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

Ahmet Ramazan Ağırtaş[1,2]
agirtas.ramazan@metu.edu.tr

and

Oğuz Yayla [1]
oguz@metu.edu.tr

[1]Institute of Applied Mathematics, Middle East Technical University, Ankara, Turkey
[2]Nethermind

April 25, 2023

Abstract

An accountable subgroup multi-signature is a kind of multi-signature scheme in which any subgroup $\mathcal{S}$ of a group $\mathcal{G}$ of potential signers jointly sign a message m, ensuring that each member of $\mathcal{S}$ is accountable for the resulting signature. In this paper, we propose three novel pairing-based accountable subgroup multi-signature (ASM) schemes, which are secure against existential forgery under chosen-message attacks and computational co-Diffie-Hellman assumption. In the first one, we use Feldman's verifiable secret sharing scheme as an implicit authentication and proof-of-possession for setting up group $\mathcal{G}$. In the second one, the members participating in authentication are decided by the subgroup. In the third one, we consider a designated combiner managing the authentication process. All schemes we propose here require fewer computations in the signature generation, signature aggregation, and verification phases than the pairing-based ASM scheme proposed by Boneh, Drijvers and Neven. Moreover, our first and third ones solve the open problem of constructing an ASM scheme in which the subgroup $\mathcal{S}$ of signers is unknown before the signature generation. Besides, we give a method of eliminating the combiner in case of knowing the subgroup of signers $\mathcal{S}$ in advance. Further, we extend our proposed schemes to aggregated versions. For N accountable subgroup multi-signatures, aggregated versions of our proposed schemes output an aggregated signature with the size of a single group ($\mathbb{G}_1$) element and require $N+1$ pairings in aggregated signature verification. In contrast, the partially aggregated ASM scheme of Boneh, Drijvers and Neven gives an aggregated signature with the size of $N+1$ group elements and requires $2N+1$ pairings in aggregated signature verification.

Keywords: accountable subgroup multi-signatures, BLS signature, aggregatable signatures

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 [8, 20, 21] 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* [3, 16, 22, 23, 24, 27] 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 [3, 11, 14, 15], aggregate signatures [5], ring signatures (also threshold, linkable and traceable variants) [26, 7, 17, 13], accountable subgroup multi-signatures [4, 19], 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. An *accountable subgroup multi-signature (ASM)* [4, 19] is a multi-signature scheme in which any subgroup $\mathcal{S} \subseteq \mathcal{G}$ jointly sign a message m, ensuring that each member of $\mathcal{S}$ is accountable for the resulting signature. This notion was firstly defined by Micali et al. in [19] by proposing the first ASM scheme. In a more recent paper [4], Boneh, Drijvers and Neven proposed another ASM scheme which is based on BLS signature [6], and solves the open problem in [19], i.e. constructing an ASM scheme in which the subgroup of signers $\mathcal{S} \subseteq \mathcal{G}$ is not determined before the signature generation.

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. 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 [12], which transforms the BLS signature scheme into an ASM scheme. The proposed vASM scheme, which also solves the open problem in [19], requires fewer multiplications and bilinear pairings than the ASM schemes proposed in [4]. 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 [19], 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 multiplications and bilinear pairings than the ASM scheme in [4]. Moreover, we give a method of consecutive and cumulative signing that eliminates the designated combiner 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 multi-signatures. 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 [4] with the signature size of $N+1$ group elements and verification with $2N+1$ pairings.

The outline of the paper is as follows. In Section 2 we give a brief background information, including definitions of bilinear pairings, co-CDH/ψ-co-CDH problems, Feldman's Verifiable Secret Sharing (VSS) protocol, multi-signatures and accountable subgroup multi-signatures, generalized forking lemma, and BLS signature scheme. In Section 3 we summarize the ASM scheme given in [4]. Then, in Section 4.1, we give our vASM scheme and prove its security in the random oracle model. Moreover, in Section 5 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 using generalized forking lemma. In Section 6, we summarize the partial aggregated ASM (AASM) scheme given in [4], and discuss the aggregated versions of our proposed schemes and their security. Finally, in Section 7, 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.

2 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, multi-signatures and accountable subgroup multi-signatures, generalized forking lemma, and BLS signature scheme.

2.1 Bilinear pairings and computational hard problems

Let $\mathbb{G}_1, \mathbb{G}_2$ be two cyclic additive groups of prime order q and $\mathbb{G}_T$ be cyclic multiplicative group with the same order.

Definition 2.1. 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: $e(A, B) \neq 1$ for all $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

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 2.2 (Computational co-Diffie-Hellman Problem [5])**.** 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 2.3 (Computational ψ-co-Diffie-Hellman Problem [4])**.** 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.

2.2 Feldman's VSS Protocol

Feldman's verifiable secret sharing (VSS) scheme [12] 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 [28] 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 $\mathbb{F}_q$ be a finite field with prime order q and g be a primitive element in $\mathbb{F}_q$. The dealer shares a secret as follows:

- Chooses a polynomial of degree $t-1$ ($< q$),

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

  with distinct and nonzero $\alpha_k \in \mathbb{F}_q^*$ for $k = 0, \ldots, t-1$, 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

$$g^{f(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} C_k^{i^k}. \tag{2.1}$$

The received share is consistent with the shared secret only if (2.1) 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 $C_0 = g^s$, where g is the generator for the cyclic group, and s is the secret to be shared. This commitment may leak information about the secret s. 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 4.1 we use this protocol as an implicit authentication and the proof of possession method. We use only the sharing, committing and verifying phases of this protocol.

2.3 Multi-signatures and Accountable Subgroup Multi-signatures

Definition 2.4. A multi-signature scheme consists of four algorithms, i.e. *ParGen, KeyGen, Sign, and Verify.* Let $\mathcal{G} = \{P_1, \ldots, P_n\}$ be a set of n players.

- **ParGen**(1^λ) takes the security parameter λ as input, and outputs the public system parameters *par* including security parameter, hash functions, cyclic groups, generators, etc.

- **KeyGen**(par) takes the system parameters *par* as input, and outputs secret and public key pair, i.e. sk and pk.

- **Sign**(par, sk, m) is an interactive protocol which is run by $\mathcal{G}$, in two steps, as follows:

  - **Individual signature generation** takes the system parameters *par*, secret key sk_i and message m as inputs, and outputs the individual signature σ_i.

  - **Individual signature aggregation** takes a set of individual signatures $\{\sigma_i\}_{i\in\mathcal{G}}$ as inputs and outputs the multi-signature σ.

- **Verify**$(par, \{pk_j\}_{j\in\mathcal{G}}, \sigma, m)$ takes system parameter *par*, multi-signature σ, message m, and public keys of the players in $\mathcal{G}$ as inputs, and outputs 1 if it is valid or 0 otherwise.

For the definition of an accountable subgroup multi-signature scheme, we add an interactive group setup algorithm **GSetup**, which is a one-time protocol run by all the players in the group $\mathcal{G}$. Then we modify the **Sign** and **Verify** algorithms as follows.

Definition 2.5. An accountable subgroup multi-signature scheme is a tuple of five algorithms, that is *ParGen, KeyGen, GSetup, Sign, and Verify.* Let $\mathcal{G} = \{P_1, \ldots, P_n\}$ be a set of n players, and $\mathcal{PK} = \{pk_1, \ldots, pk_n\}$ is the set of public keys of all the users in group $\mathcal{G}$.

- **ParGen**(1^λ) takes the security parameter λ as input, and outputs the public system parameters par including security parameter, hash functions, cyclic groups, generators, etc.
- **KeyGen**(par) takes the system parameters par as input, and outputs secret and public key pair, i.e. sk and pk.
- **GSetup**$(par, sk_i, \mathcal{PK})$ is an interactive protocol which is run by all the players in $\mathcal{G}$. It takes system parameters par, secret keys sk_i and set of all public keys $\mathcal{PK}$, and outputs a membership key mk_i, a set of membership public keys MPK, and a set of commitments COM.
- **Sign**(par, mk_i, m) is an interactive protocol which is run by any subset $\mathcal{S} \subseteq \mathcal{G}$, in two steps, as follows:
  - **Individual signature generation** takes the system parameters par, membership key mk_i and message m as inputs, and outputs the individual signature σ_i.
  - **Individual signature aggregation** takes a set of individual signatures $\{\sigma_i\}_{i \in \mathcal{S}}$ as inputs and outputs the accountable subgroup multi-signature σ.
- **Verify**$(par, \text{MPK}, \text{COM}, \mathcal{S}, \sigma, m)$ takes system parameter par, multi-signature σ, message m, definition of the subset $\mathcal{S}$, the set of membership public keys MPK, and the commitment set COM as inputs, and outputs 1 if it is valid or 0 otherwise.

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 $P_i \in \mathcal{S}$ run the **Sign**$(\cdot)$ 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 **Verify**$(par, \text{MPK}, \text{COM}, \mathcal{S}, \sigma, m) = 1$. 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.

Setup: The challenger randomly picks n values and computes membership public keys MPK and the commitment set COM, with respect to the indices $\{1, \ldots, n\}$. Finally it runs the adversary $\mathcal{A}(par, \text{MPK}, \text{COM})$, where par is the system parameters.

Signature queries: The adversary $\mathcal{A}$ makes queries on any message m, for any subset $\mathcal{S} \subseteq \{1, \ldots, n\}$ of users, and challenger responds with valid signatures.

Output: The adversary $\mathcal{A}$ eventually outputs a subset of indices $\mathcal{S}$, a message m, and an accountable subgroup multi-signature σ. The adversary $\mathcal{A}$ wins the game if **Verify**$(par, \text{MPK}, \text{COM}, \mathcal{S}, \sigma, m) = 1$, where the message m has been never queried as part of a signing query before.

Definition 2.6. Let $\Pi = \{ParGen, KeyGen, GSetup, Sign, Verify\}$ be an accountable subgroup multi-signature scheme. We say that an adversary $\mathcal{A}$ is a (t, q_H, q_S, ϵ)-forger if it runs in time t, makes at most q_H random oracle queries and q_S signing queries, and wins the above game with probability at least ϵ. We say that the accountable subgroup multi-signature scheme is (t, q_H, q_S, ϵ)-secure if no such adversary exists.

2.4 Generalized forking lemma

In order to prove the security of Schnorr-based signature schemes, Pointcheval and Stern firstly defined the forking lemma in [25]. Bellare and Neven generalized this lemma in [2]. In the security proofs of schemes in [4], the authors use the lemma in [1], which is a generalization of the forking lemma by Bagherzandi, Cheon, and Jarecki. We also use the latter generalized forking lemma in the security proofs of our two constructions.

Let $\mathcal{A}$ be an algorithm that interacts with random-oracle and takes inp as input. Let $f = (\rho, h_1, \ldots, h_{q_H})$ be the randomness used in the process of $\mathcal{A}$. Let ρ be $\mathcal{A}$'s random tape, h_i be the response to $\mathcal{A}$'s i-th hash query, q_H be the maximal number of hash queries. Let Ω be the space of all randomness vectors like f, and $f|_i$ be defined as $f|_i = (\rho, h_1, \ldots, h_{i-1})$ for any $i \leq q_H$. $\mathcal{A}(inp, f)$ is considered as successful if it returns a pair $(J, \{out_j\}_{j \in J})$, where J is a non-empty multi-set of indexes with $|J| = n$ and $\{out_j\}_{j \in J}$ is the multi-set of side outputs. Let ϵ be the probability that $\mathcal{A}(inp, f)$ is successful for fresh randomness $f \xleftarrow{\$} \Omega$ and for an input $inp \xleftarrow{\$}$ IG generated by an input generator IG. On input inp, the generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ proceeds as follows:

- $f = (\rho, h_1, \ldots, h_{q_H}) \xleftarrow{\$} \Omega$
- $(J, \{out_j\}_{j \in J}) \leftarrow \mathcal{A}(inp, f)$
- If $J = \emptyset$, then output "$fail$"
- Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \leq \ldots \leq j_n$.
- For $i = 1$ to n do
  - $succ_i \leftarrow 0$,
  - $k_i \leftarrow 0$,
  - $k_{max} = 8nq_H/\epsilon . \ln(8n/\epsilon)$
  - Repeat until $succ_i = 1$ or $k_i > k_{max}$
    - $k_i = k_i + 1$
    - $f'' \xleftarrow{\$} \Omega$ such that $f''|_{j_i} = f|_{j_i}$
    - Let $f'' = (\rho, h_1, \ldots, h_{j_i - 1}, h''_{j_i}, \ldots, h''_{q_H})$
    - $(J'', \{out''_j\}_{j \in J''}) \leftarrow \mathcal{A}(inp, f'')$
    - If $h''_{j_i} \neq h_{j_i}$ and $J'' \neq \emptyset$ and $j_i \in J''$ then
      - $out'_{j_i} \leftarrow out''_{j_i}$
      - $succ_i \leftarrow 1$
- If $succ_i = 1$ for all $i = 1, \ldots, n$, then output $(J, \{out_j\}_{j \in J}, \{out'_j\}_{j \in J})$
- Else output "$fail$"

The $\mathcal{GF}_{\mathcal{A}}$ algorithm is considered to be successful if it does not return "fail".

Lemma 1 (Generalized Forking Lemma [1])**.** *Let IG be a randomized algorithm generating inp. Let $\mathcal{A}$ be also a randomized algorithm that makes at most q_H random-oracle queries in time τ, which succeeds with probability ϵ. If $q > 8nq_H/\epsilon$, then $\mathcal{GF}_{\mathcal{A}}(inp)$ runs in time at most $\tau . 8n^2 q_H/\epsilon \; ln(8n/\epsilon)$ and succeeds with probability at least $\epsilon/8$, where the probability is over the choice of $inp \xleftarrow{\$}$ IG and the coins of $\mathcal{GF}_{\mathcal{A}}$.*

The forking lemma tells us that if an adversary obtains a successful forgery, it can obtain another successful forgery on the same message but with different random oracle query values. We will use this lemma to prove the security of two of our schemes in the random oracle model.

2.5 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 function which maps any arbitrary binary string onto the group $\mathbb{G}_1$. 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 [6].

Definition 2.7. We say that an adversary $\mathcal{A}$ (t, q_S, q_H, ϵ)-breaks a signature scheme if it runs in time at most t, makes at most q_S signature queries and at most q_H hash function queries, and the success probability of $\mathcal{A}$ is at least ϵ. A signature scheme is (t, q_S, q_H, ϵ)-secure against existential forgery under adaptive chosen-message attacks if no such adversary exists.

Theorem 2.8 (Theorem 3.2. [6])**.** *If solving the co-CDH problem in $\mathbb{G}_1 \times \mathbb{G}_2$ is (t', ϵ')-hard, then the BLS signature scheme is (t, q_S, q_H, ϵ)-secure against existential forgery under adaptive chosen-message attacks, for*

$$t = t' - c_{\mathbb{G}_1}(q_H + 2q_S), \text{and}$$
$$\epsilon = \epsilon' e(q_S + 1).$$

where $c_{\mathbb{G}_1}$ is a constant that depends on $\mathbb{G}_1$, and e is the base of natural logarithm.

Although Theorem 2.8 was proved in [6] 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 [5]. 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 [4], and using PoP requires additional verification operations. It is not fully compatible with the applications in cryptocurrencies [18]. In order to eliminate these disadvantages, Boneh, Drijvers and Neven proposed in [4] 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.

3 Boneh-Drijvers-Neven ASM Scheme

This section follows the notation given in both [4] 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 [4] 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 his/her individual signature on the message m

$$s_i = H_0(apk, m)^{sk_i} \cdot mk_i, \tag{3.1}$$

   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, she 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

$$e\Big(H_0(apk, m), pk\Big) \cdot e\Big(\prod_{j \in \mathcal{S}} H_2(apk, j), apk\Big) \stackrel{?}{=} e(s, g_2). \tag{3.2}$$

Theorem 3.1 (Theorem 3. [4]). *ASM scheme is unforgeable under the ψ-co-CDH problem (Definition 2.3) in the random oracle model. More precisely, ASM scheme is $(\tau, q_H, q_S, \epsilon)$-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if ψ-co-CDH problem is $(\tau + q_H \cdot \max(\tau_{exp_2^1}, \tau_{exp_1^2}) + q_G(l-1)\tau_{exp_1} + q_S(\tau_{exp_2^l} + \tau_{exp_1}) + 2\tau_{pair} + \tau_{exp_1^3}) \cdot 8q_H^2/((1-(q_S+q_H)/q) \cdot \epsilon) \cdot \ln(8q_H/((1-(q_S+q_H)/q)\epsilon)), (1-(q_S+q_H)/q) \cdot \epsilon/(8q_H))$-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 τexp_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 3.1 by [4]. 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 (3.1) is a BLS signature on (apk, m) by $|\mathcal{S}|$ signers; on the other hand, the second part mk_i is a 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 [4]. 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 $y_i = g_2^{sk_i}$, and constructs the PoP by $\pi_i = H_3(y_i)^{sk_i}$, 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 (y_i, π_i). In order to compute the aggregated public key (apk) of the group, they first check $e(H_3(y_i), y_i) \stackrel{?}{=} e(\pi_i, g_2)$, 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 [4], each user's public key consists of two group elements, and each user computes two extra pairings before computing the aggregated public key apk.

4 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 his 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.

4.1 vASM: An ASM scheme with VSS based group setup

We give the steps of the vASM scheme 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:

   - 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 $\alpha_k^{(i)}$'s are all nonzero and distinct, for $k = 1, \ldots, n-1$.
   - Computes the set of commitments $\text{COM}_i := \{C_k^{(i)} = g_2^{\alpha_k^{(i)}} | k = 0, \ldots, n-1\}$.
   - Sends $(f_i(j), \text{COM}_i)$ to j-th user in $\mathcal{G}$, for $j = 1, \ldots, n$.
   - After receiving $(f_j(i), \text{COM}_j)$,
     - computes the membership key $mk_i = \sum_{j \in \mathcal{G}} f_j(i)$.
     - computes $\text{COM} := \{C_k = \prod_{j \in \mathcal{G}} C_k^{(j)} | k = 0, \ldots, n-1\}$.
   - Checks:

     (a) $C_0 \stackrel{?}{=} \prod_{i \in \mathcal{G}} pk_i$

     (b) $g_2^{mk_i} \stackrel{?}{=} \prod_{k=0}^{n-1} C_k^{i^k}$
   - If either (a) or (b) fails, then she aborts. Else, she makes COM and set of membership public keys mpk_i's public. Define $\text{MPK} = \{mpk_i\}_{i \in \mathcal{G}}$. Note that these public keys can also be computed by the verifier, i.e. $mpk_i = g_2^{mk_i} = \prod_{k=0}^{n-1} C_k^{i^k}$.

3. Signature Generation: A signer $i \in \mathcal{G}$ computes his/her individual signature $s_i = H_0(m)^{mk_i}$ 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 the set of signers $\mathcal{S} \subseteq \mathcal{G}$. Then, she computes the aggregated subgroup multi-signature $\sigma = \prod_{i \in \mathcal{S}} s_i$.

5. Verification: Anyone, who is given $\{par, \mathrm{MPK}, \mathrm{COM}, \mathcal{S}, m, \sigma\}$, can verify the signature σ by checking

$$e\big(H_0(m), \prod_{i\in\mathcal{S}} mpk_i\big) \stackrel{?}{=} e(\sigma, g_2). \tag{4.1}$$

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

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

Remarks on vASM

1. Unlike the threshold multi-signatures [3, 11, 14, 15], ASM schemes [4, 19] 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 $|\mathcal{S}|$ [19].

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. Notice that $C_0 \stackrel{?}{=} \prod_{i\in\mathcal{G}} pk_i$ can be satisfied only if the users know their secret keys. Therefore the consistency checks (a) and (b) in the group setup phase provide proof of possession for each user and force all users to be honest. Hence, in the vASM scheme, no user can set a special rogue key.

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 (4.1) as

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

   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 2.2). In order to avoid this vulnerability, the polynomials f_i in the group setup phase of the vASM scheme can be set to degree $r \geq n-1$. In this way, at most $r-n+1$ 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.

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 his secret key sk_i to sign messages and to participate in any multi-signatures; on the other hand, he participates in the vASM scheme with his membership key mk_i.

4.2 Security

We follow the security reduction of the BLS signature scheme given in [6] with a few modifications to prove the security of the proposed scheme in Section 4.1. First, we give Theorem 4.1 which states the security reduction of our proposed vASM scheme.

Theorem 4.1. *If the ψ-co-CDH problem (Definition 2.3) in $\mathbb{G}_1 \times \mathbb{G}_2$ is (t', ϵ')-hard, then vASM scheme is (t, q_H, q_S, ϵ)-secure (see Definition 2.6) against existential forgery under adaptive chosen message attacks in the random oracle model for all t and ϵ satisfying*

$$t \leq t' - t_{exp_1}(q_H + (n+1)q_S) \text{ and } \epsilon \geq e(q_S + 1) \cdot \epsilon',$$

where t_{exp_1} is the time required by an exponentiation in $\mathbb{G}_1$ and n is the maximum number of potential signers involved in a vASM signature.

Proof. Let $\mathcal{F}$ be a forger that (t, q_H, q_S, ϵ)-breaks the vASM signature scheme. We construct an algorithm $\mathcal{A}$ which (t', ϵ')-breaks the ψ-co-CDH problem (see Definition 2.3). Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups and g_1, g_2 be their generators, respectively as in Definition 2.1. Algorithm $\mathcal{A}$ is given a triplet (g_2, $A = g_2^{\alpha}$, $B = g_1^{\beta}$), and access to the oracle $\mathcal{O}^{\psi}(.)$ which takes $g_2^x \in \mathbb{G}_2$ as input and outputs $g_1^x \in \mathbb{G}_1$. We assume without loss of generality that the set of indices of our target group is $\{1, 2, \ldots, n\}$.

Setup. Algorithm $\mathcal{A}$ proceeds as follows.

- It chooses n random $r_j \xleftarrow{\$} \mathbb{Z}_q$ for $j = 1, 2, \ldots, n$.
- It computes corresponding n membership public keys $mpk_j = A \cdot g_2^{r_j}$ for $j = 1, \ldots, n$.
- Algorithm $\mathcal{A}$ computes the commitment set $\text{COM} = \{C_0, \ldots, C_{n-1}\}$ with respect to the membership public keys using the system of equations below.

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 2 & \ldots & 2^{n-1} \\ & & \vdots & \\ 1 & n & \ldots & n^{n-1} \end{bmatrix} \cdot \begin{bmatrix} C_0 \\ C_1 \\ \vdots \\ C_{n-1} \end{bmatrix} = \begin{bmatrix} mpk_1 \\ mpk_2 \\ \vdots \\ mpk_n \end{bmatrix}$$

Because the above leftmost matrix is an $n \times n$ Vandermonde matrix and its entries are the powers of the indices of the membership public keys, i.e. they are distinct; hence it is invertible. Since the rightmost matrix is known, $\mathcal{A}$ can compute the matrix in the middle, i.e. the commitment set $\text{COM} = \{C_0, C_1, \ldots, C_{n-1}\}$. (Note that we operate in additive groups, but we use a multiplicative notation as common in the literature. Therefore, matrix operation above gives the desired result for the vASM commitment set.)

- It gives the set of membership public keys MPK $= \{mpk_1, \ldots, mpk_n\}$ and the commitment set COM to the forger $\mathcal{F}$.

Hash query. Algorithm $\mathcal{A}$ maintains a list $\mathcal{L}$ of tuples $\langle m_i, w_i, b_i, c_i \rangle$ for the i-th query, where m_i, w_i, b_i, c_i are defined below. The list $\mathcal{L}$ is initially empty, and when $\mathcal{F}$ queries the oracle H, for a value $m \in \{0,1\}^*$, $\mathcal{A}$ responds as follows.

1. If the query m_i has already been made before, $\mathcal{A}$ looks up to the list $\mathcal{L}$, finds the tuple $\langle m_i, w_i, b_i, c_i \rangle$, and responds with $H(m_i) = w_i \in \mathbb{G}_1$.
2. Otherwise, $\mathcal{A}$ generates a random $c_i \xleftarrow{\$} \{0,1\}$ with probability $\Pr[c_i = 0] = 1/(q_S + 1)$. Then, $\mathcal{A}$ picks a random $b_i \xleftarrow{\$} \mathbb{Z}_q$, and computes $w_i \leftarrow B^{1-c_i} \psi(g_2)^{b_i}$.
3. It adds the tuple $\langle m_i, w_i, b_i, c_i \rangle$ to the list $\mathcal{L}$.

Signature query. $\mathcal{A}$ responds to $\mathcal{F}$'s i-th signature query $(m_i, \mathcal{S}_i)$, where m_i is the message to be signed and $\mathcal{S}_i$ is the set of indices of the subgroup of signers, as follows:

1. It runs the above hash query algorithm to get the tuple $\langle m_i, w_i, b_i, c_i \rangle$. If $c_i = 0$, then it aborts.
2. Otherwise, it defines $\sigma_i = \prod_{j \in \mathcal{S}_i} \sigma_j$, where $\sigma_j = \psi(A)^{b_i} \psi(g_2)^{r_j b_i} \in \mathbb{G}_1$, and $j \in \mathcal{S}_i$. Note that $\sigma_i = w_i^{\sum_{j \in \mathcal{S}_i} \alpha + r_j}$ and therefore σ_i is a valid vASM signature on message m_i by the subgroup $\mathcal{S}_i$. Because
$$\begin{aligned} e(\sigma_i, g_2) &= e(w_i^{\sum_{j \in \mathcal{S}_i} \alpha + r_j}, g_2) \\ &= e(w_i, g_2^{\sum_{j \in \mathcal{S}_i} \alpha + r_j}) \\ &= e(w_i, \prod_{j \in \mathcal{S}_i} A \cdot g_2^{r_j}) \\ &= e(H(m_i), \prod_{j \in \mathcal{S}_i} mpk_j) \end{aligned}$$
as in the verification equation of the vASM scheme.
3. Algorithm $\mathcal{A}$ sends σ_i to forger $\mathcal{F}$.

Output. Forger $\mathcal{F}$ outputs a tuple $(\sigma_f, m_f, \mathcal{S}_f)$, where σ_f is a valid signature on a message m_f by a subgroup $\mathcal{S}_f$.

1. If the signature query has already been made on message m_f before, then $\mathcal{A}$ aborts.
2. If there is no tuple in the list $\mathcal{L}$ containing m_f, $\mathcal{A}$ runs the hash query algorithm for m_f.
3. $\mathcal{A}$ checks if σ_f is a valid signature on m_f by the signers in $\mathcal{S}_f$, i.e.
$$e(H(m_f), \prod_{j \in \mathcal{S}_i} mpk_j) = e(\sigma_f, g_2).$$
4. If it is not valid, $\mathcal{A}$ aborts.
5. Otherwise, $\mathcal{A}$ finds the tuple $\langle m_f, w, b, c \rangle$ in the list $\mathcal{L}$.
   - If $c = 1$, then $\mathcal{A}$ aborts.

- Otherwise, we have $c = 0$ and $H(m_f) = w = B.\psi(g_2)^b$. This means that

$$\sigma_f = B^{\sum_{j\in\mathcal{S}_f} \alpha+r_j} \cdot \psi(g_2)^{b\sum_{j\in\mathcal{S}_f} \alpha+r_j}.$$

- Algorithm $\mathcal{A}$ computes B^α as follows.

$$B^\alpha = \left(\frac{\sigma_f}{B^{\sum_{j\in\mathcal{S}_f} r_j} \cdot \psi(A)^{b|\mathcal{S}_f|} \cdot \psi(g_2)^{\sum_{j\in\mathcal{S}_f} r_j b}}\right)^{|\mathcal{S}_f|^{-1}}$$

It is easy to check that the value on the right-hand side is indeed equivalent to $B^\alpha = g_1^{\alpha\beta}$:

$$\begin{aligned}
\left(\frac{\sigma_f}{B^{\sum_{j\in\mathcal{S}_f} r_j} \cdot \psi(A)^{b|\mathcal{S}_f|} \cdot \psi(g_2)^{\sum_{j\in\mathcal{S}_f} r_j b}}\right)^{|\mathcal{S}_f|^{-1}} &= \left(\frac{B^{\sum_{j\in\mathcal{S}_f} \alpha+r_j} \cdot \psi(g_2)^{b\sum_{j\in\mathcal{S}_f} \alpha+r_j}}{g_1^{\sum_{j\in\mathcal{S}_f} \beta r_j} \cdot g_1^{b\alpha|\mathcal{S}_f|} \cdot g_1^{\sum_{j\in\mathcal{S}_f} r_j b}}\right)^{|\mathcal{S}_f|^{-1}} \\
&= \left(\frac{g_1^{\alpha\beta|\mathcal{S}_f|} \cdot g_1^{\beta\sum_{j\in\mathcal{S}_f} r_j} \cdot g_1^{b\alpha|\mathcal{S}_f|} \cdot g_1^{b\sum_{j\in\mathcal{S}_f} r_j}}{g_1^{\beta\sum_{j\in\mathcal{S}_f} r_j} \cdot g_1^{b\alpha|\mathcal{S}_f|} \cdot g_1^{b\sum_{j\in\mathcal{S}_f} r_j}}\right)^{|\mathcal{S}_f|^{-1}} \\
&= \left(g_1^{\alpha\beta|\mathcal{S}_f|}\right)^{|\mathcal{S}_f|^{-1}} \\
&= g_1^{\alpha\beta}
\end{aligned}$$

Above, we gave the construction of Algorithm $\mathcal{A}$. Now we give the success probability and the running time of it. The probability of $\mathcal{A}$ aborts in the signature query phase is equivalent to the probability that $c_i = 0$. From the hash query algorithm we know that $Pr[c_i = 0] = 1/(q_S + 1)$. Therefore the probability that $\mathcal{A}$ does not abort after the i-th query will be $(1 - 1/(q_S + 1))^i$. Since $\mathcal{F}$ makes q_S queries, the probability that $\mathcal{A}$ does not abort after the q_S-th query is at least $(1-1/(q_S+1))^{q_S} \geq 1/e$. Moreover, by Definition 2.6, the probability that $\mathcal{F}$ outputs a valid forgery is at least ϵ. After a valid forgery, $\mathcal{A}$ aborts if $c_i = 1$, which has probability $1 - 1/(q_S + 1)$. Therefore, given a valid forgery, the probability that $\mathcal{A}$ does not abort is $1/(q_S + 1)$. Hence, the overall success probability of Algorithm $\mathcal{A}$ is $1/e \cdot \epsilon \cdot 1/(q_S + 1)$, that is $\epsilon/(e(q_S + 1) \geq \epsilon'$, as desired.

Moreover, $\mathcal{A}$'s running time is nearly identical to $\mathcal{F}$'s running time. In addition to $\mathcal{F}$'s running time, $\mathcal{A}$'s running time includes the time required to respond to (q_H+q_S) hash queries and q_S signature queries. Each requires exponentiation in $\mathbb{G}_1$, which takes t_{exp_1} running time. Note that each hash query requires a single exponentiation while each signature query requires at most n exponentiations. Therefore the overall running time of the algorithm $\mathcal{A}$ is $t + t_{exp_1}((q_H + (n + 1)q_S)) \leq t'$ as desired.

□

5 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 3, 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 (3.1) 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 (3.2). In the following, we describe two scenarios.

5.1 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 3. The ASMwSA is as follows:

1. Key Generation: Identical to the Key Generation in Section 3.

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

   - 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 his/her 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 $\sigma = \prod_{i \in \mathcal{S}} s_i$ and $spk = \prod_{i \in \mathcal{S}} pk_i^{a_i}$.

5. Verification: Anyone who is given $\{par, apk, spk, \mathcal{S}, m, \sigma\}$ can verify the signature σ by checking

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

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

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

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 her 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 3, which also results in a legitimate aggregated signature. We note that this eliminates 1 round of transmission cost.

2. In the signature generation phase, $H_0(apk, m)$ may be replaced by $H_0(spk, m)$ because the subgroup $\mathcal{S}$ is known in advance.

3. We note that the user $i \in \mathcal{G}$ may compute her 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.

5.1.1 Security

We follow the security reduction of the MSP signature scheme given in [4] with a few modifications to prove the security of the proposed scheme in Section 5.1. Below, we give Theorem 5.1 which states the security reduction of our proposed ASMwSA scheme.

Theorem 5.1. *ASMwSA is unforgeable (as defined in Definition 2.6) under the co-CDH problem (Definition 2.2) in the random oracle model. More precisely, ASMwSA is (t, q_H, q_S, ϵ)-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if co-CDH problem is*

$$(t + t_{exp_2^n} + q_H \cdot t_{exp_1^2} + q_S \cdot t_{exp_1} + (l \cdot t_{mul_1} + 2t_{pair})) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon),\ \epsilon/(8q_H))\text{-hard},$$

where $n = |\mathcal{G}|$ is the number of potential signers and $l = |\mathcal{S}|$ is the signers involved in an ASMwSA signature, t_{exp_1} and t_{exp_2} denote the time required to compute exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $t_{exp_1^i}$ and $t_{exp_2^i}$ denote the time required to compute i-multi-exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and t_{mul_1} denotes the multiplication in $\mathbb{G}_1$.

Proof. Suppose we have a (t, q_H, q_S, ϵ)-forger $\mathcal{F}$ against ASMwSA scheme. An algorithm $\mathcal{A}$, given $(A = g_1^\alpha, B_1 = g_1^\beta, B_2 = g_2^\beta)$ where $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$, and a randomness $f = \{\rho, h_1, \ldots, h_{q_S}\}$ as input, proceeds as follows.

- $\mathcal{A}$ picks a random index $k \xleftarrow{\$} \{1, \ldots, q_H\}$.
- $\mathcal{A}$ sets $pk_i = B_2$.
- $\mathcal{A}$ runs the forger $\mathcal{F}$ on input pk_i with random tape ρ.
- $\mathcal{A}$ receives $\mathcal{PK}$ from $\mathcal{F}$ such that $pk_i \in \mathcal{PK}$.
- $\mathcal{A}$ maintains three lists, i.e. $\mathcal{L}_0, \mathcal{L}_1$ and $\mathcal{L}_2$, which are initially empty.
- $\mathcal{A}$ responds $\mathcal{F}$'s j-th H_1-queries on $(pk_j, \mathcal{PK})$ for $j = 1, \ldots, q_H$ as follows:
  - If this is the first query on $(pk_j, \mathcal{PK})$ with $pk_j \in \mathcal{PK}$ and $j \neq i$, $\mathcal{A}$ chooses a random value $w_j \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(pk_j, \mathcal{PK}) = w_j$. If $j = i$, it sets $H_1(pk_i, \mathcal{PK}) = h_i$. $\mathcal{A}$ responds to $\mathcal{F}$ with $H_1(pk_j, \mathcal{PK})$.

– If this query is done before, then $\mathcal{A}$ responds to $\mathcal{F}$ with earlier values.

– For other types of queries, $\mathcal{A}$ responds with a random value in $\mathbb{Z}_q$.

– Add the response to the list $\mathcal{L}_1$.

- We assume that $\mathcal{F}$ makes no repeated H_2-queries. $\mathcal{A}$ responds $\mathcal{F}$'s H_2 queries of the form (apk, j) for $j = 1, \ldots, n$ as follows:

  – $\mathcal{A}$ responds with $g_1^{r_j^{(2)}}$, where $r_j^{(2)}$ is randomly selected from $\mathbb{Z}_q$,

  – Add $\langle (apk, j), H_2(apk, j), r_j^{(2)} \rangle$ to the list $\mathcal{L}_2$.

- We assume that $\mathcal{F}$ makes no repeated H_0-queries. $\mathcal{A}$ responds $\mathcal{F}$'s j-th H_0-query of the form (apk, m_j) for $j = 1, \ldots, q_H$ as follows:

  – If $j \neq k$ then responds with $g_1^{r_j^{(0)}}$, where $r_j^{(0)}$ is randomly selected from $\mathbb{Z}_q$.

  – If $j = k$, then responds with $A_0 = A/(g_1^{r_i^{(2)}})$, where $r_i^{(2)} \in \mathbb{Z}_q$ is the randomly selected value for the i-th user's H_2 query.

  – Add $\langle (apk, m_j), H_0(apk, m_j), r_j^{(0)} \rangle$ to the list $\mathcal{L}_0$.

- For signing queries of $\mathcal{F}$ on a message m_j and a subset $\mathcal{S}_j \subseteq \mathcal{G}$ for $1 \leq j \leq q_S$, $\mathcal{A}$ proceeds as follows:

  – If $i \notin \mathcal{S}_j$, then $\mathcal{A}$ aborts.

  – $\mathcal{A}$ looks up the list $\mathcal{L}_0$. If $H_0(apk, m_j) = A_0$, then it aborts.

  – If $H_2(apk, z) \notin \mathcal{L}_2$ for any $z \in \mathcal{S}_j$, then it aborts.

  – Otherwise $\mathcal{A}$ responds with $\sigma_i = \big(B_1^{r_j^{(0)}} \cdot B_1^{\sum_{u \in \mathcal{S}_j} r_u^{(2)}}\big)^{h_i}$.

  Actually, the resulting signature σ_i is a valid signature and can be verified by the public keys of the signers in $\mathcal{S}_j$. Because

$$\begin{aligned} e\big(\sigma_i, g_2\big) &= e\Big(\big(B_1^{r_j^{(0)}} \cdot B_1^{\sum_{u \in \mathcal{S}_j} r_u^{(2)}}\big)^{h_i}, g_2\Big) \\ &= e\big((H_0(apk, m_j) \cdot \prod_{z \in \mathcal{S}_j} H_2(apk, z))^{h_i \beta}, g_2\big) \\ &= e\big(H_0(apk, m_j) \cdot \prod_{z \in \mathcal{S}_j} H_2(apk, z), pk_i^{h_i}\big) \end{aligned}$$

  as in the verification equation of the ASMwSA scheme.

- Eventually $\mathcal{F}$ outputs a triplet $(\sigma_f, m_f, \mathcal{S}_f)$ where σ_f is an ASMwSA signature on the message m_f for the subgroup $\mathcal{S}_f$.

- $\mathcal{A}$ looks up into list $\mathcal{L}_0$ for m_f,

  – If $H_0(apk, m_f) \neq A_0$, then $\mathcal{A}$ aborts.

  – Otherwise, σ_f is a valid signature on m_f for the subgroup $\mathcal{S}_f$ with probability ϵ such that

$$e(H_0(apk, m_f) \cdot \prod_{j \in \mathcal{S}_f} H_2(apk, j), spk_f) = e(\sigma_f, g_2). \tag{5.1}$$

The algorithm's running time $\mathcal{A}$ is equivalent to the running time of the forger $\mathcal{F}$ plus some extra computations that $\mathcal{A}$ makes.

- Computing apk: The running time of computing the aggregated public key apk is $t_{exp_2^n}$, where n is the number of potential signers in the group $\mathcal{G}$.
- Hash queries:
  - For H_0 and H_2, the running time is at most $t_{exp_1^2}$.
  - Therefore, overall running time of the hash queries including H_0 and H_2 will be at most $q_H \cdot t_{exp_1^2}$.
- Signing queries:
  - Computing the signature takes t_{exp_1}. Therefore overall running time of the signature queries is $q_S \cdot t_{exp_1}$.
- For verification of the output of $\mathcal{F}$ takes $l \cdot t_{mul_1}$ and $2t_{pair}$.
- Therefore, overall running time of the algorithm $\mathcal{A}$ is $t+t_{exp_2^n}+q_H \cdot t_{exp_1^2}+q_S \cdot t_{exp_1}+(l \cdot t_{mul_1}+2t_{pair})$.

The probability that $\mathcal{A}$ does not abort is equivalent to the probability that $\mathcal{A}$ correctly guesses the hash index of the valid forgery of $\mathcal{F}$, which is $1/q_H$. Since the forger $\mathcal{F}$'s success probability is ϵ, the success probability of the algorithm $\mathcal{A}$ is ϵ/q_H.

Now we construct an algorithm $\mathcal{B}$ which solves co-CDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ on input a co-CDH instance $(A, B_1, B_2) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$ and a forger $\mathcal{F}$. The algorithm $\mathcal{B}$ actually runs generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ on input (A, B_1, B_2) and algorithm $\mathcal{A}$ as described above. $\mathcal{B}$ proceeds as follows

- If $\mathcal{GF}_{\mathcal{A}}$ outputs fail, then $\mathcal{B}$ aborts.
- If $\mathcal{GF}_{\mathcal{A}}$ outputs $(\{j_f\}, \{out\}, \{out'\})$, then
  - $\mathcal{B}$ parses out as $(\sigma, \mathcal{PK}, spk, a_1, \ldots, a_n)$ and out' as $(\sigma', \mathcal{PK}', spk', a'_1, \ldots, a'_{n'})$.
  - Note that out and out' were obtained from two executions of $\mathcal{A}$ with randomness f and f' such that $f|_{j_f} = f'|_{j_f}$ for some integer $j_f \leq q_S$. In other words, these executions are identical to the j_f-th H_1-query of the type that has not been queried before.
  - This means that the arguments of this query are identical, i.e. $\mathcal{PK} = \mathcal{PK}'$, and $n = n'$.
  - From the construction of $\mathcal{GF}_{\mathcal{A}}$, we know that $a_i = h_{j_f}$, $a'_i = h'_{j_f}$, and by the forking lemma (see Lemma 1) we have $a_i \neq a'_i$.
  - We know that $spk = \prod_{j \in \mathcal{S}} pk^{a_j}$ and $spk' = \prod_{j \in \mathcal{S}} pk^{a'_j}$.
  - Since the algorithm $\mathcal{A}$ assigned $H_1(pk_j, \mathcal{PK}) \leftarrow a_j$ for all $j \neq i$, we have $a_j = a'_j$, and therefore $spk/spk' = pk_i^{a_i - a'_i}$.
  - Notice that $\mathcal{A}$'s output (see (5.1)) satisfies both

$$e(H_0(apk, m_f) \cdot \prod_{j \in \mathcal{S}_f} H_2(apk, pk_j), spk_f) = e(\sigma_f, g_2)$$

    and

$$e(H_0(apk, m_f) \cdot \prod_{j \in \mathcal{S}_f} H_2(apk, pk_j), spk'_f) = e(\sigma'_f, g_2).$$

– Then we have

$$e(A, B_2^{(a_i - a_i')}) = e(\sigma_f/\sigma_f', g_2).$$

– Hence, $(\sigma_f/\sigma_f')^{1/(a_i - a_i')}$ is the solution to the co-CDH instance $(A = g_1^{\alpha}, B_1 = g_1^{\beta}, B_2 = g_2^{\beta})$ which is given to algorithm $\mathcal{B}$ as input.

Lemma 1 says that if $q > 8q_H/\epsilon$, then Algorithm $\mathcal{B}$ runs in time at most $(t + t_{exp_2^n} + q_H \cdot t_{exp_1^2} + q_S \cdot t_{exp_1} + (l \cdot t_{mul_1} + 2t_{pair})) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon)$, and succeeds with probability at most $\epsilon/(8q_H)$. □

5.2 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 3.

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

   - 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, she computes the aggregated subgroup multi-signature $\sigma = \prod_{i \in \mathcal{S}} s_i \cdot \prod_{i \in \mathcal{S}} \prod_{j \in \mathcal{S}} \mu_{ij}$ for $j \neq i$, and aggregated public key spk of the subgroup $\mathcal{S}$ as $spk = \prod_{i \in \mathcal{S}} pk_i^{a_i}$.

5. Verification: Anyone who is given $\{par, apk, spk, \mathcal{S}, m, \sigma\}$ can verify signature σ by checking

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

Since ASMwCA is a modified version of ASMwSA, its correctness property follows from the correctness of the ASMwSA schemes as shown in Section 5.1.

Remarks on ASMwCA

1. The subgroup $\mathcal{S} \in \mathcal{G}$ of the signers is determined by the combiner from the set of received individual signatures. Hence, no signer knows her 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 can not 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. l multi-exponentiations for each signer.

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 to scenarios in Sections 3 and 5.1, 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 she 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 her 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.

5.2.1 Security

Here we assume WLOG that the combiner is the algorithm $\mathcal{A}$. Then the security reduction of the following theorem will be identical to the security reduction of Theorem 5.1.

Theorem 5.2. *ASMwCA is unforgeable (as defined in Definition 2.6) under the co-CDH problem (Definition 2.2) in the random oracle model. More precisely, ASMwSA is (t, q_H, q_S, ϵ)-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if co-CDH problem is*

$$(t + t_{exp_2^n} + q_H \cdot t_{exp_1^2} + q_S \cdot t_{exp_1} + (l \cdot t_{mul_1} + 2t_{pair})) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon),\ \epsilon/(8q_H))\text{-hard},$$

where $n = |\mathcal{G}|$ is the number of potential signers and $l = |\mathcal{S}|$ is the signers involved in a ASMwCA signature, t_{exp_1} and t_{exp_2} denote the time required to compute exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $t_{exp_1^i}$ and $t_{exp_2^i}$ denote the time required to compute i-multi-exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and t_{mul_1} denotes the multiplication in $\mathbb{G}_1$.

Proof. As we state above, the only elements to be kept secret are the membership key components and the secret keys. Since the algorithm $\mathcal{A}$ acts as an honest signer, we may assume -without loss of generality- that Algorithm $\mathcal{A}$ also acts as the designated combiner. Hence, the proof follows from the security proof of Theorem 5.1. □

5.3 Eliminating the combiner

Consider 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 aggregated signature $s_{k_i} = s_{k_{i-1}} \cdot \big(H_0(spk, m)^{a_{k_i} sk_{k_i}} \cdot smk_{k_i}\big)$, where $s_{k_{i-1}}$ is the $(i-1)$-th aggregated signature computed by the signer $k_{i-1} \in \mathcal{S}$ and $s_{k_0} = 1_{\mathbb{G}_1}$.
- Computes the i-th aggregated public key $spk_{k_i} = spk_{k_{i-1}} \cdot pk_{k_i}^{a_{k_i}}$, where $spk_{k_{i-1}}$ is the $(i-1)$-th aggregated public key computed by the signer $k_{i-1} \in \mathcal{S}$ and $spk_{k_0} = 1_{\mathbb{G}_2}$.
- Sends (s_{k_i}, spk_{k_i}) to the signer k_{i+1} for $i < l$.
- Finally the last signer k_l outputs the pair s_{k_l}, spk_{k_l}. Then, the aggregated signature and the public key pair for $\mathcal{S}$ will be $\sigma := s_{k_l}$ and $spk := spk_{k_l}$. This process is shown in Figure 1.

In ASMwSA and ASMwCA, each signer sends her individual signature to the combiner. On the other hand, in this scenario, each signer sends the signature to another signer. Hence, this reduces the cost of network traffic and makes the channel traffic intermittent. However, in this case, each user sends two group elements instead of one, that is, the transmission size is doubled.

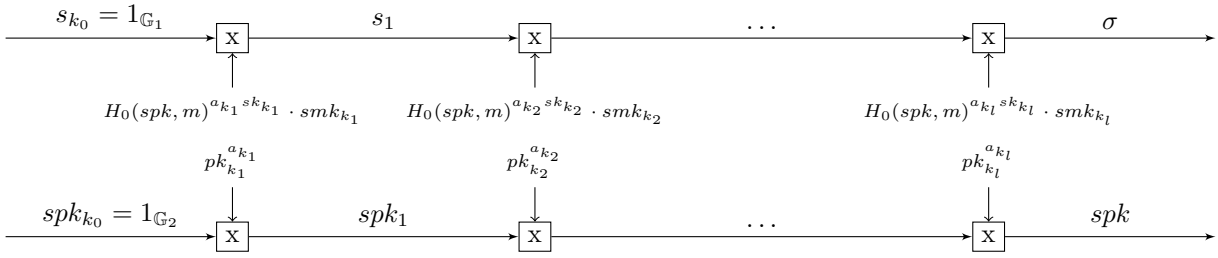

Figure 1: ASM scheme without a combiner

5.4 Advantages of subgroup-specific membership key

1. Group-specific membership keys in Boneh-Drijvers-Neven ASM scheme given in Section 3 are generated by one round of interactive protocol, in which all members participate. But in case of using subgroup-specific ones given in Sections 5.1 and 5.2, no interactive protocol is needed.

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 a big deal 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 $H_2(pk_j)$ instead of $H_2(apk, j)$ for computing the components of membership keys, registering/unregistering a member to/from the group $\mathcal{G}$ does not require a new group setup.

3. Users in $\mathcal{S} \subseteq \mathcal{G}$ do multiplications to get smk; but all group members in $\mathcal{G}$ need to do multiplications to get mk. If $|\mathcal{S}|$ is very small with respect to $|\mathcal{G}|$, then the computation of smk is easier. Similarly, spk can be calculated by less number of multi-exponentiation than apk.

6 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 3 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 6.1, we describe the AASM scheme that is given in [4].

6.1 Partially Aggregated ASM Scheme (AASM)

The aggregation of several ASM signatures is discussed in [4], 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 of the irrelevancy between membership keys and messages [4]. In addition to the phases of the ASM scheme, two more phases are defined in [4]. 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

$$\prod_{i=1}^{N} \Big(e\big(H_0(apk_i, m_i), pk_i^{b_i}\big) \cdot e\big(\prod_{j \in \mathcal{S}_i} H_2(apk_i, j), apk_i^{b_i}\big) \Big) \stackrel{?}{=} e(s, g_2). \tag{6.1}$$

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 (6.1) of the AASM scheme requires $(2N + 1)$ pairings.

6.2 Aggregated versions of proposed schemes

Our schemes defined in Sections 4.1, 5.1 and 5.2 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.

6.2.1 Aggregated vASM Scheme (AvASM)

The vASM scheme described in Section 4.1 can be extended to aggregated version AvASM as in Section 6.1. Let H_5 be the hash function defined in the previous section.

- Signature Aggregation: Given $(par, \{(\mathrm{MPK}_i, \mathcal{S}_i, m_i, \sigma_i)_{i=1}^N\})$. Output $\Sigma \leftarrow \prod_{i=1}^{N} \sigma_i$.
- Aggregate Signature Verification: Given $(par, \{(\mathrm{MPK}_i, \mathcal{S}_i, m_i)_{i=1}^N\}, \Sigma)$.
- Accept if and only if

$$\prod_{i=1}^{N} e\Big(H_0(m_i), \prod_{j \in \mathcal{S}_i} mpk_j \Big) = e(\Sigma, g_2). \tag{6.2}$$

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

$$\begin{aligned}
\prod_{i=1}^{N} e\Big(H_0(m_i), \prod_{j \in \mathcal{S}_i} mpk_j \Big) &= \prod_{i=1}^{N} e\Big(H_0(m_i), \prod_{j \in \mathcal{S}_i} g_2^{mk_j} \Big) \\
&= \prod_{i=1}^{N} e\Big(H_0(m_i), g_2^{\sum_{j \in \mathcal{S}_i} mk_j} \Big) \\
&= \prod_{i=1}^{N} e\Big(H_0(m_i)^{\sum_{j \in \mathcal{S}_i} mk_j}, g_2 \Big) \\
&= \prod_{i=1}^{N} e\Big(\sigma_i, g_2 \Big) \\
&= e\Big(\prod_{i=1}^{N} \sigma_i, g_2 \Big) \\
&= e(\Sigma, g_2)
\end{aligned}$$

Next, we reduce the security of the AvASM scheme to the security of the vASM scheme.

Theorem 6.1. *If the vASM scheme is unforgeable in the random oracle model, then the AvASM scheme is also unforgeable in the random oracle model. More precisely, AvASM is (t, q_H, q_S, ϵ)-secure against existential forgery under adaptive chosen message attacks in the random oracle model, for all t and ϵ satisfying*

$$t \leq t' - t_{exp_1}(q_H + (n+1)q_S) + q_S \cdot t_{exp_1^N} \text{ and } \epsilon \geq e(q_S + 1) \cdot \epsilon'$$

where t_{exp_1} is the time required by an exponentiation in $\mathbb{G}_1$ and n is the maximum number of potential signers involved in a single vASM signature.

Proof. Consider the forger $\mathcal{F}$ in the security proof of Theorem 4.1. We construct an algorithm $\mathcal{A}$ as in the security proof of Theorem 4.1, except that $\mathcal{F}$ now returns an aggregated vASM signature instead of a single vASM signature, i.e. an AvASM signature Σ, a set $\{\text{MPK}_j, \mathcal{S}_j, m_j\}_{j=1}^N$, and the set $\{\text{MPK}^*, \mathcal{S}^*, m^*\}$.

Assume that $\langle m^*, w^*, b^*, c^* \rangle$ is in the list $\mathcal{L}$ such that $c^* = 0$ and $w^* = B \cdot \psi(g_2)^b$. This gives us

$$e(B \cdot \psi(g_2)^b, \prod_{j \in \mathcal{S}^*} mpk_j) \cdot \prod_{i=1}^{n} e\left(H_0(m_i), \prod_{j \in \mathcal{S}_i} mpk_j\right) = e(\Sigma, g_2). \tag{6.3}$$

Then, $\mathcal{A}$ computes $\sigma \leftarrow \Sigma \cdot \prod_{i=1}^{N} \left(\prod_{j \in \mathcal{S}_i} \psi(mpk_j) \right)^{-\sum_{j \in \mathcal{S}_i} (r_j + b_i)}$, and this signature σ satisfies

$$e(H(m^*), \prod_{j \in \mathcal{S}^*} mpk_j) = e(\sigma, g_2), \tag{6.4}$$

which is a single vASM signature on message m^* by the subgroup S^*.

Note that the success probability is the same as the forger in the security proof of the vASM scheme. On the other hand, Algorithm $\mathcal{A}$ computes extra $t_{exp_1^N}$ multi-exponentiations for extracting σ from Σ. □

6.2.2 Aggregated versions of ASMwSA/ASMwCA Schemes (AASMwSA/AASMwCA)

The ASMwSA and ASMwCA schemes described in Section 5 can be extended to aggregated versions AASMwSA and AASMwCA as in Section 6.1. In addition to the phases of the described schemes in Section 5, signature aggregation and aggregate signature verification phases are given below:

- Signature Aggregation: Given $(par, \{(apk_i, spk_i, \mathcal{S}_i, m_i, \sigma_i)_{i=1}^N\})$, compute $\Sigma \leftarrow \prod_{i=1}^{N} \sigma_i$.
- Aggregate Signature Verification: Given $(par, \{(apk_i, spk_i, \mathcal{S}_i, m_i)_{i=1}^N\}, \Sigma)$, accept if and only if

$$\prod_{i=1}^{N} e\left(H_0(apk_i, m_i). \prod_{j \in \mathcal{S}_i} H_2(apk_i, j), spk_i\right) = e(\Sigma, g_2). \tag{6.5}$$

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

$$\begin{aligned}
\prod_{i=1}^{N} e\Big(H_0(apk_i, m_i).\prod_{j\in\mathcal{S}_i} H_2(apk_i, j), spk_i\Big) &= \prod_{i=1}^{N} e\Big(H_0(apk_i, m_i).\prod_{j\in\mathcal{S}_i} H_2(apk_i, j), spk_i\Big)\\
&= \prod_{i=1}^{N} e\Big(H_0(apk_i, m_i)^{\sum\limits_{j\in\mathcal{S}_i} a_j sk_j}\cdot\Big(\prod_{j\in\mathcal{S}_i} H_2(apk_i, j)\Big)^{\sum\limits_{j\in\mathcal{S}_i} a_j sk_j}, g_2\Big)\\
&= \prod_{i=1}^{N} e\Big(s_i\cdot smk_i, g_2\Big)\\
&= e\Big(\prod_{i=1}^{N}\sigma_i, g_2\Big)\\
&= e\Big(\Sigma, g_2\Big)
\end{aligned}$$

As we did in the security proof of AvASM scheme above, we next reduce the security of AASMwSA and AASMwCA schemes to the security of ASMwSA/ASMwCA schemes.

Theorem 6.2. *AASMwSA and AASMwCA are unforgeable (as defined in Definition 2.6) under the ψ-co-CDH problem (Definition 2.3) in the random oracle model. More precisely, AASMwSA and AASMwCA are (t, q_H, q_S, ϵ)-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if ψ-co-CDH problem is*

$$(t + q_H\cdot max(t_{exp_2^n}, t_{exp_1^2}) + n\cdot q_S\cdot(t_{exp_2^n} + t_{exp_1}) + 2t_{pair} + t_{exp_1^N})\cdot 8q_H^2/\epsilon\cdot\ln(8q_H/\epsilon),\ \epsilon/(8q_H))\text{-hard},$$

where n is the number of potential signers involved in a AASMwSA and ASMwCA signatures, t_{exp_1} and t_{exp_2} denote the time required to compute exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $t_{exp_1^i}$ and $t_{exp_2^i}$ denote the time required to compute i-multi-exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.

Proof. Consider the security proof of Theorem 5.1. We construct the same algorithm $\mathcal{A}$, but this time it's forger $\mathcal{F}$ gives an aggregate multi-signature, i.e. an aggregate multi-signature Σ, a set of tuples $\{(apk_i, spk_i, \mathcal{S}_i, m_i, \sigma_i)_{i=1}^N\}$, a set of public keys $\mathcal{PK}$, a subgroup $\mathcal{S}^*$ and a message m^*. We know that apk^* and spk^* can be generated from $\mathcal{PK}$ and $\mathcal{S}^*$.

Assume that the algorithm $\mathcal{A}$ guesses the k-th query $H_0(apk^*, m^*)$ correctly. Then the following equality holds

$$e(A_0, spk^*)\cdot\prod_{i=1}^{N} e\Big(H_0(apk_i, m_i).\prod_{j\in\mathcal{S}_i} H_2(apk_i, j), spk_i\Big) = e(\Sigma, g_2). \tag{6.6}$$

$\mathcal{A}$ looks up the tables $\mathcal{L}_0$ and $\mathcal{L}_2$:

- $\mathcal{L}_0$, for the pairs (apk_j, m_j), such that $H_0(apk_j, m_j) = g_1^{r_j^{(0)}}$.
- $\mathcal{L}_2$, for the pairs (apk_j, i_j), such that $H_2(apk_j, i_j) = g_1^{r_{i_j}^{(2)}}$.

Then $\mathcal{A}$ computes $\sigma \leftarrow \Sigma\cdot\prod\limits_{i=1}^{N}\mathcal{O}(spk_i)^{-\sum\limits_{j\in\mathcal{S}_i}(r_i^{(0)} + r_{i_j}^{(2)})}$, and this signature satisfies

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

Notice that we have A_0 for a single ASMwSA/ASMwCA forgery. Therefore, the rest will be the same as in the security proof of Theorem 5.1. The running time of Algorithm $\mathcal{A}$ will increase because of the extraction of the σ, which costs extra $t_{exp_1^N}$ running time. On the other hand, the success probability stays the same. □

7 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, the vASM scheme has $2n$ exponentiations and (n^2+n-2) multiplications $\mathbb{G}_2$ in the group setup.

It can also be seen from Table 1 that the vASM scheme requires fewer operations than ASM and ASM-PoP in the verification phase. 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 group setup, 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.

The ASMwCA scheme has efficient group setup and verification phases, but it requires more computations in the signature aggregation phase. On the other hand, it requires extra storage for (n^2-n-1) $\mathbb{G}_1$ elements.

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 consists 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 $2N+1$.

8 Conclusion

In this work, we propose a novel BLS-based ASM scheme (vASM) that is more efficient than the ones in [4] in terms of signature generation, signature aggregation, and verification. On the other hand, our vASM scheme requires a one-time group setup with more multiplications. We propose two more accountable subgroup multi-signature schemes with subgroup authentication (ASMwSA) and combiner authentication (ASMwCA), which are also more efficient, but they have storage and transmission disadvantages in comparison with the ones in [4]. Moreover, we compare the aggregated versions of the proposed schemes with the aggregated version of Boneh et al.'s ASM [4]. We see that aggregated versions of our schemes, i.e. AvASM, AASMwSA, and AASMwCA, are more efficient regarding aggregated signature size and verification time. According to the requirements of the system involving an ASM scheme, our schemes could be good alternatives, especially when faster verification is desired. It would be a good future work to add accountability to known multi-signature schemes by the methods given in this paper.

References

[1] Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Proceedings of the 15th ACM Conference*

Table 1: Comparison of the schemes

Phases	ASM [4]	ASM with PoP [4]	vASM	ASMwSA	ASMwCA
Key Generation	1 $\text{Exp}_{\mathbb{G}_2}$	For pk: 1 $\text{Exp}_{\mathbb{G}_2}$ For proof: 1 Hash (H_3) 1 $\text{Exp}_{\mathbb{G}_2}$	1 $\text{Exp}_{\mathbb{G}_2}$	1 $\text{Exp}_{\mathbb{G}_2}$	1 $\text{Exp}_{\mathbb{G}_2}$
Key Aggregation	n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$	For proof: n Hash (H_3) 2n pairings For apk: 1 Hash (H_4) $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$		n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$	n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$
Group Setup	n Hash (H_2) n $\text{Exp}_{\mathbb{G}_1}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_1}$	n Hash (H_2) n $\text{Exp}_{\mathbb{G}_1}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_1}$	$2n$ $\text{Exp}_{\mathbb{G}_2}$ (n^2+n-2) $\text{Mul}_{\mathbb{G}_2}$	n Hash (H_2) n $\text{Exp}_{\mathbb{G}_1}$	n Hash (H_2) n $\text{Exp}_{\mathbb{G}_1}$
Signature Generation	1 Hash (H_0) 1 $\text{Exp}_{\mathbb{G}_1}$ 1 $\text{Mul}_{\mathbb{G}_1}$	1 Hash (H_0) 1 $\text{Exp}_{\mathbb{G}_1}$ 1 $\text{Mul}_{\mathbb{G}_1}$	1 Hash (H_0) 1 $\text{Exp}_{\mathbb{G}_1}$	1 Hash (H_0) 1 $\text{Exp}_{\mathbb{G}_1}$ l $\text{Mul}_{\mathbb{G}_1}$	1 Hash (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-l) $\text{Mul}_{\mathbb{G}_1}$
Verification	For $apk^{(c)}$: n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ For verification: 1 Hash (H_0) l Hash (H_2) $(l-1)$ $\text{Mul}_{\mathbb{G}_1}$ 1 $\text{Mul}_{\mathbb{G}_T}$ 3 Pairings	For proof: n Hash (H_3) n $\text{Exp}_{\mathbb{G}_2}$ $2n$ pairings For $apk^{(c)}$: 1 Hash (H_4) $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ For verification: 1 Hash (H_0) l Hash (H_2) $(l-1)$ $\text{Mul}_{\mathbb{G}_1}$ 1 $\text{Mul}_{\mathbb{G}_T}$ 3 Pairings	1 Hash (H_0) $(l-1)$ $\text{Mul}_{\mathbb{G}_2}$ 2 Pairings	For $apk/spk^{(a)}$: n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ For verification: 1 Hash H_0 l Hash (H_2) l $\text{Mul}_{\mathbb{G}_1}$ 2 Pairings	For $apk/spk^{(a)}$: n Hash (H_1) n $\text{Exp}_{\mathbb{G}_2}$ $(n-1)$ $\text{Mul}_{\mathbb{G}_2}$ For verification: 1 Hash (H_0) l Hash (H_2) l $\text{Mul}_{\mathbb{G}_1}$ 2 Pairings
Transmission	For group setup: $(n-1)$ $\mathbb{G}_1$ Elt. For signature: 1 $\mathbb{G}_1$ Elt. 1 $\mathbb{G}_2$ Elt.	For group setup: $(n-1)$ $\mathbb{G}_1$ Elt. For signature: 1 $\mathbb{G}_1$ Elt. 1 $\mathbb{G}_2$ Elt.	For signature: 1 $\mathbb{G}_1$ Elt.	For signature: 1 $\mathbb{G}_1$ Elt.	For group setup: $(n-1)$ $\mathbb{G}_1$ Elt. For signature: 1 $\mathbb{G}_1$ Elt.
Broadcasting	For pk: 1 $\mathbb{G}_2$ Elt.	For pk: 1 $\mathbb{G}_2$ Elt. For Proof: 1 $\mathbb{G}_1$ Elt.	For pk: 1 $\mathbb{G}_2$ Elt. For mpk: 1 $\mathbb{G}_2$ Elt. For COM_i: n $\mathbb{G}_2$ Elt. For $\text{COM}^{(b)}$: n $\mathbb{G}_2$ Elt.	For pk: 1 $\mathbb{G}_2$ Elt.	For pk: 1 $\mathbb{G}_2$ Elt.
Storage	For sk: 1 integer ($\mathcal{O}(\lambda)$-bit) For mk: 1 $\mathbb{G}_1$ Elt.	For sk: 1 integer ($\mathcal{O}(\lambda)$-bit) For mk: 1 $\mathbb{G}_1$ Elt.	For mk: 1 integer ($\mathcal{O}(\lambda)$-bit)	For sk: 1 integer ($\mathcal{O}(\lambda)$-bit) For mk: n $\mathbb{G}_1$ Elt.	For sk: 1 integer ($\mathcal{O}(\lambda)$-bit) For mk: 1 $\mathbb{G}_1$ Elt. Combiner: (n^2-n) $\mathbb{G}_1$ Elt.

[a] As apk contains the components of spk, no extra cost is needed for computing spk.

[b] Since all the users compute the same commitment set COM, it is enough to be broadcast by only one user.

[c] apk is computed by the verifier. But to avoid the extra cost, it would be given to the combiner.

$\text{Exp}_{\mathbb{G}_i}$ Exponentiation in group $\mathbb{G}_i$ for $i \in \{1,2\}$.

$\text{Mul}_{\mathbb{G}_i}$ Multiplication in group $\mathbb{G}_i$ for $i \in \{1,2,T\}$.

$\mathbb{G}_i$ Elt. Group elements are written as $\mathbb{G}_i$ Elt. for $i \in \{1,2\}$.

H_1, H_2, H_3, H_4, H_5 Hash functions are as in the schemes.

λ Security parameter.

on Computer and Communications Security, CCS '08, page 449–458, New York, NY, USA, 2008. Association for Computing Machinery.

[2] Mihir Bellare and Gregory 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, page 390–399, New York, NY, USA, 2006. Association for Computing Machinery.

[3] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. pages 31–46, 01 2003.

[4] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 435–464, Cham, 2018. Springer International Publishing.

[5] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptol.*, 17(4):297–319, September 2004.

[7] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 465–480, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[8] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013.

[9] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 410–424, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[10] David Chaum and Eugène 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*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[11] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 307–315, New York, NY, 1990. Springer New York.

[12] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (SFCS 1987)*, pages 427–438. IEEE, 1987.

[13] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, pages 181–200, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[14] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 354–371, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

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

[16] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983.

[17] Joseph K. Liu and Duncan S. Wong. Linkable ring signatures: Security models and new schemes. In Osvaldo Gervasi, Marina L. Gavrilova, Vipin Kumar, Antonio Laganà, Heow Pueh Lee, Youngsong Mun, David Taniar, and Chih Jeng Kenneth Tan, editors, *Computational Science and Its Applications – ICCSA 2005*, pages 614–623, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multisignatures with applications to Bitcoin. *Designs, Codes and Cryptography*, 87, 09 2019.

[19] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, page 245–254, New York, NY, USA, 2001. Association for Computing Machinery.

[20] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[21] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, USA, 2016.

[22] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 139–148, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

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

[24] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, pages 252–265, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[25] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 387–398, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[26] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. In *Theoretical Computer Science*, pages 164–186. Springer, 2006.

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

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