membership keys and the setup.

CRediT authorship contribution statement

Ahmet Ramazan Ağırtaş: Conceptualization, Methodology, Formal analysis, Investigation, Software, Writing – original draft, and Writing – review & editing

Oğuz Yayla: Supervision, Validation, Formal analysis, Writing – review & editing

Declaration of competing interest

The authors are the inventors of two Turkish patents related to this work: TR 2021/022220, "Eşleme tabanlı hesap verilebilir altgrup çoklu-imza şemaları içeren yöntem" (published 21 May 2024), which covers the pairing-based schemes of this paper, and TR 2022/014511, "Kuantum-dirençli kafes-tabanlı hesap verilebilir altgrup çoklu-imza şeması" (published 22 April 2024). Both patents are owned by Middle East Technical University; the authors are named as inventors.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the authors used Claude (Anthropic) in order to edit the text. The authors also used the same tool in the research itself, to check the reasoning of the security analysis and to write part of the implementation and test code (Section 9). After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Data availability

The scripts that verify Propositions 5.1 and 5.3, and the implementation and benchmark code, are available at `https://github.com/ahmetramazan/vasm-pairing`.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix A. Eliminating the combiner in ASMwSA

Consider ASMwSA in the case that the subgroup $\mathcal{S} := \{k_i : i = 1, 2, \ldots, l\}$ is determined before the signature protocol starts. In order to eliminate the designated combiner, the signer k_i for $i = 1, 2, \ldots, l$ proceeds as follows:

- Computes the i-th partial aggregated signature $\sigma_i = \sigma_{i-1} \cdot \big(H_0(apk, m)^{a_{k_i} sk_{k_i}} \cdot smk_{k_i}\big)$, where σ_{i-1} is the $(i-1)$-th partial aggregated signature computed by the signer $k_{i-1} \in \mathcal{S}$ and $\sigma_0 = 1_{\mathbb{G}_1}$.
- Sends σ_i to the signer k_{i+1} for $i < l$.
- Finally the last signer k_l outputs the aggregated signature $\sigma := \sigma_l$. This process is shown in Figure A.1.

Since $\sigma = \prod_{i=1}^{l} s_{k_i}$ with the individual signatures $s_{k_i} = H_0(apk, m)^{a_{k_i} sk_{k_i}} \cdot smk_{k_i}$ of ASMwSA, σ is an ASMwSA signature on m for $\mathcal{S}$, and the verifier checks it as in ASMwSA, computing spk itself (Remark 6.1):

$$e\Big(H_0(apk, m) \cdot \prod_{j \in \mathcal{S}} H_2(apk, j), spk\Big) \stackrel{?}{=} e(\sigma, g_2).$$

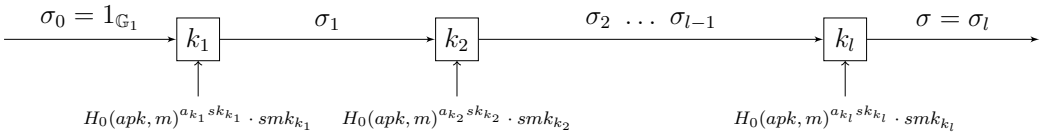

Figure A.1: ASMwSA without a combiner

In ASMwSA and ASMwCA, each signer sends their individual signature to the combiner. On the other hand, in this scenario, each signer sends the partial aggregated signature to another signer. Hence no party has to collect all individual signatures, and each signer still sends one element of $\mathbb{G}_1$. The signers, however, act one after another: signing takes l sequential steps instead of one round, and it stops if one signer does not respond. Since the combiner of ASMwSA only multiplies the individual signatures and need not be trusted, this changes the communication pattern but not the trust assumptions. The construction does not apply to ASMwCA, where

the components μ_{ij} are held by the combiner and the signers do not compute smk_i. This variant is covered by Theorem 6.4. The signatures are those of ASMwSA, and only the way they are multiplied changes: an honest signer k_i multiplies the value σ_{i-1} it receives by its individual signature s_{k_i}, and the adversary, who sees σ_{i-1} and σ_i, learns exactly $s_{k_i} = \sigma_i \sigma_{i-1}^{-1}$, which is what the signing oracle of ASMwSA returns.

References

[1] V. Buterin, Ethereum: A next-generation smart contract and decentralized application platform, `https://github.com/ethereum/wiki/wiki/White-Paper`, accessed 23 November 2026 (2013).

[2] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, accessed 23 November 2026 (2009).
URL `http://www.bitcoin.org/bitcoin.pdf`

[3] A. Narayanan, J. Bonneau, E. Felten, A. Miller, S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton University Press, USA, 2016.

[4] K. Itakura, K. Nakamura, A public-key cryptosystem suitable for digital multisignatures, NEC Research & Development 71 (1983) 1–8.

[5] K. Ohta, T. Okamoto, A digital multisignature scheme based on the Fiat-Shamir scheme, in: H. Imai, R. L. Rivest, T. Matsumoto (Eds.), Advances in Cryptology — ASIACRYPT '91, Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 139–148. `doi:10.1007/3-540-57332-1_11`.

[6] K. Ohta, T. Okamoto, Multi-signature schemes secure against active insider attacks, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 82 (1) (1999) 21–31.

[7] A. Boldyreva, Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme, in: Public Key Cryptography – PKC 2003, Vol. 2567 of Lecture Notes in Computer Science, Springer, 2003, pp. 31–46. `doi:10.1007/3-540-36288-6_3`.

[8] M. Bellare, G. Neven, Multi-signatures in the plain public-key model and a general forking lemma, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 390–399. doi:10.1145/1180405.1180453.
URL https://doi.org/10.1145/1180405.1180453

[9] J. Camenisch, M. Stadler, Efficient group signature schemes for large groups, in: B. S. Kaliski (Ed.), Advances in Cryptology — CRYPTO '97, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 410–424.

[10] D. Chaum, E. van Heyst, Group signatures, in: Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings, Vol. 547 of Lecture Notes in Computer Science, Springer, 1991, pp. 257–265. doi:10.1007/3-540-46416-6_22.

[11] Y. Desmedt, Y. Frankel, Threshold cryptosystems, in: G. Brassard (Ed.), Advances in Cryptology — CRYPTO' 89 Proceedings, Springer New York, New York, NY, 1990, pp. 307–315. doi:10.1007/0-387-34805-0_28.

[12] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Robust threshold DSS signatures, in: U. Maurer (Ed.), Advances in Cryptology — EUROCRYPT '96, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 354–371. doi:10.1007/3-540-68339-9_31.

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

[14] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: E. Biham (Ed.), Advances in Cryptology — EUROCRYPT 2003, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 416–432. doi:10.1007/3-540-39200-9_26.

[15] R. L. Rivest, A. Shamir, Y. Tauman, How to leak a secret: Theory and applications of ring signatures, in: Theoretical Computer Science, Springer, 2006, pp. 164–186. doi:10.1007/11685654_7.

[16] D. Boneh, M. Drijvers, G. Neven, Compact multi-signatures for smaller blockchains, in: T. Peyrin, S. Galbraith (Eds.), Advances in Cryptology – ASIACRYPT 2018, Springer International Publishing, Cham, 2018, pp. 435–464. doi:10.1007/978-3-030-03329-3_15.

[17] S. Micali, K. Ohta, L. Reyzin, Accountable-subgroup multisignatures: Extended abstract, in: Proceedings of the 8th ACM Conference on Computer and Communications Security, CCS '01, Association for Computing Machinery, New York, NY, USA, 2001, p. 245–254. doi:10.1145/501983.502017.
URL https://doi.org/10.1145/501983.502017

[18] D. Boneh, C. Komlo, Threshold signatures with private accountability, in: Advances in Cryptology – CRYPTO 2022, Part IV, Vol. 13510 of Lecture Notes in Computer Science, Springer, 2022, full version: Cryptology ePrint Archive, Paper 2022/1636. doi:10.1007/978-3-031-15985-5_19.

[19] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing, J. Cryptol. 17 (4) (2004) 297–319. doi:10.1007/s00145-004-0314-9.
URL https://doi.org/10.1007/s00145-004-0314-9

[20] D. Boneh, A. Partap, L. Rotem, Proactive refresh for accountable threshold signatures, in: J. Clark, E. Shi (Eds.), Financial Cryptography and Data Security (FC 2024), Vol. 14745 of Lecture Notes in Computer Science, Springer, Cham, 2025, pp. 140–159. doi:10.1007/978-3-031-78679-2_8.

[21] D. Boneh, A. Partap, B. Waters, Accountable multi-signatures with constant size public keys, in: T. Jager, J. Pan (Eds.), Public-Key Cryptography – PKC 2025, Vol. 15675 of Lecture Notes in Computer Science, Springer, Cham, 2025, pp. 32–65. doi:10.1007/978-3-031-91823-0_2.

[22] J. Wang, J. Zhou, Z. Cao, X. Dong, Traceable multi-signature scheme with distributed combiners and lightweight setup for accountable blockchain transactions, in: H. Xiong, L. Zhou, M. Atiquzzaman (Eds.), Information Security Practice and Experience (ISPEC 2025), Vol. 16418 of Lecture Notes in Computer Science, Springer, Singapore, 2026, pp. 296–318. doi:10.1007/978-981-95-9284-5_17.

[23] X. Chen, P. Luo, H. Li, W. Susilo, Q. Huang, A non-interactive identity-based accountable-subgroup multi-signature scheme from lattices, Science China Information Sciences 69 (2026) 192105. doi:10.1007/s11432-025-4692-7.

[24] A. R. Ağırtaş, O. Yayla, A lattice-based accountable subgroup multi-signature scheme with verifiable group setup, Cryptology ePrint Archive, Paper 2024/014, accessed 23 November 2026 (2024).
URL https://eprint.iacr.org/2024/014

[25] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in: 28th Annual Symposium on Foundations of Computer Science (SFCS 1987), IEEE, 1987, pp. 427–438.

[26] A. R. Ağırtaş, O. Yayla, Pairing-based accountable subgroup multi-signatures with verifiable group setup, Cryptology ePrint Archive, Paper 2022/018, accessed 23 November 2026 (2022).
URL https://eprint.iacr.org/2022/018

[27] A. R. Ağırtaş, Verifiable accountable subgroup multi-signatures, Ph.D. thesis, Institute of Applied Mathematics, Middle East Technical University, Ankara, Turkey, accessed 23 November 2026 (2024).
URL https://open.metu.edu.tr/handle/11511/108469

[28] G. Maxwell, A. Poelstra, Y. Seurin, P. Wuille, Simple Schnorr multi-signatures with applications to Bitcoin, Designs, Codes and Cryptography 87 (09 2019). doi:10.1007/s10623-019-00608-x.

[29] T. Ristenpart, S. Yilek, The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks, in: Advances in Cryptology – EUROCRYPT 2007, Vol. 4515 of Lecture Notes in Computer Science, Springer, 2007, pp. 228–245. doi:10.1007/978-3-540-72540-4_13.

[30] D. Boneh, J. Bradley, S. Gorbunov, R. S. Wahby, H. Wee, C. A. Wood, Z. Zhang, BLS signatures, Internet-Draft draft-irtf-cfrg-bls-signature-07, IRTF CFRG, accessed 23 November 2026 (Jul. 2026).
URL https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature/07/

[31] F. Baldimtsi, K. K. Chalkias, F. Garillot, J. Lindstrøm, B. Riva, A. Roy, M. Sedaghat, A. Sonnino, P. Waiwitlikhit, J. Wang, Subset-optimized BLS multi-signature with key aggregation, in: Financial Cryptography and Data Security (FC 2024), Vol. 14745 of Lecture Notes in Computer Science, Springer, 2025, pp. 188–205, full version: Cryptology ePrint Archive, Paper 2023/498. doi:10.1007/978-3-031-78679-2_10.

[32] A. R. Ağırtaş, O. Yayla, Compartment-based and hierarchical threshold delegated verifiable accountable subgroup multi-signatures, in: A. Dąbrowski, J. Pieprzyk, J. Pomykała (Eds.), Number-Theoretic Methods in Cryptology (NuTMiC 2024), Vol. 14966 of Lecture Notes in Computer Science, Springer, Cham, 2025, pp. 283–313. doi:10.1007/978-3-031-82380-0_10.

[33] R. Bacho, B. Wagner, Tightly secure non-interactive BLS multi-signatures, in: Advances in Cryptology – ASIACRYPT 2024, Vol. 15485 of Lecture Notes in Computer Science, Springer, 2025, pp. 397–422. doi:10.1007/978-981-96-0888-1_13.

[34] S.-Y. Tan, T.-S. Ng, S.-H. Heng, Efficient fork-free BLS multi-signature scheme with incremental signing, in: Provable and Practical Security (ProvSec 2024), Vol. 14903 of Lecture Notes in Computer Science, Springer, 2025, pp. 250–268. doi:10.1007/978-981-96-0954-3_13.

[35] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, Y. Zhang, hinTS: Threshold signatures with silent setup, in: 2024 IEEE Symposium on Security and Privacy (SP), IEEE, 2024. doi:10.1109/SP54263.2024.00057.

[36] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613. doi:10.1145/359168.359176.
URL http://doi.acm.org/10.1145/359168.359176

[37] C.-P. Schnorr, Efficient signature generation by smart cards, Journal of cryptology 4 (3) (1991) 161–174.

[38] A. Fiat, A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, in: Advances in Cryptology – CRYPTO '86, Vol. 263 of Lecture Notes in Computer Science, Springer, 1987, pp. 186–194. doi:10.1007/3-540-47721-7_12.

[39] D. Pointcheval, J. Stern, Security proofs for signature schemes, in: U. Maurer (Ed.), Advances in Cryptology — EUROCRYPT '96, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 387–398. doi:10.1007/3-540-68339-9_33.

[40] G. Fuchsbauer, E. Kiltz, J. Loss, The algebraic group model and its applications, in: Advances in Cryptology – CRYPTO 2018, Part II, Vol. 10992 of Lecture Notes in Computer Science, Springer, Cham, 2018, pp. 33–62. doi:10.1007/978-3-319-96881-0_2.

[41] G. Fuchsbauer, A. Plouviez, Y. Seurin, Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model, in: Advances in Cryptology – EUROCRYPT 2020, Vol. 12106 of Lecture Notes in Computer Science, Springer, Cham, 2020, pp. 63–95. doi:10.1007/978-3-030-45724-2_3.

[42] T. P. Pedersen, A threshold cryptosystem without a trusted party, in: Advances in Cryptology – EUROCRYPT '91, Vol. 547 of Lecture Notes in Computer Science, Springer, 1991, pp. 522–526. doi:10.1007/3-540-46416-6_47.

[43] C. Komlo, I. Goldberg, FROST: Flexible round-optimized Schnorr threshold signatures, in: Selected Areas in Cryptography – SAC 2020, Vol. 12804 of Lecture Notes in Computer Science, Springer, 2021. doi:10.1007/978-3-030-81652-0_2.

[44] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Secure distributed key generation for discrete-log based cryptosystems, Journal of Cryptology 20 (1) (2007) 51–83. doi:10.1007/s00145-006-0347-3.

[45] J.-S. Coron, On the exact security of full domain hash, in: Advances in Cryptology – CRYPTO 2000, Vol. 1880 of Lecture Notes in Computer Science, Springer, 2000, pp. 229–235. doi:10.1007/3-540-44598-6_14.

[46] A. Bagherzandi, J.-H. Cheon, S. Jarecki, Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS '08, Association for Computing Machinery, New

York, NY, USA, 2008, p. 449–458. doi:10.1145/1455770.1455827.
URL https://doi.org/10.1145/1455770.1455827

[47] D. Wagner, A generalized birthday problem, in: Advances in Cryptology – CRYPTO 2002, Vol. 2442 of Lecture Notes in Computer Science, Springer, 2002, pp. 288–304. doi:10.1007/3-540-45708-9_19.

[48] D. Boneh, M. Drijvers, G. Neven, Compact multi-signatures for smaller blockchains, Cryptology ePrint Archive, Paper 2018/483, full version of the ASIACRYPT 2018 paper. Accessed 23 November 2026 (2018).
URL https://eprint.iacr.org/2018/483

[49] K. Wedderburn, py_arkworks_bls12381: Python bindings for BLS12-381 curve operations using arkworks, version 0.5.0, Software, accessed 23 November 2026.
URL https://github.com/ethereum/py-arkworks-bls12381

[50] A. Faz-Hernandez, S. Scott, N. Sullivan, R. S. Wahby, C. A. Wood, Hashing to Elliptic Curves, RFC 9380, RFC Editor (Aug. 2023). doi:10.17487/RFC9380.

[51] A. Vlasov, K. Olson, A. Stokes, A. Sanso, EIP-2537: Precompile for BLS12-381 curve operations, Ethereum Improvement Proposals, no. 2537, accessed 23 November 2026 (Feb. 2020).
URL https://eips.ethereum.org/EIPS/eip-2537

[52] Nomic Foundation, Hardhat: Ethereum development environment, version 2.29, Software, accessed 23 November 2026.
URL https://hardhat.org