Privacy-Preserving Aggregate-Signatures: Generic Constructions and Practical Instantiations

Xiaoyang Wei, Shuai Han, and Shengli Liu

Shanghai Jiao Tong University, Shanghai 200240, China
{weixy02,dalen17,slliu}@sjtu.edu.cn

Abstract. Aggregate signatures allow a set of signers to compress individual signatures on distinct messages into a short signature, offering significant savings in storage and verification time. However, existing aggregate signatures neither support key aggregation nor achieve strong privacy guarantees for signers. In a very recent work, Nick, Ruffing and Seurin (EUROCRYPT'26) proposed DahLIAS, a pairing-free aggregate signature scheme with constant size signatures. Unfortunately, DahLIAS fails to provide aggregated verification and privacy properties. As a side contribution, they also constructed a generic transformation from multi-signatures to aggregate-signatures. However, the transformed schemes cannot satisfy unrestrictedness and privacy.

In this paper, we formally introduce the notion of aggregate signatures with verifiable key aggregation (ASvKA), along with new unforgeability and privacy definitions. We then present a generic transformation that turns any multi-signature (MS) scheme into aggregate signature scheme with verifiable key aggregation and privacy properties, which also lift weaker unforgeability of the underlying MS to stronger unforgeability of ASvKA. Finally, we instantiate our transformation with two concrete multi-signature schemes. For pairing-free schemes, we propose PP-SpeedyASvKA, a two-round privacy-preserving aggregate signature derived from the multi-signature SpeedyMuSig, achieving the strongest unforgeability and privacy while preserving the efficiency. For pairing-based schemes, we construct PP-BAS-0 and PP-BAS-1 from a BLS multi-signature, offering different trade-offs between unforgeability and privacy.

1 Introduction

An aggregate-signature (AS) scheme [7] allows n signers with key pairs $(pk_i, sk_i)_{i\in[n]}$ to sign n messages $(m_i)_{i\in[n]}$ simultaneously, and yields a short aggregate signature σ. Then anyone with the public key/message pairs $\{(pk_i, m_i)\}_{i\in[n]}$ can verify the validity of σ. AS is different and more flexible than other multi-party signature schemes like multi-signature [4, 21, 23, 28, 1] or threshold-signature [16, 28, 14, 11] schemes, which requires n signers to sign a same message. Since it saves the storage for signatures, AS can be used to reduce bandwidth and size of transactions in blockchain systems, obtaining increasing attention in recent years. There were many AS schemes proposed in the literature [7, 3, 17, 20, 19, 5, 9, 10, 12, 26, 24], some of which are based on pairing groups

[7, 5, 3, 26], while others are pairing-free [20, 3, 19, 9, 10, 12, 24]. However, many existing AS schemes [7, 3, 17, 10, 24] suffer from two drawbacks.

Lack of Key Aggregation & Privacy in Aggregate-Signatures. Most existing AS schemes require the verification of aggregate signature σ to input the whole list of public keys $(pk_i)_{i\in[n]}$. However, this has two potential limitations.

For efficiency, it is desirable to support *key aggregation*, i.e., aggregating $(pk_i)_{i\in[n]}$ into a single aggregated public key apk, which suffices for the signature verification. Although the notion of key aggregation has been extensively studied for multi-signature schemes (MS) [21, 23, 28, 1], as far as we know, there is only one existing AS scheme in a recent work [24, App. A] supports key aggregation.

For security, as noted by Lehmann and Özbay [18] in the realm of MS, the aggregated public key apk might leak information about the amount or identity of signers, resulting in privacy issues. They defined *privacy* notions for MS schemes, and observed that no MS with deterministic key aggregation can satisfy privacy if the adversary is able to obtain all public keys.[1] The privacy issues also arise in AS schemes. The only existing AS scheme supporting key aggregation [24] has deterministic key aggregation, so it cannot guarantee privacy of signers.

Restrictedness of Aggregate-Signatures. Some AS schemes [7, 20, 24, App. A] are restricted to producing aggregate signatures for a list $\{(pk_i, m_i)\}_{i\in[n]}$ without duplicate public keys and/or messages (i.e., the public keys $(pk_i)_{i\in[n]}$ and/or the messages $(m_i)_{i\in[n]}$ must be distinct). This restriction is usually used to avoid some trivial attacks (like rouge-key attacks [7, 24]) to guarantee their security. To solve this problem, Bellare et al. [3] proposed the notion called *unrestrictedness*, which supports securely aggregate-sign any list (regardless of public key or message repetitions). As pointed out in [24], the property of unrestrictedness is important for applications to cryptocurrencies, as transactions may involve multiple coins spent by a same user (and thus require multiple messages signed under a same public key).

Based on the above discussions, we address the following question:

Can we construct aggregate-signature schemes supporting key aggregation, with strong privacy, security and unrestrictedness?

1.1 Related Works

Below we review related works on AS schemes according to whether it is pairing-based, pairing-free, or obtained via generic transformations.

Pairing-Based Aggregate-Signatures. Boneh et al. [7] first introduced the concept of aggregate-signatures and proposed schemes AS-1, AS-2 based on the pairing-based BLS scheme [8]. A potential drawback of these schemes is that the messages signed by different signers are required to be distinct. To solve this problem, Bellare et al. [3] further proposed schemes AS-3, AS-4 with unrestrictedness. Recently, Sakai [26] proposed NIZK-AS, a pairing-based aggregate-signature

[1] Since if the key aggregation algorithm is deterministic, the adversary can invoke the algorithm itself to decide whether apk is an aggregation of $(pk_i)_{i\in[n]}$ or not.

with tight security based on non-interactive zero-knowledge (NIZK) proofs, and Hohenberger et al. [15] proposed AS-WRO, a pairing-based aggregate-signature without random oracles.

Pairing-Free Aggregate-Signatures. Chalkias et al. [10] applied a "half-aggregation" technique for non-interactive Schnorr-based aggregate signatures and presented ASchnorr, where a set of Schnorr signatures can be compressed to half their original size. Chen and Zhao [12] provided a tight reduction of ASchnorr in the algebraic group model (AGM). Recently, Nick, Ruffing and Seurin [24] proposed DahLIAS, a two-round aggregate signature over pairing-free groups.

Generic Transformations. In [24, App. A], Nick, Ruffing and Seurin also proposed a transformation from multi-signatures (MS) to aggregate-signatures (AS). Their transformation yields the only AS scheme supporting key aggregation. However, the resulting AS scheme is restricted, i.e., the distinctness of public keys should be checked during signature verification, which will be less efficient and convenient in practical applications.

Recently, Lehmann et al. [18, 1] proposed a transformation for MS schemes to improve the security and add privacy of MS. However, as far as we know, no existing works have yet studied the privacy of AS schemes.

1.2 Our Contributions

In this work, we introduce the notion of aggregate signature schemes with verifiable key aggregation (ASvKA), and propose a transformation for ASvKA schemes with strong privacy, unforgeability and unrestrictedness.

New Definition: Aggregate-Signatures with Verifiable Key Aggregation. To enable efficient aggregated verification while ensuring privacy, we formalize aggregate signature with verifiable key aggregation (ASvKA), and define three levels of unforgeability AS-UNF-1/2/3 and privacy AS-FullPriv for it.

On one hand, multiple public keys $(pk_i)_{i\in[n]}$ from different signers can be aggregated into a single apk in a randomized and verifiable way, with a proof of aggregation π. With π, one can verify the correctness of key aggregation; without π, one can hardly tell whether apk is an aggregated key (captured by the AS-FullPriv security), thus leaking no information about the signing groups.

On the other hand, by using the aggregated public key apk, one can efficiently verify the validity of aggregate signatures σ, without the need of inputting all public keys $(pk_i)_{i\in[n]}$.[2] We refer to Sect. 3 for more details.

New Transformation: ASvKA from MS. We present a generic transformation of ASvKA from multi-signature schemes MS. The resulting ASvKA can achieve stronger unforgeability (like AS-UNF-2/3) even if the underlying MS satisfies only the weakest MS-UNF-1.

[2] In fact, in our formalization of ASvKA (cf. Sect. 3), the messages $(m_i)_{i\in[n]}$ can also be combined into a single message m, so that the verification of aggregate signatures σ only needs m (and apk), improving the efficiency further.

Moreover, our ASvKA achieves unrestrictedness, i.e., the signers can produce aggregated-signatures for any set $\{(pk_i, m_i)\}_{i\in[n]}$ (regardless of public key or message repetitions). Besides, the aggregated signing process of our ASvKA is more efficient in case of public key repetitions. For example, suppose that signer 1 with public key pk_1 wants to sign three messages m_1, m_2, m_3 (i.e., the set contains $(pk_1, m_1), (pk_1, m_2), (pk_1, m_3)$ with duplicated pk_1), it can invoke the aggregate signing algorithms only once, instead of three times.

New Instantiations: Non-Interactive Pairing-Based & Two-Round Pairing-Free Schemes. Given the recent rapid progress on MS schemes [6, 21, 23, 13, 2, 28, 1], we can get many ASvKA schemes via our transformation. Concretely, by instantiating our transformation from a BLS multi-signature scheme MS-BLS [6] and a Schnorr multi-signature scheme SpeedyMuSig [13, 2] respectively, we obtain two non-interactive pairing-based schemes PP-BAS-0, PP-BAS-1 and a two-round pairing-free scheme PP-SpeedyASvKA. In particular, our instantiations PP-BAS-1 and PP-SpeedyASvKA achieve the strongest AS-UNF-3 unforgeability, AS-FullPriv privacy and unrestrictedness, simultaneously.

We show a comparison with existing aggregate-signature schemes in Table 1.

Table 1. Comparison of non-interactive and interactive aggregate-signature schemes. The column **Round** shows the total number of signing rounds. The column **Sig. Size** shows the size of aggregate signatures, where n is the number of signers and $|\mathbb{G}|/|\mathbb{G}_1|$ denote the size of cyclic group/pairing group. The column **Unforg.** shows the unforgeability level of the scheme. The column **Key Agg.** shows whether the scheme provides public key aggregation. The column **Pairing-Free** shows whether the scheme is pairing-free or pairing-based. The column **Privacy** shows whether the scheme provides privacy guarantees. The column **Unrestrict.** shows whether the scheme achieves unrestrictedness (without requiring the distinctness of public keys and messages). The column **Agg. Ver.** shows whether the scheme supports aggregated verification. * Our PP-BAS-1 achieves AS-FullPriv against a key-prefixed variant of BLS (cf. Subsect. 5.1).

Scheme	Round	Sig. Size	Unforg.	Key Agg.	Pairing-Free	Privacy	Unrestrict.	Agg. Ver.
Non-Interactive Schemes								
AS-1, AS-2 [7]	1	$\|\mathbb{G}_1\|$	AS-UNF-1	×	×	×	×	×
AS-3 [3]	1	$\|\mathbb{G}_1\|$	AS-UNF-1	×	×	×	✓	×
AS-4 [3]	1	$\|\mathbb{G}_1\| + n$	AS-UNF-1	×	×	×	✓	×
ASchnorr [10, 12]	1	$n\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	AS-UNF-1	×	✓	×	✓	×
NIZK-AS [26]	1	$2\|\mathbb{G}_1\| + n$	AS-UNF-1	×	×	×	✓	×
AS-WRO [15]	1	$2\|\mathbb{G}_1\|$	AS-UNF-1	×	×	×	✓	×
PP-BAS-0 (Subsect. 5.1)	1	$\|\mathbb{G}_1\|$	AS-UNF-2	✓	×	AS-FullPriv	✓	✓
PP-BAS-1 (Subsect. 5.1)	1	$\|\mathbb{G}_1\|$	AS-UNF-3	✓	×	AS-FullPriv*	✓	✓
Interactive Schemes								
BN-IAS(App. A) [24, 4]	3	$\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	AS-UNF-2	×	✓	×	×	×
MuSig2-IAS(App. A) [24, 23]	2	$\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	AS-UNF-2	✓	✓	×	×	✓
DahLIAS [24]	2	$\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	AS-UNF-2	×	✓	×	✓	×
PP-SpeedyASvKA (Subsect. 5.2)	2	$\|\mathbb{G}\| + \|\mathbb{Z}_p\|$	AS-UNF-3	✓	✓	AS-FullPriv	✓	✓

2 Preliminaries

Notations. Let κ denote the security parameter, and all algorithms and adversaries take 1^κ as an implicit input. If x is defined by y or the value of y is assigned to x, we write $x := y$. For $i < j \in \mathbb{N}$, define $[i, j] := \{i, i+1, ..., j\}$ and $[j] := \{1, 2, ..., j\}$. For a set $\mathcal{X}$, denote by $x \leftarrow_\$ \mathcal{X}$ the procedure of sampling x from $\mathcal{X}$ uniformly at random. If $\mathcal{D}$ is distribution, $x \leftarrow_\$ \mathcal{D}$ means that x is sampled according to $\mathcal{D}$. If $\mathcal{A}$ is a randomized algorithm, we let $y := \mathcal{A}(x; \rho)$ denote the operation of running $\mathcal{A}$ on inputs of x and random coins ρ and assigning its output to y, and $y \leftarrow \mathcal{A}(x)$ when coins ρ are chosen uniformly at random. "PPT" abbreviates probabilistic polynomial-time. Denote by negl some negligible function. We use "Assert P" in algorithms to denote a check of statement P: if P is false, the algorithm returns 0; otherwise, the algorithm continues.

We present the definitions of linear hash function and multi-signatures (MS) in Appendix A, which will serve as the building blocks of our generic transformation. We also recall the security assumptions, the standard BLS [8] and Schnorr [27] signature schemes, and the existing syntax of aggregate-signatures (AS) in Appendix B for completeness.

3 Aggregate-Signatures with Verifiable Key Aggregation

As motivated in the introduction, the existing syntax of AS cannot guarantee privacy if the adversary knows all signers' public keys. Therefore, inspired by a privacy-preserving technique used in multi-signature schemes [18, 1], we propose a new syntax for aggregate-signatures with verifiable key aggregation (ASvKA), and formalize its unforgeability and privacy notions accordingly.

3.1 Syntax and Correctness

Compared with the existing AS syntax, we make the following modifications.

- **Proofs of Possession.** To prevent rogue-key attacks, during key generation KGen, each signer generates its key pair (pk, sk), where $pk = (X, pop)$ contains a standard public key component X and its proof of possession pop, and then the validity of each signer can be verified via the KVf algorithm.
- **Verifiable Key Aggregation.** We propose a verifiable key aggregation algorithm KAg, which aggregates a set of public keys PK into an aggregated key apk along with a proof of aggregation π. Accordingly, one can check whether apk is an aggregation of PK via the KAgVf algorithm by using the proof π, where π is only shared by inside signers (to make privacy possible).
- **Message Combination.** We also define a message combination algorithm MsgCom, which takes a set of public keys and messages $L = \{(X_i, m_i)\}_{i \in [n]}$ as input, and outputs a combined message m.

Definition 1 (ASvKA). *An aggregate-signature scheme with verifiable key aggregation (ASvKA) consists of a tuple of PPT algorithms* $\mathsf{ASvKA} = (\mathsf{KGen}, \mathsf{KVf}, \mathsf{KAg}, \mathsf{KAgVf}, \mathsf{MsgCom}, \mathsf{AggSign}, \mathsf{Combine}, \mathsf{Vrfy})$ *s.t.:*

- $\underline{(pk = (X, pop), sk) \leftarrow \mathsf{KGen}}$: *The key generation algorithm generates a pair of public key and secret key* (pk, sk)*, where* pk *contains a public key component* X *and its proof of possession* pop*.*
- $\underline{0/1 \leftarrow \mathsf{KVf}(pk = (X, pop))}$: *The public key verification algorithm takes a public key* $pk = (X, pop)$ *as input, and outputs a bit indicating whether* pk *is valid or not. (If the verification succeeds, the public key component* X *will be added into a set* PK *for legal signers.)*
- $\underline{(apk, \pi) \leftarrow \mathsf{KAg}(PK)}$: *The key aggregation algorithm takes a set of public keys* $PK = (X_i)_{i\in[n]}$ *as input, and outputs an aggregated public key* apk *along with a proof of aggregation* π*.*
- $\underline{0/1 \leftarrow \mathsf{KAgVf}(PK, apk, \pi)}$: *The apk verification algorithm takes a set* $PK = (X_i)_{i\in[n]}$ *of public keys, an aggregated public key* apk *and a proof* π *as input, and outputs a bit indicating whether* apk *is a valid aggregation of* PK*.*
- $\underline{m \leftarrow \mathsf{MsgCom}(L)}$: *The message combination algorithm takes a set* $L = \{(X_i, m_i)\}_{i\in[n]}$ *of public keys and messages as input, and outputs a combined message* m*.*
- $\underline{z_i \leftarrow \mathsf{AggSign}(sk_i, L, apk, \pi)}$: *For the* i*-th signer, on input the secret key* sk_i*, a set* $L = \{(X_i, m_i)\}_{i\in[n]}$ *of public keys and messages, an aggregated key* apk *and a proof of aggregation* π*, the signing algorithm outputs a signature share* z_i*. If* $\mathsf{AggSign}$ *is an interactive protocol with multiple rounds (say* R *rounds), we denote sub-algorithms* $\mathsf{AggSign}_j$ *(*$j \in R$*) as follows:*

$$
\begin{aligned}
&(\mathsf{st}_{i,1}, z_{i,1}) \leftarrow \mathsf{AggSign}_1(sk_i, L, apk, \pi)\\
&(\mathsf{st}_{i,j}, z_{i,j}) \leftarrow \mathsf{AggSign}_j(\mathsf{st}_{i,j-1}, \mathsf{inp}_{i,j-1}),\ j \in [2, R-1]\\
&z_i \leftarrow \mathsf{AggSign}_R(\mathsf{st}_{i,R-1}, \mathsf{inp}_{i,R-1})
\end{aligned}
$$

  where $\mathsf{st}_{i,j}$ *denotes the state of the signer* i *after the* j*-th signing round,* $z_{i,j}$ *denotes the output of the signer* i *in round* j*, and* $\mathsf{inp}_{i,j-1} = \{z_{k,j-1}\}_k$ *is the set of outputs for round* $j-1$ *from all signers* k*.*
- $\underline{\sigma \leftarrow \mathsf{Combine}(L, \{z_i\}, [\pi])}$: *The signature combining algorithm takes a set* $L = \{(X_i, m_i)\}_{i\in[n]}$ *of public keys and messages and a set of signature shares* $\{z_i\}$ *from all signers as input, and outputs a combined signature* σ*. Here* $[\pi]$ *denotes that the proof of aggregation* π *is an optional input, which is required for the generic transformation (in Sect. 4), but not needed for the instantiations (in Sect. 5).*
- $\underline{0/1 \leftarrow \mathsf{Vrfy}(L, apk, \sigma)}$: *The signature verification algorithm takes a set* $L = \{(X_i, m_i)\}_{i\in[n]}$ *of public keys and messages, an aggregated key* apk *and a signature* σ *as input, and outputs a bit indicating whether* σ *is valid signature.*

Correctness. *For any number of signers* n*, any messages* $m_1, ..., m_n$*, any* $(pk_i = (X_i, pop_i), sk_i) \leftarrow \mathsf{KGen}$ *with* $i \in [n]$*, let* $L := \{(X_i, m_i)\}_{i\in[n]}$ *and* $PK := (X_i)_{i\in[n]}$*, for any* $(apk, \pi) \leftarrow \mathsf{KAg}(PK)$*,* $z_i \leftarrow \mathsf{AggSign}(sk_i, L, apk, \pi)$ *with* $i \in [n]$*, and any* $\sigma \leftarrow \mathsf{Combine}(L, \{z_i\})$*, it holds that* $\mathsf{KVf}(pk_i = (X_i, pop_i)) = 1$ *for all* $i \in [n]$*,* $\mathsf{KAgVf}(PK, apk, \pi) = 1$ *and* $\mathsf{Vrfy}(L, apk, \sigma) = 1$*.*

Definition 2 (Aggregated Verification for ASvKA). *We say that an* ASvKA *scheme supports "aggregated verification", if there exists an additional PPT signature verification algorithm* $\mathsf{AggVrfy}$*:*

– $0/1 \leftarrow \mathsf{AggVrfy}(m, apk, \sigma)$: *The aggregated verification algorithm takes a combined message m (instead of a set L of public keys and messages), an aggregated key apk and a signature σ as input, and outputs a bit indicating whether σ is a valid signature.*

Correctness further requires $\mathsf{AggVrfy}(m, apk, \sigma) = 1$ *for* $m \leftarrow \mathsf{MsgCom}(L)$.

3.2 Unforgeability Definitions

We propose different levels of unforgeability and freshness definitions for ASvKA.

$\underline{\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{ASvKA},\mathcal{A}}}$:
$(pk^* = (X^*, pop^*), sk^*) \leftarrow \mathsf{KGen}$
$LPK := \{X^*\}$ //record valid key components
$Q, Q_1, ..., Q_R := \emptyset$
$\mathcal{O} := (\mathcal{O}_{\mathrm{REG}}, \mathcal{O}_{\mathrm{AGGSIGN}\,1}, ..., \mathcal{O}_{\mathrm{AGGSIGN}\,R})$
$(\sigma^*, L^*, \pi^*, apk^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pk^*)$
Parse $L^* = \{(X_i^*, m_i^*)\}_{i\in[n]}$
$PK^* := (X_i^*)_{i\in[n]}$
Return 1 if $PK^* \subseteq LPK \wedge$
$\mathsf{KAgVf}(PK^*, apk^*, \pi^*) = 1 \wedge$
$\mathsf{Vrfy}(L^*, apk^*, \sigma^*) = 1 \wedge$
$\exists\, i \in [n],$ s.t. $X_i^* = X^* \wedge\ \mathsf{fresh}(m_i^*, L^*, apk^*, Q) = 1$
//fresh is defined in Fig. 2

$\underline{\mathcal{O}_{\mathrm{REG}}(pk = (X, pop))}$:
If $\mathsf{KVf}(X, pop) = 1$: $LPK := LPK \cup \{X\}$, Return 1
Else: Return 0

$\underline{\mathcal{O}_{\mathrm{AGGSIGN}\,1}(k, L, apk, \pi)}$:
//k denotes the index of the signing query
Parse $L = \{(X_i, m_i)\}_{i\in[n]}$
$PK := (X_i)_{i\in[n]}$
Assert $(k, \cdots) \notin Q_1 \wedge X^* \in PK\ \wedge PK \subseteq LPK$
$\wedge\ \mathsf{KAgVf}(PK, apk, \pi) = 1$
For all $i \in [n]$ s.t. $X_i = X^*$:
$Q_1 := Q_1 \cup \{(k, m_i, L, apk)\}$
$(\mathsf{st}_{k,1}, z_{k,1}) \leftarrow \mathsf{AggSign}_1(sk^*, L, apk, \pi)$
Return $z_{k,1}$

$\underline{\mathcal{O}_{\mathrm{AGGSIGN}\,j}(k, \mathsf{inp}_{k,j-1})}$://$j \in [2, R-1]$
Assert $k \notin Q_j \wedge k \in Q_1, ..., Q_{j-1}$
$Q_j := Q_j\ \cup \{k\}$
$(\mathsf{st}_{k,j}, z_{k,j}) \leftarrow \mathsf{AggSign}_j(\mathsf{st}_{k,j-1}, \mathsf{inp}_{k,j-1})$
Return $z_{k,j}$

$\underline{\mathcal{O}_{\mathrm{AGGSIGN}\,R}(k, \mathsf{inp}_{k,R-1})}$:
Assert $k \notin Q_R \wedge k \in Q_1, ..., Q_{R-1}$
$Q_R := Q_R \cup \{k\}$
$z_{k,R} \leftarrow \mathsf{AggSign}_R(\mathsf{st}_{k,R-1}, \mathsf{inp}_{k,R-1})$
For all $(k, m_i, L, apk) \in Q_1$:
$Q := Q \cup \{(m_i, L, apk)\}$
Return $z_{k,R}$

Fig. 1. The unforgeability experiment $\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{ASvKA},\mathcal{A}}$ for ASvKA schemes with multiple signing rounds, where fresh is defined in Fig. 2.

AS-UNF-X	AS-UNF-1	AS-UNF-2	AS-UNF-3
$\mathsf{fresh}(m_i^*, L^*, apk^*, Q) = 1$ if	$(m_i^*, \cdot, \cdot) \notin Q$	$(m_i^*, L^*, \cdot) \notin Q$	$(m_i^*, L^*, apk^*) \notin Q$

Fig. 2. Definitions of fresh for different levels of unforgeability in ASvKA.

Definition 3 (Unforgeability for ASvKA). *An* ASvKA *scheme is* AS-UNF-X *secure, if for any PPT adversary $\mathcal{A}$, it holds that* $\mathsf{Adv}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{ASvKA},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{ASvKA},\mathcal{A}} \Rightarrow 1] \leq \mathsf{negl}(\kappa)$, *where the experiment* $\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{ASvKA},\mathcal{A}}$ *is defined in Fig. 1.*

The adversary $\mathcal{A}$ gets a public key $pk^* = (X^*, pop^*)$ of an honest signer, and has access to oracle $\mathcal{O}_{\mathrm{REG}}$ for registering new public keys, as well as oracles $\mathcal{O}_{\mathrm{AGGSIGN}\,1}, ..., \mathcal{O}_{\mathrm{AGGSIGN}\,R}$ for R signing rounds, which compute and output the honest signer's signature shares $z_{k,j}$. Finally, $\mathcal{A}$ forges a signature σ^* w.r.t. a set $L^* = \{(X_i^*, m_i^*)\}_{i\in[n]}$ of public keys and messages – now along with an aggregated key apk^* and a proof of aggregation π^*. The forgery is valid if (i) all $(X_i^*)_{i\in[n]}$ are valid public keys (i.e., have been registered in $\mathcal{O}_{\mathrm{REG}}$), (ii) apk^*

is a valid aggregation of $(X_i^*)_{i\in[n]}$ with proof π^*, (iii) σ^* is a valid aggregate signature of L^*, and moreover, (iv) the honest user's public key X^* should appear in $(X_i^*)_{i\in[n]}$ (i.e., $\exists i \in [n]$, s.t. $X_i^* = X^*$) and the corresponding m_i^* should not be signed via the $\mathcal{O}_{\text{AGGSIGN}}$ oracles (as defined by the fresh).

We provide three levels of unforgeability AS-UNF-1/2/3, each corresponds to a distinct freshness check on the forgery. In particular, AS-UNF-3 is the strongest one, as it ensures that $\mathcal{A}$ can hardly output a valid signature σ^* for $L^* = \{(X_i^*, m_i^*)\}_{i\in[n]}$ and apk^*, even if $\mathcal{A}$ has queried $\mathcal{O}_{\text{AGGSIGN}1}, ..., \mathcal{O}_{\text{AGGSIGN}R}$ oracles on m_i^* w.r.t. the same set L^*, as long as the aggregated key apk is different.

3.3 Privacy Definitions

Lehmann and Özbay [18] proposed three levels of privacy for multi-signature MS schemes, where full privacy is the strongest one.

- **Full Privacy:** One cannot distinguish an aggregate public key and a multi-signature from a public key and a signature of an ordinary signature scheme.
- **Set Privacy:** One cannot distinguish which signing group an aggregated public key and a multi-signature are derived from.
- **Membership Privacy:** The aggregated public key and multi-signature do not leak information about individual signers (but might leak the size of the signing group).

Inspired by the above privacy properties, we propose AS-FullPriv, the strongest privacy-preserving property for ASvKA, which ensures that one cannot distinguish an aggregated key of a group of signers from a public key of a single signer, and cannot distinguish an aggregate signature generated by a signing group from an ordinary signature generated by a single signer. It implies that there is no leakage of the information about signing groups or individual signers.

$\underline{\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(b)}$: // $b \in \{0,1\}$	$\underline{\mathcal{O}_{\text{CHL}}(\{m_i\}_{i\in S^*})}$:
For $i \in [n]$: $(pk_i = (X_i, pop_i), sk_i) \leftarrow \mathsf{KGen}$	$L_{S^*} := \{(X_i, m_i)\}_{i\in S^*}$
$S^* \leftarrow \mathcal{A}(\{pk_i, sk_i\}_{i\in[n]})$	If $b = 0$:
$PK_{S^*} := (X_i)_{i\in S^*}$	For $i \in S^*$:
If $b = 0$: $(apk, \pi) \leftarrow \mathsf{KAg}(PK_{S^*})$, $pk^* := apk$	$z_i \leftarrow \mathsf{AggSign}(sk_i, L_{S^*}, apk, \pi)$
If $b = 1$: $(pk, sk) \leftarrow \mathsf{KGen}$, $pk^* := pk$	Return $\sigma \leftarrow \mathsf{Combine}(L_{S^*}, \{z_i\}_{i\in S^*})$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\text{CHL}}(\cdot)}(pk^*)$	If $b = 1$:
Return b^*	$m \leftarrow \mathsf{MsgCom}(L_{S^*})$
	Return $\sigma \leftarrow \mathsf{Sign}(sk, m)$

Fig. 3. The AS-FullPriv experiment $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(b)$ for ASvKA schemes.

Definition 4 (Full Privacy for ASvKA). *An* ASvKA *scheme has full privacy (*AS-FullPriv*), if there exists a PPT signing algorithm* Sign *(which takes as input a signer's secret key sk and a message m, and outputs a signature σ), such that for any PPT adversary $\mathcal{A}$ and any polynomial number n of users, it holds that*

$$\mathsf{Adv}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(\kappa) := \left|\Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(0) \Rightarrow 1] - \Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(1) \Rightarrow 1]\right| \leq \mathsf{negl}(\kappa),$$

where the experiment $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(b)$ *(*$b \in \{0,1\}$*) is defined in Fig. 3.*

The adversary $\mathcal{A}$ can get the public/secret keys $\{pk_i = (X_i, pop_i), sk_i\}_{i\in[n]}$ of all users, and designates a signing group S^*. Then $\mathcal{A}$ receives either an aggregated public key $pk^* = apk$ of $PK_{S^*} = (X_i)_{i\in S^*}$ via KAg (if $b = 0$) or an independently generated public key $pk^* = pk$ via KGen for an individual signer (if $b = 1$).

Moreover, $\mathcal{A}$ has access to a challenge oracle $\mathcal{O}_{\mathrm{CHL}}$ by querying a set of messages $\{m_i\}_{i\in S^*}$. The oracle $\mathcal{O}_{\mathrm{CHL}}$ either returns an aggregate signature of $\{(X_i, m_i)\}_{i\in S^*}$ (if $b = 0$) or a standard signature of the combined message m (if $b = 1$). Our $\mathsf{AS\text{-}FullPriv}$ ensures that $\mathcal{A}$ can hardly distinguish $b = 0$ from $b = 1$.

4 Generic Transformation from MS to ASvKA

In this section, we propose a generic transformation from MS schemes (that meet certain conditions) to ASvKA schemes. We note that our transformation not only transforms multi-signatures (which allows multiple parties to sign a *same* message) to aggregate-signatures (which allows multiple parties to sign *different* messages), but also improves the level of unforgeability.

Generic Transformation. Let $\mathsf{MS} = (\mathsf{MS.Setup}, \mathsf{MS.KGen}, \mathsf{MS.MulSign}, \mathsf{MS.Combine}, \mathsf{MS.Vrfy})$ be a R-round multi-signature scheme (cf. Appendix A). Our construction of $\mathsf{ASvKA} = (\mathsf{ASvKA.Setup}, \mathsf{ASvKA.KGen}, \mathsf{ASvKA.KVf}, \mathsf{ASvKA.KAg}, \mathsf{ASvKA.KAgVf}, \mathsf{ASvKA.MsgCom}, \mathsf{ASvKA.AggSign}, \mathsf{ASvKA.Combine}, \mathsf{ASvKA.Vrfy})$ is presented in Fig. 4, which has also R rounds.

More precisely, we present two slightly different versions of our transformation in Fig. 4: Construction 1 (with framed boxes) and Construction 2 (with gray boxes), which are only slightly different in the $\mathsf{ASvKA.AggSign}_1$ and $\mathsf{ASvKA.Vrfy}$ algorithms.[3] As will be shown later, we will prove slightly different security theorems for these two constructions. In particular, our Construction 2 can lift the weakest $\mathsf{MS\text{-}UNF\text{-}1}$ schemes to the strongest $\mathsf{AS\text{-}UNF\text{-}3}$ schemes.

Below we explain the requirements on the underlying MS scheme and the specific algorithms of our ASvKA scheme in more detail.

- **Requirements on MS.** The secret key sk_{MS} and public key pk_{MS} of MS satisfy additive homomorphism, which is determined by a linear hash function F from $\mathcal{D}$ to $\mathcal{R}$ where $\mathcal{D}$ and $\mathcal{R}$ are two $\mathcal{S}$ modules (cf. Appendix A). More precisely, sk_{MS} is randomly sampled in $\mathcal{D}$, and the corresponding $pk_{\mathsf{MS}} := \mathsf{F}(sk_{\mathsf{MS}})$ is uniformly distributed over $\mathcal{R}$. We also use three hash functions $\mathsf{H}_{\mathsf{pop}} : \{0,1\}^* \to \mathcal{S}, \mathsf{H}_{\mathsf{dum}} : \{0,1\}^* \to \mathcal{D}$ and $\mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^\kappa$ in our transformation that will be modeled as random oracles (ROs).
- **Extracting Distinct Public Keys.** Our ASvKA scheme enjoys the advantage that a signer can sign multiple messages in a signing session (thus

[3] More precisely, Construction 1 invokes the underlying $\mathsf{MS.MulSign}_1$ and $\mathsf{MS.Vrfy}$ algorithms on the combined message m, while Construction 2 on (m, apk).

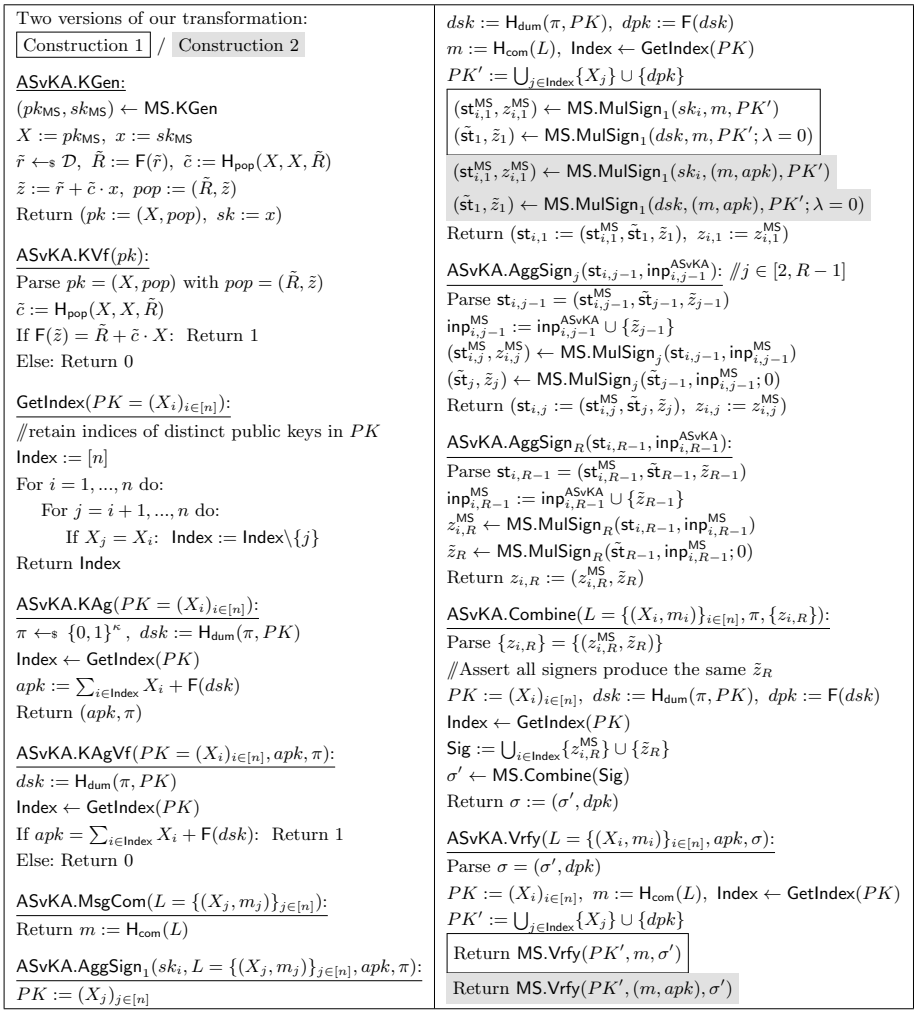

Fig. 4. Our transformation from MS to ASvKA, where $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$ are hash functions. Here we present two slightly different versions: Construction 1 (with framed boxes) and Construction 2 (with gray boxes). GetIndex is an internal algorithm invoked by the algorithms of ASvKA. The random coins $\lambda = 0$ used in $\mathsf{ASvKA.AggSign}_j(j \in [R])$ indicates that all random coins selected when executing $\mathsf{MS.MulSign}_j(j \in [R])$ algorithms are zeros.

achieving unrestrictedness), and moreover, this can be accomplished by invoking the signing algorithms $\{\mathsf{AggSign}_j\}_{j\in[R]}$ each only once by the signer.[4]

[4] For example, suppose that signer 1 (with key pair $(pk_1 = (X_1, pop_1), sk_1)$) wants to sign three messages m_1, m_2, m_3, it can invoke $\mathsf{AggSign}_1(sk_1, L, apk, \pi)$ with $L = \{(X_1, m_1), (X_1, m_2), (X_1, m_3), ...\}$ just once, and invoke the remaining algorithms $\mathsf{AggSign}_j$ $(j \in [2, R])$ only once each, rather than three times each.

To support this desirable property, we propose an auxiliary algorithm $\mathsf{GetIndex}$, which takes a set $PK = (X_i)_{i\in[n]}$ of public keys (which may contain duplicates) as input and outputs a set $\mathsf{Index} \subseteq [n]$ which retains the indices of distinct public keys X_i in PK.[5] $\mathsf{GetIndex}$ is an internal algorithm that is used in almost all the remaining algorithms of our ASvKA scheme.

- **Verifiable Key Aggregation.** To aggregate public keys $PK = (X_i)_{i\in[n]}$ while ensuring privacy, inspired by [1], our $\mathsf{ASvKA.KAg}$ algorithm first picks a random string $\pi \leftarrow\!\!\$\ \{0,1\}^\kappa$ and computes $(dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK), dpk := \mathsf{F}(dsk))$ as a fresh key pair of a *dummy signer*. Moreover, to avoid heavy exponentiations, we compute $apk := \sum_{i\in\mathsf{Index}} X_i + \mathsf{F}(dsk)$ as the aggregated key where $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$, and set π as the proof of aggregation.
- **Aggregate Signature Generation.** During the signing process of our ASvKA scheme, the dummy signer also contributes to the aggregate signature generation under its key pair (dsk, dpk). Since the dummy signer is virtual, all signers need to compute the signature share of the dummy signer (i.e., the $\tilde{z}_j$ in each $\mathsf{ASvKA.AggSign}_j$) by using $dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK)$, besides their own signature shares (i.e., the $z_{i,j}$ in each $\mathsf{ASvKA.AggSign}_j$). Moreover, we need to ensure that the dummy signer's signature shares $\tilde{z}_j$ computed by all signers are the same, and to this end, all signers use the same coin $\lambda = 0$ for generating the dummy signer's signature share.

Security Analysis. Below we prove the security of the two versions of our transformation (i.e., Construction 1 & Construction 2) via the following two theorems, which show that our transformation improves the level of unforgeability.[6]

Theorem 1 (Unforgeability of Construction 1). *For Construction 1 (Fig. 4 with framed boxes), where* $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$ *are modeled as ROs, we have:*

- *If* MS *is* $\mathsf{MS\text{-}UNF\text{-}2}$ *secure, then* ASvKA *is* $\mathsf{AS\text{-}UNF\text{-}3}$ *secure.*
- *If* MS *is* $\mathsf{MS\text{-}UNF\text{-}1}$ *secure, then* ASvKA *is* $\mathsf{AS\text{-}UNF\text{-}2}$ *secure.*

Proof Overview. Below we present a proof overview, and refer to Appendix C.1 for the formal proof. For the first part of the theorem, let $\mathcal{A}$ be an adversary against the $\mathsf{AS\text{-}UNF\text{-}3}$ security of ASvKA, then we construct an adversary $\mathcal{B}$ against the $\mathsf{MS\text{-}UNF\text{-}2}$ security of MS. $\mathcal{B}$ embeds the public key $pk^*_{\mathsf{MS}} = X^*$ of MS as the public key component of the honest signer, generates a valid proof of possession for X^* by controlling the random oracle, and simulates the $\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}3}}_{\mathsf{ASvKA},\mathcal{A}}$ experiment for $\mathcal{A}$. After receiving a forgery $(\sigma^*, L^*, \pi^*, apk^*)$ from $\mathcal{A}$ where $L^* = \{(X^*_i, m^*_i)\}_{i\in[n]}$, $\mathcal{B}$ parses $\sigma^* = (\sigma'^*, dpk^*)$, and computes

$$\begin{aligned} &PK^* := (X^*_i)_{i\in[n]}, m^* := \mathsf{H}_{\mathsf{com}}(L^*), \\ &\mathsf{Index}^* \leftarrow \mathsf{GetIndex}(PK^*), PK'^* := \textstyle\bigcup_{i\in\mathsf{Index}^*}\{X^*_i\} \cup \{dpk^*\}. \end{aligned} \tag{1}$$

Finally, $\mathcal{B}$ returns (PK'^*, m^*, σ'^*) as its own forgery for MS.

If $\mathcal{A}$'s output is a valid $\mathsf{AS\text{-}UNF\text{-}3}$ forgery, i.e.,

[5] E.g., for $PK = (X_1, X_2 = X_1, X_3 \neq X_1)$, $\mathsf{GetIndex}(PK)$ outputs $\mathsf{Index} = \{1, 3\}$.

[6] Here we focus on unforgeability and will not prove the correctness and privacy of our transformation, but will instead prove them for the instantiations in Sect. 5 later.

(i) $PK^* \subseteq LPK$, (ii) $\mathsf{ASvKA.KAgVf}(PK^*, apk^*, \pi^*) = 1$,
(iii) $\mathsf{ASvKA.Vrfy}(L^*, apk^*, \sigma^*) = 1$,
(iv) $\exists i \in [n]$, s.t. $X_i^* = X^*$ and $(m_i^*, L^*, apk^*) \notin Q$,

we need to prove that $\mathcal{B}$'s output is a valid MS-UNF-2 forgery, i.e.,

(i′) $pk_{\mathsf{MS}}^* = X^* \in PK'^*$, (ii′) $\mathsf{MS.Vrfy}(PK'^*, m^*, \sigma'^*) = 1$,
(iii′) $(m^*, PK'^*) \notin Q_{\mathsf{MS}}$, where Q_{MS} denotes the set Q in the MS-UNF-2 experiment (cf. Fig. 7). More precisely, Q_{MS} records all tuples (m, PK') that $\mathcal{B}$ has queried its own first round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m, PK')$ for some k and also queried its own R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this index k.

Clearly, (i′) follows from (iv), and (ii′) follows from (iii) by our construction of ASvKA.Vrfy, thus we have to show that (iii′) holds.

Suppose towards a contradiction that (iii′) does not hold, i.e., $(m^*, PK'^*) \in Q_{\mathsf{MS}}$. This means that $\mathcal{B}$ has queried the first-round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m^*, PK'^*)$ oracle for some index k, and this query must be triggered by the first-round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L^*, apk^*, \pi)$ query from $\mathcal{A}$ on this k. In addition, $\mathcal{B}$ has queried its R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ oracle for this k, which is triggered by the R-th round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query from $\mathcal{A}$ on this k. Consequently, according to the $\mathsf{Exp}^{\text{AS-UNF-3}}_{\mathsf{ASvKA},\mathcal{A}}$ experiment (cf. Fig. 1), during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}$ query made by $\mathcal{A}$, for all $i \in [n]$ s.t. $X_i^* = X^*$, the tuples (k, m_i^*, L^*, apk^*) will be added to Q_1, and furthermore, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query, the tuples (m_i^*, L^*, apk^*) will be added to Q, which leads to a contradiction with (iv).

In conclusion, we have that (iii′) also holds and $\mathcal{B}$ outputs a valid MS-UNF-2 forgery, which concludes the first part of the theorem.

For the second part of the theorem, we construct an adversary $\mathcal{B}$ against the MS-UNF-1 security by simulating the $\mathsf{Exp}^{\text{AS-UNF-2}}_{\mathsf{ASvKA},\mathcal{A}}$ experiment for an adversary $\mathcal{A}$ against the AS-UNF-2 security. $\mathcal{B}$ answers $\mathcal{A}$'s queries in the same way as above, except that the winning condition (iv) for $\mathcal{A}$ will be changed to

(iv) $\exists i \in [n]$, s.t. $X_i^* = X^*$ and $(m_i^*, L^*, \cdot) \notin Q$,

and the winning condition (iii′) for $\mathcal{B}$ will be changed to

(iii′) $(m^*, \cdot) \notin Q_{\mathsf{MS}}$.

Similar to the above analysis, we can show the new (iii′) based on the new (iv) condition, and conclude the second part of the theorem. □

Theorem 2 (Unforgeability of Construction 2). *For Construction 2 (Fig. 4 with gray boxes), where* $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$ *are modeled as ROs, we have:*

– *If* MS *is* MS-UNF-1 *secure, then* ASvKA *is* AS-UNF-3 *secure.*

Proof Overview. The proof is similar to the proof of the first part of Theorem 1, and we highlight the differences in gray boxes. We construct an adversary $\mathcal{B}$ against the MS-UNF-1 security of MS, by simulating the $\mathsf{Exp}^{\text{AS-UNF-3}}_{\mathsf{ASvKA},\mathcal{A}}$

experiment for an adversary $\mathcal{A}$ against the AS-UNF-3 security of ASvKA. $\mathcal{B}$ answers the queries for $\mathcal{A}$ in the same way as in the proof of Theorem 1, except that during $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L, apk, \pi)$ queries from $\mathcal{A}$, $\mathcal{B}$ calls its own first-round signing oracle $z_{k,1}^{\mathsf{MS}} \leftarrow \mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, \bar{m} := (m, apk), PK')$ instead of $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m, PK')$. Besides, $\mathcal{B}$ uses dsk to compute $(\tilde{\mathsf{st}}_{k,1}, \tilde{z}_{k,1}) \leftarrow \mathsf{MS.MulSign}_1(dsk, \bar{m} := (m, apk), PK'; \lambda = 0)$ instead of $\mathsf{MS.MulSign}_1(dsk, m, PK'; \lambda = 0)$. After receiving a forgery $(\sigma^*, L^*, \pi^*, apk^*)$ from $\mathcal{A}$ where $L^* = \{(X_i^*, m_i^*)\}_{i\in[n]}$, $\mathcal{B}$ parses $\sigma^* = (\sigma'^*, dpk^*)$, computes PK^*, m^*, Index^*, PK'^* in the same way as (1), and returns $(PK'^*, \bar{m}^* := (m^*, apk^*), \sigma'^*)$ as its own forgery. In this proof, the winning condition (iii′) for $\mathcal{B}$ will be changed to

(iii′) $(\bar{m}^* = (m^*, apk^*), \cdot) \notin Q_{\mathsf{MS}}$.

Similarly, we consider a contradiction that $\mathcal{B}$ outputs an invalid MS-UNF-1 forgery, i.e., $(\bar{m}^* = (m^*, apk^*), \cdot) \in Q_{\mathsf{MS}}$, which means that $\mathcal{B}$ has queried its first-round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, \bar{m}^* = (m^*, apk^*), \cdot)$ for some k, and this query must have been triggered by the first-round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L^*, apk^*, \pi)$ query from $\mathcal{A}$. In addition, $\mathcal{B}$ has queried its R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ oracle for this k, which is triggered by the R-th round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query from $\mathcal{A}$. With a similar analysis as that in the proof of Theorem 1, we can show that during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}$ query, for all $i \in [n]$ s.t. $X_i^* = X^*$, the tuples (k, m_i^*, L^*, apk^*) will be added to Q_1, and furthermore, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query, the tuples (m_i^*, L^*, apk^*) will be added to Q, which contradicts (iv). This shows that (iii′) must hold and we conclude the theorem. We provide a full proof of Theorem 2 in Appendix C.2. □

5 Aggregate-Signature Instantiations

In this section, we present two instantiations of our generic transformation. The first instantiation is constructed from the BLS multi-signature scheme MS-BLS proposed by Boneh et al. [6], and yields two non-interactive and pairing-based ASvKA schemes PP-BAS-0, PP-BAS-1 via our transformation. The second instantiation is constructed from a Schnorr multi-signature scheme SpeedyMuSig with proofs of possession proposed in [13, 2], and yields a two-round but pairing-free ASvKA scheme PP-SpeedyASvKA via our transformation. We note that our instantiations PP-BAS-1 and PP-SpeedyASvKA achieve the strongest AS-UNF-3 unforgeability and the strongest AS-FullPriv privacy simultaneously.

5.1 Non-Interactive Pairing-Based Aggregate-Signatures

Now we instantiate our transformation based on the BLS multi-signature scheme MS-BLS proposed by Boneh et al. [6], which is non-interactive and pairing-based.

More precisely, via Construction 1 and Construction 2 of our transformation, we can obtain two non-interactive pairing-based ASvKA schemes PP-BAS-0 and

PP-BAS-1, respectively. We will show that these two schemes achieve different levels of unforgeability and privacy.

- PP-BAS-0 achieves the AS-UNF-2 unforgeability and the strongest privacy AS-FullPriv against the standard BLS scheme [8].
- PP-BAS-1 achieves the strongest AS-UNF-3 unforgeability and the strongest privacy AS-FullPriv against a key-prefixed version of BLS.

Instantiations. We present the concrete instantiations PP-BAS-0 and PP-BAS-1 in Fig. 5 and provide explanations for specific algorithms as follows.

- Both schemes rely on a pairing group $\mathsf{gpar} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerated bilinear pairing, and g is a generator of $\mathbb{G}_2$. For simplicity, we require gpar to be an implicit input of the algorithms of both schemes.
- In both schemes, we instantiate the linear hash function $\mathsf{F} : \mathcal{D} \to \mathcal{R}$ as $\mathsf{F}(x) := g^x$, where $\mathcal{D} := \mathbb{Z}_p$, the input $x \in \mathbb{Z}_p$ and $\mathcal{R} := \mathbb{G}_2$.
- The differences between the two schemes lie in the ASvKA.AggSign and ASvKA.Vrfy algorithms: in PP-BAS-0, m is used as the input to $\mathsf{H}_{\mathsf{sig}}$; whereas in PP-BAS-1, apk and m are bound together and used as the input to $\mathsf{H}_{\mathsf{sig}}$.
- In both schemes, the ASvKA.Combine algorithm does not use the proof of aggregation π (so we omit π from the input), and does not need to output the dummy public key dpk as part of the aggregate-signature σ.
- Both schemes satisfy "*aggregated verification*" (as per Def. 2), i.e., the aggregate-signature σ can be verified via an aggregated verification algorithm ASvKA.AggVrfy with a combined message $m := \mathsf{H}_{\mathsf{com}}(L)$ and apk (instead of L and apk). The description of ASvKA.AggVrfy is also provided in Fig. 5.

It is straightforward to verify the correctness of the two schemes. For PP-BAS-0, the aggregate-signature is $\sigma := \tilde{z} \cdot \prod_{i \in \mathsf{Index}} z_i = \mathsf{H}_{\mathsf{sig}}(m)^{dsk} \cdot \prod_{i \in \mathsf{Index}} \mathsf{H}_{\mathsf{sig}}(m)^{sk_i} = \mathsf{H}_{\mathsf{sig}}(m)^{dsk + \sum_{i \in \mathsf{Index}} sk_i}$. By the properties of bilinear pairing map e, we have

$$\begin{aligned} e(\sigma, g) &= e(\mathsf{H}_{\mathsf{sig}}(m)^{dsk + \sum_{i \in \mathsf{Index}} sk_i}, g) = e(\mathsf{H}_{\mathsf{sig}}(m), g^{dsk + \sum_{i \in \mathsf{Index}} sk_i}) \\ &= e(\mathsf{H}_{\mathsf{sig}}(m), g^{dsk} \cdot \prod_{i \in \mathsf{Index}} X_i) = e(\mathsf{H}_{\mathsf{sig}}(m), apk), \end{aligned}$$

so the signature verification will output 1. By replacing $\mathsf{H}_{\mathsf{sig}}(m)$ with $\mathsf{H}_{\mathsf{sig}}(apk, m)$, we can show the correctness of PP-BAS-1.

Unforgeability of PP-BAS-0 and PP-BAS-1. Boneh et al. [6] has shown the MS-UNF-1 security of the multi-signature scheme MS-BLS (as recalled in Lemma 1 below). So by our Theorem 1 and Theorem 2, it follows that PP-BAS-0 satisfies AS-UNF-2 unforgeability, while PP-BAS-1 satisfies the strongest AS-UNF-3 unforgeability, as summarized in the following corollary.

Lemma 1 (Unforgeability of MS-BLS [6]). *The multi-signature scheme* MS-BLS *proposed in [6] is* MS-UNF-1 *secure under the co-CDH assumption in the random oracle model.*

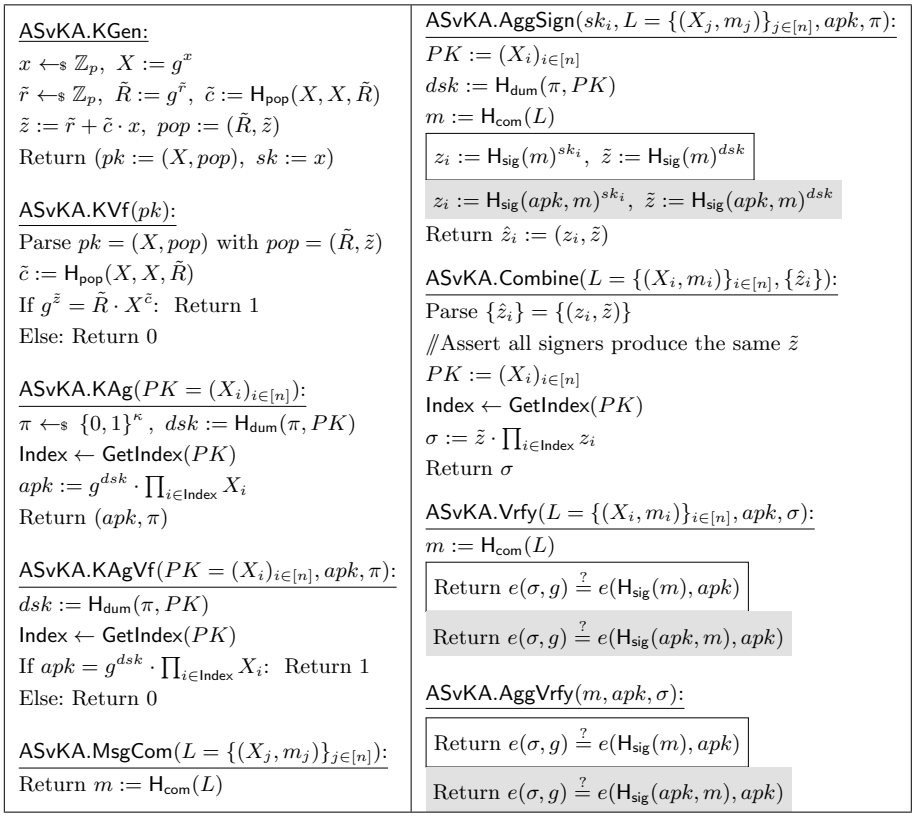

Fig. 5. Our non-interactive pairing-based instantiations PP-BAS-0 (with framed boxes) and PP-BAS-1 (with gray boxes), where $\mathsf{H}_{\mathsf{pop}} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_{\mathsf{dum}} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{G}_1, \mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^\kappa$ are hash functions, and $\mathsf{GetIndex}(PK)$ is the auxiliary algorithm that retains indices of distinct public keys in PK (cf. Fig. 4).

Corollary 1 (Unforgeability of PP-BAS-0 and PP-BAS-1). PP-BAS-0 *and* PP-BAS-1 *in Fig. 5 achieve* AS-UNF-2 *security and* AS-UNF-3 *security, respectively, under the co-CDH assumption*[7] *in the random oracle model.*

Privacy of PP-BAS-0. Below we will show that PP-BAS-0 achieves the strongest AS-FullPriv, i.e., an aggregated public key and aggregate-signatures of PP-BAS-0 are indistinguishable from a public key and signatures of the BLS scheme.

Theorem 3 (Full Privacy of PP-BAS-0). PP-BAS-0 *achieves* AS-FullPriv *against the standard BLS scheme (cf. Appendix B.2) in the random oracle model.*

Proof of Theorem 3. Let $\mathcal{A}$ be a PPT adversary against the AS-FullPriv of PP-BAS-0. Our proof proceeds with a sequence of games, where Game_0 is the

[7] This is the assumption that the security of MS-BLS is based on [6], and we provide a formal definition in Appendix B.1 for completeness.

$\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 0$ (cf. Fig. 3) and Game_3 is the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 1$. Let Ev_i be the event that $\mathcal{A}$ outputs $b^* = 1$ in Game_i.

- Game_0 : It is the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 0$. More precisely, let $(pk_i = (X_i, pop_i), sk_i = x_i)$ be the key pair of signer $i \in [n]$. In this game, upon receiving a signer set S^* from $\mathcal{A}$, we set pk^* as an aggregated public key for $PK_{S^*} = (X_i)_{i\in S^*}$, i.e., $pk^* := apk^* := g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} X_i$, where $dsk^* := \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ with $\pi^* \leftarrow_{\$} \{0,1\}^\kappa$ and $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK_{S^*})$. Then when answering $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries for $\mathcal{A}$, we compute σ as an aggregate-signature for $L_{S^*} = \{(X_i, m_i)\}_{i\in S^*}$, i.e., $\sigma := \tilde{z} \cdot \prod_{i\in\mathsf{Index}} z_i = \mathsf{H}_{\mathsf{sig}}(m)^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} \mathsf{H}_{\mathsf{sig}}(m)^{sk_i}$, where $m := \mathsf{H}_{\mathsf{com}}(L_{S^*})$.
- Game_1 : This game introduces an abort condition for $\mathcal{A}$'s random oracle queries to $\mathsf{H}_{\mathsf{dum}}$. More precisely, if $\mathcal{A}$ makes a $\mathsf{H}_{\mathsf{dum}}$ query on $(\pi^*, \cdot)$, Game_1 aborts. Since π^* is only used in the computation of $dsk^* = \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ in the game, by the random oracle model, $\mathcal{A}$ learns nothing about π^* and can hardly query $\mathsf{H}_{\mathsf{dum}}$ on $(\pi^*, \cdot)$. So we have $|\Pr[Ev_1] - \Pr[Ev_0]| \le q/2^\kappa$, where q denotes the number of $\mathsf{H}_{\mathsf{dum}}$ queries that $\mathcal{A}$ makes.
- Game_2 : This game is almost the same as Game_1, except how we compute $pk^* = apk^*$ and how we compute σ for $\mathcal{A}$'s $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries. More precisely, we first computes an *aggregated secret key* $ask^* := dsk^* + \sum_{i\in\mathsf{Index}} sk_i$, where $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK_{S^*})$. Then we use ask^* to compute $pk^* = apk^* := g^{ask^*}$, and compute $\sigma := \mathsf{H}_{\mathsf{sig}}(m)^{ask^*}$ for $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries, where $m := \mathsf{H}_{\mathsf{com}}(L_{S^*})$. It is easy to see that the changes are just conceptual, since $apk^* = g^{ask^*} = g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} g^{sk_i} = g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} X_i$, and $\sigma = \mathsf{H}_{\mathsf{sig}}(m)^{ask^*} = \mathsf{H}_{\mathsf{sig}}(m)^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} \mathsf{H}_{\mathsf{sig}}(m)^{sk_i}$. Thus $\Pr[Ev_2] = \Pr[Ev_1]$.
- Game_3 : This game is almost the same as Game_2, except that we sample $ask^* \leftarrow_{\$} \mathbb{Z}_p$ uniformly, instead of computing $ask^* := dsk^* + \sum_{i\in\mathsf{Index}} sk_i$. Note that by the change in Game_1, $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{dum}}$ on $(\pi^*, \cdot)$, thus $dsk^* = \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ is uniformly random over $\mathbb{Z}_p$ from $\mathcal{A}$'s view, and so is ask^*. Consequently, the change is conceptual and $\Pr[Ev_3] = \Pr[Ev_2]$.

  Finally, we note that Game_3 is exactly the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 1$. This is because in Game_3, we have $pk^* := g^{ask^*}$ and $\sigma := \mathsf{H}_{\mathsf{sig}}(m)^{ask^*}$ for $ask^* \leftarrow_{\$} \mathbb{Z}_p$, which are exactly the public key and signature of the standard BLS scheme (cf. Appendix B.2).

Overall, we have $\mathsf{Adv}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(\kappa) = \big|\Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(0) \Rightarrow 1] - \Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(1) \Rightarrow 1]\big| = \big|\Pr[Ev_0] - \Pr[Ev_3]\big| \le q/2^\kappa$ and conclude the theorem. □

Privacy of PP-BAS-1. We note that PP-BAS-1 cannot satisfy AS-FullPriv if Sign is the standard BLS signing algorithm, because its signature verification algorithm is not compatible with the standard BLS. However, we can still prove its AS-FullPriv against a specific Sign' algorithm, which is a *key-prefixed variant* of the BLS scheme, as defined below.

- **Key Generation.** The signer chooses $sk \leftarrow_{\$} \mathbb{Z}_p$ and computes $pk := g^{sk}$.
- **Signing.** In order to sign a message $m \in \{0,1\}^*$, the signer computes $\sigma := \mathsf{H}_{\mathsf{sig}}(pk, m)^{sk}$ and returns σ as its signature.

– **Signature Verification.** Verification checks $e(\sigma, g) \stackrel{?}{=} e(\mathsf{H}_{\mathsf{sig}}(pk, m), pk)$.

Theorem 4 (Full Privacy of PP-BAS-1). PP-BAS-1 *achieves* AS-FullPriv *against the* Sign′ *algorithm in the random oracle model.*

The proof is essentially the same as that for Theorem 3, except that we replace all $\mathsf{H}_{\mathsf{sig}}(m)$ with $\mathsf{H}_{\mathsf{sig}}(pk, m)/\mathsf{H}_{\mathsf{sig}}(apk, m)$. We refer to Appendix C.3 for a formal proof.

5.2 Two-Round Pairing-Free Aggregate-Signatures

In this subsection, we instantiate our transformation based on the Schnorr multi-signature scheme SpeedyMuSig [13, 2], which runs in 2 rounds and is pairing-free.

More precisely, via Construction 2 of our transformation, we can obtain a two-round pairing-free ASvKA scheme PP-SpeedyASvKA, which achieves the strongest AS-UNF-3 unforgeability and the strongest AS-FullPriv simultaneously.

Instantiations. We present the concrete instantiation PP-SpeedyASvKA in Fig. 6 and provide explanations for specific algorithms as follows.

– PP-SpeedyASvKA relies on a cyclic group $\mathsf{gpar} = (\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of prime order p, and g is a generator of $\mathbb{G}$. For simplicity, we require gpar to be an implicit input of the algorithms of PP-SpeedyASvKA.
– In PP-SpeedyASvKA, we instantiate the linear hash function $\mathsf{F} : \mathcal{D} \to \mathcal{R}$ as $\mathsf{F}(x) := g^x$, where $\mathcal{D} := \mathbb{Z}_p$, the input $x \in \mathbb{Z}_p$ and $\mathcal{R} := \mathbb{G}$.
– Similar to the instantiations in Subsect. 5.1, the ASvKA.Combine algorithm does not need the proof of aggregation π as input, and does not need to output the dummy public key dpk as part of the aggregate-signature σ. Moreover, PP-SpeedyASvKA satisfies "*aggregated verification*" (as per Def. 2).

It is straightforward to verify the correctness of PP-SpeedyASvKA. By expanding the left side of the verification equation, we have

$$g^z = g^{\tilde{z}+\sum_{i\in\mathsf{Index}} z_i} = g^{c\cdot dsk} \cdot \prod_{i\in\mathsf{Index}} g^{r_i+b\cdot s_i+c\cdot sk_i} = g^{c\cdot(dsk+\sum_{i\in\mathsf{Index}} sk_i)} \cdot \bar{R} \cdot \bar{S}^b = apk^c \cdot \hat{R}.$$

Unforgeability of PP-SpeedyASvKA. Crites et al. [13, 2] have shown the MS-UNF-1 of SpeedyMuSig (as recalled in Lemma 2 below). So by our Theorem 2, it follows that our PP-SpeedyASvKA satisfies the strongest AS-UNF-3 unforgeability.

Lemma 2 (Unforgeability of SpeedyMuSig [13, 2]). *The multi-signature scheme* SpeedyMuSig *proposed in [13, 2] is* MS-UNF-1 *secure under the one-more discrete logarithm* (OMDL) *assumption and the Schnorr knowledge of exponent* (schnorr-koe) *assumption in the random oracle model.*

Corollary 2 (Unforgeability of PP-SpeedyASvKA). PP-SpeedyASvKA *in Fig. 6 achieves* AS-UNF-3 *security under the* OMDL *and the* schnorr-koe *assumptions*[8] *in the random oracle model.*

[8] These are the assumptions that the security of SpeedyMuSig is based on [13, 2], and we provide a formal definition in Appendix B.1 for completeness.

ASvKA.KGen:
$x \leftarrow\!\!\$\ \mathbb{Z}_p,\ X := g^x$
$\tilde{r} \leftarrow\!\!\$\ \mathbb{Z}_p,\ \tilde{R} := g^{\tilde{r}},\ \tilde{c} := \mathsf{H}_{\mathsf{pop}}(X, X, \tilde{R})$
$\tilde{z} := \tilde{r} + \tilde{c} \cdot x,\ pop := (\tilde{R}, \tilde{z})$
Return $(pk := (X, pop),\ sk := x)$

ASvKA.KVf(pk):
Parse $pk = (X, pop)$ with $pop = (\tilde{R}, \tilde{z})$
$\tilde{c} := \mathsf{H}_{\mathsf{pop}}(X, X, \tilde{R})$
If $g^{\tilde{z}} = \tilde{R} \cdot X^{\tilde{c}}$: Return 1
Else: Return 0

ASvKA.ASvKA.KAg$(PK = (X_i)_{i\in[n]})$:
$\pi \leftarrow\!\!\$\ \{0,1\}^\kappa,\ dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK)$
$\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$
$apk := g^{dsk} \cdot \prod_{i\in\mathsf{Index}} X_i$
Return (apk, π)

ASvKA.KAgVf$(PK = (X_i)_{i\in[n]}, apk, \pi)$:
$dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK)$
$\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$
If $apk = g^{dsk} \cdot \prod_{i\in\mathsf{Index}} X_i$: Return 1
Else: Return 0

ASvKA.MsgCom$(L = \{(X_j, m_j)\}_{j\in[n]})$:
Return $m := \mathsf{H}_{\mathsf{com}}(L)$

ASvKA.AggSign$_1(sk_i, L = \{(X_j, m_j)\}_{j\in[n]}, apk, \pi)$:
$r_i \leftarrow\!\!\$\ \mathbb{Z}_p,\ R_i := g^{r_i}$
$s_i \leftarrow\!\!\$\ \mathbb{Z}_p,\ S_i := g^{s_i}$
$\rho_i := (R_i, S_i),\ \mathsf{st}_i := (r_i, s_i, sk_i, L, apk, \pi)$
Return (ρ_i, st_i)

ASvKA.AggSign$_2(\mathsf{st}_i, \mathsf{inp}_i)$:
Parse $\mathsf{st}_i = (r_i, s_i, sk_i, L, apk, \pi)$
with $L = \{(X_j, m_j)\}_{j\in[n]}$
Parse $\mathsf{inp}_i = \{\rho_j = (R_j, S_j)\}_j$
$PK := (X_j)_{j\in[n]}$
$dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK),\ m := \mathsf{H}_{\mathsf{com}}(L)$
$\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$
$\bar{R} := \prod_{j\in\mathsf{Index}} R_j,\ \bar{S} := \prod_{j\in\mathsf{Index}} S_j$
$b := \mathsf{H}_{\mathsf{non}}(apk, m, \bar{R}, \bar{S})$
$\hat{R} := \bar{R} \cdot \bar{S}^b,\ c := \mathsf{H}_{\mathsf{sig}}(apk, m, \hat{R})$
$z_i := r_i + b \cdot s_i + c \cdot sk_i$
$\tilde{z} := c \cdot dsk$
Return $\hat{z}_i := (\hat{R}, z_i, \tilde{z})$

ASvKA.Combine$(L = \{(X_i, m_i)\}_{i\in[n]}, \{\hat{z}_i\})$:
Parse $\{\hat{z}_i\} = \{(\hat{R}, z_i, \tilde{z})\}$
//Assert all signers produce the same $\hat{R}$ and $\tilde{z}$
$PK := (X_j)_{j\in[n]}$
$\mathsf{Index} := \mathsf{GetIndex}(PK)$
$z := \sum_{i\in\mathsf{Index}} z_i + \tilde{z}$
Return $\sigma := (\hat{R}, z)$

ASvKA.Vrfy$(L = \{(X_i, m_i)\}_{i\in[n]}, apk, \sigma)$:
Parse $\sigma = (\hat{R}, z)$
$m := \mathsf{H}_{\mathsf{com}}(L)$
$c := \mathsf{H}_{\mathsf{sig}}(apk, m, \hat{R})$
Return $g^z \stackrel{?}{=} \hat{R} \cdot apk^c$

ASvKA.AggVrfy(m, apk, σ):
Parse $\sigma = (\hat{R}, z)$
$c := \mathsf{H}_{\mathsf{sig}}(apk, m, \hat{R})$
Return $g^z \stackrel{?}{=} \hat{R} \cdot apk^c$

Fig. 6. Our two-round pairing-free instantiation PP-SpeedyASvKA, where $\mathsf{H}_{\mathsf{pop}} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_{\mathsf{dum}} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathsf{H}_{\mathsf{non}} : \{0,1\}^* \to \mathbb{Z}_p$ and $\mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$ are hash functions, and $\mathsf{GetIndex}(PK)$ is the auxiliary internal algorithm that retains indices of distinct public keys in PK (cf. Fig. 4).

Privacy of PP-SpeedyASvKA. Below we show that PP-SpeedyASvKA also achieves the strongest privacy AS-FullPriv, i.e., an aggregated public key and aggregate-signatures of PP-SpeedyASvKA are indistinguishable from a public key and signatures of the standard Schnorr scheme.

Theorem 5 (Full Privacy of PP-SpeedyASvKA). PP-SpeedyASvKA *achieves* AS-FullPriv *against the standard Schnorr scheme (cf. Appendix B.2) in the random oracle model.*

The proof is similar to that for Theorem 3, and we postpone it to Appendix C.4.

References

[1] Abou Haidar, C., Das, D., Lehmann, A., Özbay, C., Kempner, O.P.: Privacy-preserving multi-signatures: Generic techniques and constructions without pairings. In: PKC 2025. pp. 66–98 (2025)

[2] Bellare, M., Crites, E.C., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: CRYPTO 2022

[3] Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: ICALP 2007. pp. 411–422 (2007)

[4] Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM CCS 2006. pp. 390–399 (2006)

[5] Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: ACM CCS 2007. pp. 276–285 (2007)

[6] Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: ASIACRYPT 2018. pp. 435–464 (2018)

[7] Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: EUROCRYPT 2003. pp. 416–432 (2003)

[8] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: ASIACRYPT 2001. pp. 514–532 (2001)

[9] Brogle, K., Goldberg, S., Reyzin, L.: Sequential aggregate signatures with lazy verification from trapdoor permutations. Information and Computation (2014)

[10] Chalkias, K., Garillot, F., Kondi, Y., Nikolaenko, V.: Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures. In: CT-RSA 2021

[11] Chen, Y.: Dazzle: Improved adaptive threshold signatures from DDH. In: PKC 2025. pp. 233–261 (2025)

[12] Chen, Y., Zhao, Y.: Half-aggregation of Schnorr signatures with tight reductions. In: ESORICS 2022. pp. 385–404 (2022)

[13] Crites, E., Komlo, C., Maller, M.: How to prove Schnorr assuming Schnorr: Security of multi-and threshold signatures. ePrint (2021), `https://ia.cr/2021/1375`

[14] Crites, E.C., Komlo, C., Maller, M.: Fully adaptive Schnorr threshold signatures. In: CRYPTO 2023. pp. 678–709 (2023)

[15] Hohenberger, S., Waters, B., Wu, D.J.: Pairing-based aggregate signatures without random oracles. In: ASIACRYPT 2025. pp. 271–303 (2025)

[16] Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: SAC 2020. pp. 34–65 (2020)

[17] Lacharité, M.S.: Security of BLS and BGLS signatures in a multi-user setting. Cryptography and Communications pp. 41–58 (2018)

[18] Lehmann, A., Özbay, C.: Multi-signatures for ad-hoc and privacy-preserving group signing. In: PKC 2024. pp. 196–228 (2024)

[19] Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: EUROCRYPT 2006

[20] Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: EUROCRYPT 2004. pp. 74–90 (2004)

[21] Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to bitcoin. Designs, Codes and Cryptography (2019)

[22] Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures. In: ACM CCS 2001. pp. 245–254 (2001)

[23] Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round Schnorr multi-signatures. In: CRYPTO 2021. pp. 189–221 (2021)

[24] Nick, J., Ruffing, T., Seurin, Y.: Dahlias: Discrete logarithm-based interactive aggregate signatures. In: EUROCRYPT 2026 (2026), https://ia.cr/2025/692
[25] Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: EUROCRYPT 2007. pp. 228–245 (2007)
[26] Sakai, Y.: Aggregate signatures tightly secure under adaptive corruptions. In: ASIACRYPT 2025. pp. 304–336 (2025)
[27] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology pp. 161–174 (1991)
[28] Tessaro, S., Zhu, C.: Threshold and multi-signature schemes from linear hash functions. In: EUROCRYPT 2023. pp. 628–658 (2023)

A Further Preliminaries

A.1 Linear Hash Function

Definition 5 (Linear Hash Function [28]). *A deterministic function* $\mathsf{F} : \mathcal{D} \to \mathcal{R}$ *is a linear hash function, if (i)* $\mathcal{D}$ *and* $\mathcal{R}$ *are two* $\mathcal{S}$*-modules for a field* $\mathcal{S}$ *satisfying* $|\mathcal{S}| \geq 2^\kappa$, $|\mathcal{D}| \geq 2^\kappa$, $|\mathcal{R}| \geq 2^\kappa$, *(ii)* F *is an epimorphism of* $\mathcal{S}$*-modules, i.e., for any* $a, a' \in \mathcal{S}$ *and* $x, x' \in \mathcal{D}$, *we have* $\mathsf{F}(ax + a'x') = a\mathsf{F}(x) + a'\mathsf{F}(x')$.

A.2 Multi-Signature Schemes

Definition 6 (Multi-Signature MS). *A multi-signature scheme* MS *consists of a tuple of PPT algorithms* $\mathsf{MS} = (\mathsf{KGen}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vrfy})$ *s.t.:*

- $\underline{(pk, sk) \leftarrow \mathsf{KGen}}$*: The key generation algorithm generates a pair of public key and secret key* (pk, sk).
- $\underline{z_i \leftarrow \mathsf{MulSign}(sk_i, m, PK)}$*: On input the secret key* sk_i, *a message* m *and a set* $PK = \{pk_i\}_{i\in[n]}$ *of public keys, the signing algorithm outputs a signature share* z_i. *If* MulSign *is an interactive protocol with multiple rounds (say* R *rounds), we denote sub-algorithms* $\mathsf{MulSign}_j$ $(j \in R)$ *as follows:*

$$
\begin{aligned}
&(\mathsf{st}_{i,1}, z_{i,1}) \leftarrow \mathsf{MulSign}_1(sk_i, m, PK)\\
&(\mathsf{st}_{i,j}, z_{i,j}) \leftarrow \mathsf{MulSign}_j(\mathsf{st}_{i,j-1}, \mathsf{inp}_{i,j-1}),\ j \in [2, R-1]\\
&z_i \leftarrow \mathsf{MulSign}_R(\mathsf{st}_{i,R-1}, \mathsf{inp}_{i,R-1})
\end{aligned}
$$

  where $\mathsf{st}_{i,j}$ *denotes the state of the signer* i *after the* j*-th signing round,* $z_{i,j}$ *denotes the output of the signer* i *in round* j, *and* $\mathsf{inp}_{i,j-1} = \{z_{k,j-1}\}_{k\in[n]}$ *is the set of outputs for round* $j-1$ *from all signers* $k \in [n]$.
- $\underline{\sigma \leftarrow \mathsf{Combine}(\{z_i\} \in [n])}$*: The signature combining algorithm takes a set of signature shares* $\{z_i\} \in [n]$ *as input, and outputs a combined signature* σ.
- $\underline{0/1 \leftarrow \mathsf{Vrfy}(PK, m, \sigma)}$*: The signature verification algorithms takes a set* $PK = \{pk_i\}_{i\in[n]}$ *of public keys, a message* m *and a signature* σ *as input, and outputs a bit indicating whether* σ *is valid signature.*

Correctness. *For any number of signers* n, *any message* m, *any* $(pk_i, sk_i) \leftarrow \mathsf{KGen}$ *with* $i \in [n]$, *let* $PK := \{pk_i\}_{i\in[n]}$, *for any* $z_i \leftarrow \mathsf{MulSign}(sk_i, m, PK)$ *with* $i \in [n]$, *and any* $\sigma \leftarrow \mathsf{Combine}(\{z_i\})$, *it holds that* $\mathsf{Vrfy}(PK, m, \sigma) = 1$.

Unforgeability Definitions. For MS, there exists two variants of unforgeability [25, 22], which correspond to different conditions for trivial forgery.

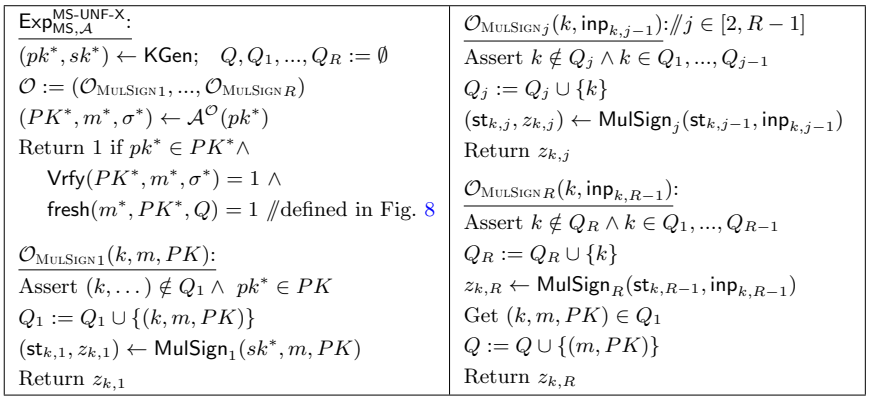

Fig. 7. The experiment $\mathsf{Exp}^{\mathsf{MS\text{-}UNF\text{-}X}}_{\mathsf{MS},\mathcal{A}}$ for MS, where fresh is defined in Fig. 8.

MS-UNF-X	MS-UNF-1 [25]	MS-UNF-2 [22]
$\mathsf{fresh}(m^*, PK^*, Q) = 1$ if	$(m^*, \cdot) \notin Q$	$(m^*, PK^*) \notin Q$

Fig. 8. Definitions of fresh for different levels of unforgeability in MS.

Definition 7 (Unforgeability for MS). *A multi-signature scheme MS is $\mathsf{MS\text{-}UNF\text{-}X}$ secure, if for any PPT $\mathcal{A}$, it holds that $\mathsf{Adv}^{\mathsf{MS\text{-}UNF\text{-}X}}_{\mathsf{MS},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\mathsf{MS\text{-}UNF\text{-}X}}_{\mathsf{MS},\mathcal{A}} \Rightarrow 1] \leq \mathsf{negl}(\kappa)$, where the experiment $\mathsf{Exp}^{\mathsf{MS\text{-}UNF\text{-}X}}_{\mathsf{MS},\mathcal{A}}$ is defined in Fig. 7.*

B Further Preliminaries (for Full Version)

B.1 Pairing Groups, Cyclic Groups and Assumptions

- **Pairing Groups.** Let $\mathsf{gpar} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g)$ be a description of pairing group, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerated bilinear pairing, and g is a generator of $\mathbb{G}_2$.
- **Cyclic Groups.** Let $\mathsf{gpar} = (\mathbb{G}, p, g)$ a description of cyclic group, where $\mathbb{G}$ is a cyclic group of prime order p, and g is a generator of $\mathbb{G}$.

Definition 8 (co-CDH Assumption). *The co-CDH assumption holds over a pairing group $\mathsf{gpar} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g)$, if for any PPT adversary $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathsf{co\text{-}CDH}}_{\mathcal{A}}(\kappa) := \Pr[h \leftarrow_{\$} \mathbb{G}_1, a \leftarrow_{\$} \mathbb{Z}_p : \mathcal{A}(\mathsf{gpar}, g^a, h) = h^a] \leq \mathsf{negl}(\kappa).$$

Definition 9 (One-More Discrete Logarithm (OMDL) Assumption [13]). *The OMDL assumption holds over a cyclic group $\mathsf{gpar} = (\mathbb{G}, p, g)$, if for any PPT adversary $\mathcal{A}$ and any polynomial n, we have*

$$\mathsf{Adv}^{\mathsf{OMDL}}_{\mathcal{A},n}(\kappa) := \Pr\left[(x_1, ..., x_n) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\text{DL}}}(\mathsf{gpar}, X_1, ..., X_n) : X_i = g^{x_i}\ \forall\ i \in [n]\right] \leq \mathsf{negl}(\kappa),$$

where $(X_1, ..., X_n) \leftarrow\!\!\$\ \mathbb{G}^n$ *and* $\mathcal{O}_{\mathrm{DL}}$ *is a discrete logarithm oracle that can be called by* $\mathcal{A}$ *for at most* $n-1$ *times.*

Definition 10 (Schnorr Knowledge of Exponent (schnorr-koe) Assumption [13]). *The Schnorr knowledge of exponent assumption holds over a cyclic group* $\mathsf{gpar} = (\mathbb{G}, p, g)$ *w.r.t. a hash function* $\tilde{\mathsf{H}}_{\mathsf{reg}} : \{0,1\}^* \to \mathbb{Z}_p$*, if for any PPT adversary* $\mathcal{A}$*, there exists a PPT extractor* Ext*, s.t.*

$$\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\kappa) := \Pr[\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}} \Rightarrow 1] \leq \mathsf{negl}(\kappa),$$

where the experiment $\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}$ *is defined in Fig. 9.*

$\underline{\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}}$:	$\underline{\mathcal{O}^{\mathsf{chal}}(X^*, \bar{R}^*, z^*)}$:
$Q_{\mathsf{sch\text{-}pop}}, Q_{\mathsf{reg}} := \emptyset$; $\mathsf{Win} := \mathsf{false}$	$\bar{c}^* := \tilde{\mathsf{H}}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*)$
$\omega \leftarrow\!\!\$\ \{0,1\}^{\mathsf{rl}_{\mathcal{A}}}$ //random coins for $\mathcal{A}$	If $(X^*, \bar{R}^*, \bar{z}^*) \in Q_{\mathsf{sch\text{-}pop}} \vee \bar{R}^* \cdot (X^*)^{\bar{c}^*} \neq g^{\bar{z}^*}$:
$\{0,1\}^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{sch\text{-}pop}}, \mathcal{O}^{\mathsf{chal}}, \mathcal{O}^{\mathsf{R\tilde{O}}}}(\mathsf{gpar}; \omega)$	Return $\perp$
Return Win	$x^* \leftarrow \mathsf{Ext}(\mathbb{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, Q_{\mathsf{reg}})$
$\underline{\mathcal{O}^{\mathsf{sch\text{-}pop}}()}$:	If $g^{x^*} \neq X^*$: $\mathsf{Win} := \mathsf{true}$
$x, \bar{r} \leftarrow\!\!\$\ \mathbb{Z}_p$	Return x^*
$X := g^x$, $\bar{R} := g^{\bar{r}}$, $\bar{c} := \tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$	$\underline{\mathcal{O}^{\mathsf{R\tilde{O}}}(\theta)}$: //random oracle
$\bar{z} := \bar{r} + \bar{c}x$ // proof of knowledge of x	If $\tilde{\mathsf{H}}_{\mathsf{reg}}(\theta) = \perp$: $\tilde{\mathsf{H}}_{\mathsf{reg}}(\theta) \leftarrow\!\!\$\ \mathbb{Z}_p$
$Q_{\mathsf{sch\text{-}pop}} := Q_{\mathsf{sch\text{-}pop}} \cup \{(X, \bar{R}, \bar{z})\}$	Return $\tilde{\mathsf{H}}_{\mathsf{reg}}(\theta)$
Return $(X, \bar{R}, \bar{z})$	

Fig. 9. The experiment $\mathsf{Exp}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}$ for defining the Schnorr knowledge of exponent (schnorr-koe) assumption over a cyclic group $\mathsf{gpar} = (\mathbb{G}, p, g)$ w.r.t. a hash function $\tilde{\mathsf{H}}_{\mathsf{reg}} : \{0,1\}^* \to \mathbb{Z}_p$, where $\mathsf{rl}_{\mathcal{A}}$ denotes the randomness length of $\mathcal{A}$.

B.2 The Standard BLS and Schnorr Schemes

- **The BLS Scheme [8].** Let g be a generator of $\mathbb{G}_2$ and $\mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{G}_1$ be a random oracle. The signer chooses a secret key $sk \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the public key $pk := g^{sk}$. The signature of a message m is $\sigma := \mathsf{H}_{\mathsf{sig}}(m)^{sk}$, which can be verified by checking $e(\sigma, g) \stackrel{?}{=} e(\mathsf{H}_{\mathsf{sig}}(m), pk)$.
- **The Schnorr Scheme [27].** Let g be a generator of $\mathbb{G}$ and $\mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$ be a random oracle. The signer chooses a secret key $sk \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the public key $pk := g^{sk}$. The signature of a message m is $\sigma := (R, z)$, where $R := g^r$ with $r \leftarrow\!\!\$\ \mathbb{Z}_p$, $c := \mathsf{H}_{\mathsf{sig}}(pk, m, R)$ and $z := r + c \cdot sk$, which can be verified by checking $g^z \stackrel{?}{=} R \cdot pk^c$ with $c := \mathsf{H}_{\mathsf{sig}}(pk, m, R)$.

B.3 Existing Syntax of Aggregate-Signature Schemes

We recall the existing syntax of an aggregate-signature scheme (AS) and revisit two different unforgeability notions that have been proposed in the literature.

Definition 11 (Aggregate-Signature AS [10, 12, 24]). *An aggregate-signature scheme* AS *consists of a tuple of PPT algorithms* $\mathsf{AS} = (\mathsf{KGen}, \mathsf{AggSign}, \mathsf{Combine}, \mathsf{Vrfy})$ *s.t.:*

- $\underline{(pk, sk) \leftarrow \mathsf{KGen}}$*: The key generation algorithm generates a pair of public key and secret key* (pk, sk)*.*
- $\underline{z_i \leftarrow \mathsf{AggSign}(sk_i, m_i, L)}$*: On input the secret key* sk_i*, a message* m_i *and a set* $L = \{(pk_i, m_i)\}_{i\in[n]}$ *of public keys and messages, the signing algorithm outputs a signature share* z_i*. If* $\mathsf{AggSign}$ *is an interactive protocol with multiple rounds (say* R *rounds), we denote sub-algorithms* $\mathsf{AggSign}_j$ $(j \in R)$ *as follows:*

$$
\begin{aligned}
&(\mathsf{st}_{i,1}, z_{i,1}) \leftarrow \mathsf{AggSign}_1(sk_i, m_i, L)\\
&(\mathsf{st}_{i,j}, z_{i,j}) \leftarrow \mathsf{AggSign}_j(\mathsf{st}_{i,j-1}, \mathsf{inp}_{i,j-1}),\ j \in [2, R-1]\\
&z_i \leftarrow \mathsf{AggSign}_R(\mathsf{st}_{i,R-1}, \mathsf{inp}_{i,R-1})
\end{aligned}
$$

  where $\mathsf{st}_{i,j}$ *denotes the state of the signer* i *after the* j*-th signing round,* $z_{i,j}$ *denotes the output of the signer* i *in round* j*, and* $\mathsf{inp}_{i,j-1} = \{z_{k,j-1}\}_{k\in[n]}$ *is the set of outputs for round* $j-1$ *from all signers* $k \in [n]$*.*
- $\underline{\sigma \leftarrow \mathsf{Combine}(\{z_i\}_{i\in[n]})}$*: The signature combining algorithm takes a set of signature shares* $\{z_i\}_{i\in[n]}$ *as input, and outputs an aggregate-signature* σ*.*
- $\underline{0/1 \leftarrow \mathsf{Vrfy}(L, \sigma)}$*: The signature verification algorithms takes a set* $L = \{(pk_i, m_i)\}_{i\in[n]}$ *of public keys and messages, and a signature* σ *as inputs, and outputs a bit indicating whether* σ *is valid signature.*

Correctness. *For any number of signers* n*, any messages* $m_1, ..., m_n$*, any* $(pk_i, sk_i) \leftarrow \mathsf{KGen}$ *with* $i \in [n]$*, let* $L := \{(pk_i, m_i)\}_{i\in[n]}$*, for any* $z_i \leftarrow \mathsf{AggSign}(sk_i, m_i, L)$ *with* $i \in [n]$ *and any* $\sigma \leftarrow \mathsf{Combine}(\{z_i\}_{i\in[n]})$*, it holds that* $\mathsf{Vrfy}(L, \sigma) = 1$*.*

Unforgeability Definitions. For AS schemes, there exists two variants for unforgeability, which correspond to different conditions for a trivial forgery. The $\mathsf{AS\text{-}UNF\text{-}1}$ definition was first proposed in [7], and requires the message (say m_i^*) to be fresh as the winning condition. A stronger $\mathsf{AS\text{-}UNF\text{-}2}$ version (proposed in [24]) requires the tuple (m_i^*, L^*) to be fresh. Both unforgeability experiments are shown in Fig. 10.

$\underline{\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{AS},\mathcal{A}}}$:	$\underline{\mathcal{O}_{\mathrm{AggSign}_j}(k, \mathsf{inp}_{k,j-1})\text{:} /\!/ j \in [2, R-1]}$
$(pk^*, sk^*) \leftarrow \mathsf{KGen};\quad Q, Q_1, ..., Q_R := \emptyset$	Assert $k \notin Q_j \wedge k \in Q_1, ..., Q_{j-1}$
$\mathcal{O} := (\mathcal{O}_{\mathrm{AggSign}_1}, ..., \mathcal{O}_{\mathrm{AggSign}_R})$	$Q_j := Q_j \cup \{k\}$
$(L^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pk^*)$	$(\mathsf{st}_{k,j}, z_{k,j}) \leftarrow \mathsf{AggSign}_j(\mathsf{st}_{k,j-1}, \mathsf{inp}_{k,j-1})$
Parse $L^* = \{(pk_i^*, m_i^*)\}_{i\in[n]}$	Return $z_{k,j}$
Return 1 if $\mathsf{Vrfy}(L^*, \sigma^*) = 1 \wedge$	$\underline{\mathcal{O}_{\mathrm{AggSign}_R}(k, \mathsf{inp}_{k,R-1})\text{:}}$
$\quad \exists\ i \in [n]$, s.t. $pk_i^* = pk^* \wedge \mathsf{fresh}(m_i^*, L^*, Q) = 1$	Assert $k \notin Q_R \wedge k \in Q_1, ..., Q_{R-1}$
$\underline{\mathcal{O}_{\mathrm{AggSign}_1}(k, m, L)\text{:}}$	$Q_R := Q_R \cup \{k\}$
Assert $(k, \cdots) \notin Q_1 \wedge (pk^*, m) \in L$	$z_{k,R} \leftarrow \mathsf{AggSign}_R(\mathsf{st}_{k,R-1}, \mathsf{inp}_{k,R-1})$
$Q_1 := Q_1 \cup \{(k, m, L)\}$	Get $(k, m, L) \in Q_1$
$(\mathsf{st}_{k,1}, z_{k,1}) \leftarrow \mathsf{AggSign}_1(sk^*, m, L)$	$Q := Q \cup \{(m, L)\}$
Return $z_{k,1}$	Return $z_{k,R}$

Fig. 10. The experiment $\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{AS},\mathcal{A}}$ for AS, where fresh is defined in Fig. 11.

AS-UNF-X	AS-UNF-1 [25]	AS-UNF-2 [22]
$\mathsf{fresh}(m_i^*, L^*, Q) = 1$ if	$(m_i^*, \cdot) \notin Q$	$(m_i^*, L^*) \notin Q$

Fig. 11. Definitions of fresh for different levels of unforgeability in AS.

Definition 12 (Unforgeability for AS). *An aggregate-signature scheme* AS *is* $\mathsf{AS\text{-}UNF\text{-}X}$ *secure, if for any PPT* $\mathcal{A}$*, it holds that* $\mathsf{Adv}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{AS},\mathcal{A}}(\kappa) := \Pr[\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{AS},\mathcal{A}} \Rightarrow 1] \leq \mathsf{negl}(\kappa)$*, where the experiment* $\mathsf{Exp}^{\mathsf{AS\text{-}UNF\text{-}X}}_{\mathsf{AS},\mathcal{A}}$ *is defined in Fig. 10.*

C Missing Proofs (for Full Version)

C.1 Proof of Theorem 1 (Unforgeability of Construction 1)

Firstly, we prove the first part of the theorem. Let $\mathcal{A}$ be a PPT adversary against the $\mathsf{AS\text{-}UNF\text{-}3}$ security of ASvKA and make at most q queries to $\mathcal{O}_{\text{REG}}, \mathcal{O}_{\text{AGGSIGN}1}, ..., \mathcal{O}_{\text{AGGSIGN}R}$ and random oracles $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$.

Below we construct a PPT adversary $\mathcal{B}$ against the $\mathsf{MS\text{-}UNF\text{-}2}$ security of MS by invoking $\mathcal{A}$. Firstly, $\mathcal{B}$ initializes arrays $\mathsf{T}_{\mathsf{pop}}, \mathsf{T}_{\mathsf{dum}}, \mathsf{T}_{\mathsf{com}}$ for $\mathcal{A}$'s queries to random oracles $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$. $\mathcal{B}$ also initializes sets $\mathcal{Q}_{\mathsf{apk}}, \mathcal{Q}_{\mathsf{msg}}$ to store aggregated public keys apk and combined messages m, initializes a set LPK to store the valid public key components which are verified via $\mathsf{ASvKA.KVf}$, and initializes sets $Q, Q_1, \cdots, Q_R$ for signing queries. We use three flags $\mathsf{BadError}, \mathsf{KeyColl}, \mathsf{MsgColl}$ initially set to false, to keep track of bad events.

Next, $\mathcal{B}$ receives $pk^*_{\mathsf{MS}} = X^*$ from its own challenger, and has access to its own oracles $\mathsf{MS}.\mathcal{O}_{\text{MULSIGN}1}, \cdots, \mathsf{MS}.\mathcal{O}_{\text{MULSIGN}R}$. Then $\mathcal{B}$ generates an ASvKA's public key $pk^* = (X^*, pop^* = (\tilde{R}^*, \tilde{z}^*))$ for the honest signer as follows: X^* is the MS's public key $pk^*_{\mathsf{MS}} = X^*$ that $\mathcal{B}$ receives from its own challenger, and to generate $pop^* = (\tilde{R}^*, \tilde{z}^*)$, $\mathcal{B}$ samples $\tilde{z}^*, \tilde{c}^* \leftarrow_{\$} \mathcal{D}$ uniformly, computes $\tilde{R}^* := \mathsf{F}(\tilde{z}^*) - \tilde{c}^* \cdot X^*$ and sets $\tilde{c}^* := \mathsf{H}_{\mathsf{pop}}(X^*, X^*, \tilde{R}^*)$. If $\mathsf{T}_{\mathsf{pop}}(X^*, X^*, \tilde{R}^*) \neq \perp$, $\mathcal{B}$ sets $\mathsf{BadError} := \mathsf{true}$ and returns $\perp$. Otherwise, $\mathcal{B}$ sends $pk^* = (X^*, pop^* = (\tilde{R}^*, \tilde{z}^*))$ to $\mathcal{A}$, and simulates the random oracles $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{dum}}, \mathsf{H}_{\mathsf{com}}$ and oracles $\mathcal{O}_{\text{REG}}, \mathcal{O}_{\text{AGGSIGN}1}, ..., \mathcal{O}_{\text{AGGSIGN}R}$ for $\mathcal{A}$ as follows.

- **Hash queries $\mathsf{H}_{\mathsf{pop}}(X, X, \tilde{R})$.** If $\mathsf{T}_{\mathsf{pop}}(X, X, \tilde{R}) = \perp$, $\mathcal{B}$ randomly chooses $\tilde{c} \leftarrow_{\$} \mathcal{S}$ and assigns $\mathsf{T}_{\mathsf{pop}}(X, X, \tilde{R}) := \tilde{c}$. Then $\mathcal{B}$ returns $\mathsf{T}_{\mathsf{pop}}(X, X, \tilde{R})$.
- **Hash queries $\mathsf{H}_{\mathsf{dum}}(\pi, PK)$.** If $\mathsf{T}_{\mathsf{dum}}(\pi, PK) = \perp$, $\mathcal{B}$ picks $dsk \leftarrow_{\$} \mathcal{D}$, computes $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$ and $apk := \sum_{i \in \mathsf{Index}} X_i + \mathsf{F}(dsk)$. If $apk \in \mathcal{Q}_{\mathsf{apk}}$, $\mathcal{B}$ sets $\mathsf{KeyColl} := \mathsf{true}$ and returns $\perp$. Otherwise, $\mathcal{B}$ adds apk to $\mathcal{Q}_{\mathsf{apk}}$ and sets $\mathsf{T}_{\mathsf{dum}}(\pi, PK) := dsk$. Then $\mathcal{B}$ returns $\mathsf{T}_{\mathsf{dum}}(\pi, PK)$.
- **Hash queries $\mathsf{H}_{\mathsf{com}}(L)$.** If $\mathsf{T}_{\mathsf{com}}(L) = \perp$, $\mathcal{B}$ randomly chooses $m \leftarrow_{\$} \{0,1\}^{\kappa}$. If $m \in \mathcal{Q}_{\mathsf{msg}}$, $\mathcal{B}$ sets $\mathsf{MsgColl} := \mathsf{true}$ and returns $\perp$. Otherwise, $\mathcal{B}$ adds m to $\mathcal{Q}_{\mathsf{msg}}$ and sets $\mathsf{T}_{\mathsf{com}}(L) := m$. Then $\mathcal{B}$ returns $\mathsf{T}_{\mathsf{com}}(L)$.
- **Register queries $\mathcal{O}_{\text{REG}}(pk)$,** where $pk = (X, pop = (\tilde{R}, \tilde{z}))$. $\mathcal{B}$ computes $\tilde{c} := \mathsf{H}_{\mathsf{pop}}(X, X, \tilde{R})$, and returns $\perp$ if $\mathsf{F}(\tilde{z}) \neq \tilde{R} + \tilde{c} \cdot X$. Otherwise, $\mathcal{B}$ adds X to LPK and returns 1.
- **The first signing round $\mathcal{O}_{\mathbf{AggSign}_1}(k, L, apk, \pi)$,** where $L = \{(X_i, m_i)\}_{i \in [n]}$. $\mathcal{B}$ sets $PK := (X_i)_{i \in [n]}$, and outputs $\perp$ directly if $\mathsf{KAgVf}(PK, apk, \pi) \neq 1$

or $X^* \notin PK$ or $PK \not\subseteq LPK$. Otherwise, for all $i \in [n]$ s.t. $X_i = X^*$, $\mathcal{B}$ adds (k, m_i, L, apk) to set Q_1. Then $\mathcal{B}$ computes $dsk := \mathsf{H}_{\mathsf{dum}}(\pi, PK)$, $dpk := \mathsf{F}(dsk)$, $m := \mathsf{H}_{\mathsf{com}}(L)$, $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK)$, and sets $PK' := \bigcup_{i \in \mathsf{Index}} \{X_i\} \cup \{dpk\}$. Next, $\mathcal{B}$ calls its own first round signing oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}$ to get $z^{\mathsf{MS}}_{k,1} \leftarrow \mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, m, PK')$, while computes $\tilde{z}_{k,1}$ using dsk by itself via $(\tilde{\mathsf{st}}_{k,1}, \tilde{z}_{k,1}) \leftarrow \mathsf{MS.MulSign}_1(dsk, m, PK'; \lambda = 0)$, and stores $(\tilde{\mathsf{st}}_{k,1}, \tilde{z}_{k,1})$ for the following oracle calls. Finally, $\mathcal{B}$ returns $z_{k,1} := z^{\mathsf{MS}}_{k,1}$ to $\mathcal{A}$.

– **The j-th ($j \geq 2$) signing round $\mathcal{O}_{\mathbf{AggSign}_j}(k, \mathsf{inp}^{\mathsf{ASvKA}}_{k,j-1})$.** If $k \in Q_j$ or $k \notin Q_1, ..., Q_{j-1}$, $\mathcal{B}$ returns $\perp$ directly. Otherwise, $\mathcal{B}$ adds k to Q_j. Then $\mathcal{B}$ extracts $(\tilde{\mathsf{st}}_{k,j-1}, \tilde{z}_{k,j-1})$ from its storage, and sets $\mathsf{inp}^{\mathsf{MS}}_{k,j-1} := \mathsf{inp}^{\mathsf{ASvKA}}_{k,j-1} \cup \{\tilde{z}_{k,j-1}\}$. Next, $\mathcal{B}$ calls its own j-th round signing oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}j}$ to get $z^{\mathsf{MS}}_{k,j} \leftarrow \mathsf{MS}.\mathcal{O}_{\text{MulSign}j}(k, \mathsf{inp}^{\mathsf{MS}}_{k,j-1})$, while computes $\tilde{z}_{k,j}$ using $\tilde{\mathsf{st}}_{k,j-1}$ by itself via $(\tilde{\mathsf{st}}_{k,j}, \tilde{z}_{k,j}) \leftarrow \mathsf{MS.MulSign}_j(\tilde{\mathsf{st}}_{k,j-1}, \mathsf{inp}^{\mathsf{MS}}_{i,j-1}; 0)$, and stores $(\tilde{\mathsf{st}}_{k,j}, \tilde{z}_{k,j})$. Finally, $\mathcal{B}$ returns $z^{\mathsf{MS}}_{k,j}$ to $\mathcal{A}$ if $j < R$, and returns $(z^{\mathsf{MS}}_{k,R}, \tilde{z}_{k,R})$ if $j = R$. If $j = R$, $\mathcal{B}$ also adds all (m_i, L, apk) s.t. $(k, m_i, L, apk) \in Q_1$ to set Q.

Eventually, $\mathcal{A}$ outputs a forgery $(\sigma^*, L^*, \pi^*, apk^*)$ where $L^* = \{(X^*_i, m^*_i)\}_{i \in [n]}$. $\mathcal{B}$ parses $\sigma^* = (\sigma'^*, dpk^*)$, and computes

$$\begin{aligned} &PK^* := (X^*_i)_{i \in [n]}, m^* := \mathsf{H}_{\mathsf{com}}(L^*), \\ &\mathsf{Index}^* \leftarrow \mathsf{GetIndex}(PK^*), PK'^* := \textstyle\bigcup_{i \in \mathsf{Index}^*} \{X^*_i\} \cup \{dpk^*\}. \end{aligned} \tag{2}$$

Finally, $\mathcal{B}$ returns (PK'^*, m^*, σ'^*) as its own forgery.

It is not hard to see that $\mathcal{B}$ simulates the AS-UNF-3 security experiment perfectly for $\mathcal{A}$, as long as BadError, KeyColl, MsgColl do not occur. In fact, these bad events can occur with only negligible probabilities, as shown below.

– BadError. During the generation of $pop^* = (\tilde{R}^*, \tilde{z}^*)$, we have $\tilde{R}^* := \mathsf{F}(\tilde{z}^*) - \tilde{c}^* \cdot X^*$ with $\tilde{z}^* \leftarrow_{\$} \mathcal{D}$, so $\tilde{R}^*$ is uniformly distributed over $\mathcal{R}$. Since $\mathcal{A}$ makes at most q oracle queries, the probability that $\mathsf{T}_{\mathsf{pop}}(X^*, X^*, \tilde{R}^*) \neq \perp$ is at most $q/|\mathcal{R}| \leq q/2^\kappa$. So $\Pr[\mathsf{BadError}] \leq q/2^\kappa$.
– KeyColl. During the $\mathsf{H}_{\mathsf{dum}}(\pi, PK)$ queries, we have $apk := \sum_{i \in \mathsf{Index}} X_i + \mathsf{F}(dsk)$ with $dsk \leftarrow_{\$} \mathcal{D}$, so apk is uniformly distributed over $\mathcal{R}$. Since $\mathcal{A}$ makes at most q queries to $\mathsf{H}_{\mathsf{dum}}$, and for each $\mathsf{H}_{\mathsf{dum}}$ query, the probability that $apk \in \mathcal{Q}_{\mathsf{apk}}$ is at most $q/|\mathcal{R}| \leq q/2^\kappa$. So $\Pr[\mathsf{KeyColl}] \leq q^2/2^\kappa$.
– MsgColl. During the $\mathsf{H}_{\mathsf{com}}(L)$ queries, m is uniformly sampled from $\{0,1\}^\kappa$. Since $\mathcal{A}$ makes at most q queries to $\mathsf{H}_{\mathsf{com}}$, and for each $\mathsf{H}_{\mathsf{com}}$ query, the probability that $m \in \mathcal{Q}_{\mathsf{msg}}$ is at most $q/2^\kappa$. So $\Pr[\mathsf{MsgColl}] \leq q^2/2^\kappa$.

Next we will show that $\mathcal{B}$ breaks the MS-UNF-2 security of MS, as long as $\mathcal{A}$ breaks the AS-UNF-3 security of ASvKA and the above bad events do not occur. Suppose that BadError, KeyColl, MsgColl do not occur, and $\mathcal{A}$'s output $(\sigma^*, L^*, \pi^*, apk^*)$ is a valid AS-UNF-3 forgery, which implies that

(i) $PK^* \subseteq LPK$,
(ii) $\mathsf{ASvKA.KAgVf}(PK^*, apk^*, \pi^*) = 1$,

(iii) $\mathsf{ASvKA.Vrfy}(L^*, apk^*, \sigma^*) = 1$,
(iv) $\exists i \in [n]$, s.t. $X_i^* = X^*$ and $(m_i^*, L^*, apk^*) \notin Q_{\mathsf{ASvKA}}$.

We need to show that $\mathcal{B}$'s output (PK'^*, m^*, σ'^*) is a valid MS-UNF-2 forgery, i.e., the following conditions hold:

(i′) $pk_{\mathsf{MS}}^* = X^* \in PK'^*$,
(ii′) $\mathsf{MS.Vrfy}(PK'^*, m^*, \sigma'^*) = 1$,
(iii′) $(m^*, PK'^*) \notin Q_{\mathsf{MS}}$, where Q_{MS} denotes the set Q in the MS-UNF-2 experiment (cf. Fig. 7). More precisely, Q_{MS} records all tuples (m, PK') that $\mathcal{B}$ has queried its own first round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m, PK')$ for some k and also queried its own R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this index k.

Clearly, (i′) follows from (iv) and (2), and (ii′) follows from (iii) and (2) by our construction of ASvKA.Vrfy (note that our ASvKA.Vrfy outputs 1 only if the underlying MS.Vrfy outputs 1). It remains to show that (iii′) holds.

Suppose towards a contradiction that (iii′) does not hold, i.e., $(m^*, PK'^*) \in Q_{\mathsf{MS}}$. Namely, $\mathcal{B}$ has queried its own first round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m^*, PK'^*)$ for some k, and also queried its own R-th round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this k. By our construction of $\mathcal{B}$ (as described above), it must hold that $\mathcal{A}$ has queried the first round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L, apk, \pi)$ for some (k, L, apk, π) in which $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m^*, PK'^*)$, and $\mathcal{A}$ has also queried the R-th round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ on this k in which $\mathcal{B}$ also queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this k.

We claim that for $\mathcal{A}$'s query $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L, apk, \pi)$ in which $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m^*, PK'^*)$, it must hold that $L = L^*$ and $apk = apk^*$. This is because in $\mathcal{A}$'s query $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L, apk, \pi)$, $\mathcal{B}$ should query $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m, PK')$ for

$$m := \mathsf{H_{com}}(L),\ PK' := \bigcup_{i \in \mathsf{Index}} \{X_i\} \cup \{dpk\}, \tag{3}$$

and it must hold $\mathsf{ASvKA.KAgVf}(PK, apk, \pi) = 1$ (o.w. $\mathcal{B}$ outputs $\perp$ directly). Since $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign}1}(k, m^*, PK'^*)$, we have $m^* = m$ and $PK'^* = PK'$.

- By (2), (3) and $m^* = m$, we have $m^* = \mathsf{H_{com}}(L^*) = \mathsf{H_{com}}(L) = m$. Since MsgColl does not occur, it follows that $L^* = L$.
- By (2), (3), $PK'^* = PK'$ and the facts that $\mathsf{ASvKA.KAgVf}(PK, apk, \pi) = 1$ and (ii) $\mathsf{ASvKA.KAgVf}(PK^*, apk^*, \pi^*) = 1$ hold, it follows that $apk = \sum_{i \in \mathsf{Index}} X_i + \mathsf{F}(dsk) = \sum_{i \in \mathsf{Index}^*} X_i^* + \mathsf{F}(dsk^*) = apk^*$.

So we have $L = L^*$ and $apk = apk^*$, i.e., $\mathcal{A}$ has queried the first round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}(k, L^*, apk^*, \pi)$ for some k and also queried the R-th round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ on this k. Consequently, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}1}$ query, for all $i \in [n]$ s.t. $X_i^* = X^*$, the tuples (k, m_i^*, L^*, apk^*) will be added to Q_1, and furthermore, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query, the tuples (m_i^*, L^*, apk^*) will be added to Q, which contradicts with (iv). This shows that (iii′) must hold.

Overall, (i′), (ii′) and (iii′) hold simultaneously, and thus $\mathcal{B}$'s output is a valid MS-UNF-2 forgery, as long as $\mathcal{A}$'s output is a valid AS-UNF-3 forgery and the above bad events do not occur. Hence $\mathsf{Adv}^{\text{MS-UNF-2}}_{\mathsf{MS},\mathcal{B}}(\kappa) \geq \mathsf{Adv}^{\text{AS-UNF-3}}_{\mathsf{ASvKA},\mathcal{A}}(\kappa) - (q + 2q^2)/2^\kappa$, which concludes the first part of the theorem.

Next, we prove the second part of the theorem. Let $\mathcal{A}$ be a PPT adversary against the AS-UNF-2 security of ASvKA, we aim to build a PPT adversary $\mathcal{B}$ against the MS-UNF-1 security of MS by invoking $\mathcal{A}$. The construction of $\mathcal{B}$ is exactly the same as above, since the security experiment is the same. Now we only need to show that $\mathcal{B}$'s output is a valid MS-UNF-1 forgery, as long as $\mathcal{A}$'s output is a valid AS-UNF-2 forgery (and the above bad events do not occur).The analysis is also similar as above, and the only exceptions are that the condition (iv) for $\mathcal{A}$'s output now becomes

(iv) $\exists i \in [n]$, s.t. $X_i^* = X^*$ and $(m_i^*, L^*, \cdot) \notin Q$,

and the condition (iii′) for $\mathcal{B}$'s output now becomes

(iii′) $(m^*, \cdot) \notin Q_{\mathsf{MS}}$.

We can also show the new (iii′) through a contradiction with the new (iv), similar as above. Consequently, we can get $\mathsf{Adv}^{\text{MS-UNF-1}}_{\mathsf{MS},\mathcal{B}}(\kappa) \geq \mathsf{Adv}^{\text{AS-UNF-2}}_{\mathsf{ASvKA},\mathcal{A}}(\kappa) - (q + 2q^2)/2^\kappa$, which concludes the second part of the theorem. □

C.2 Proof of Theorem 2 (Unforgeability of Construction 2)

The proof is very similar to the proof of the first part of Theorem 1, and below we will highlight the differences in gray boxes.

Let $\mathcal{A}$ be a PPT adversary against the AS-UNF-3 security of ASvKA. We will construct a PPT adversary $\mathcal{B}$ against the MS-UNF-1 security of MS. More precisely, $\mathcal{B}$ simulates the AS-UNF-3 security experiment for $\mathcal{A}$ in exactly the same way as that in the proof of Theorem 1, except that during the $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}(k, L, apk, \pi)$ queries, $\mathcal{B}$ calls its own first round signing oracle for $z^{\mathsf{MS}}_{k,1} \leftarrow \mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m} := (m, apk), PK')$ (instead of $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, m, PK')$), and computes $\tilde{z}_{k,1}$ using dsk by itself via $(\tilde{\mathsf{st}}_{k,1}, \tilde{z}_{k,1}) \leftarrow \mathsf{MS.MulSign}_1(dsk, \bar{m} := (m, apk), PK'; \lambda = 0)$ (instead of $\mathsf{MS.MulSign}_1(dsk, m, PK'; \lambda = 0)$).

Assuming that all bad events do not occur, then $\mathcal{A}$ outputs a forgery $(\sigma^*, L^*, \pi^*, apk^*)$ where $L^* = \{(X_i^*, m_i^*)\}_{i\in[n]}$. $\mathcal{B}$ parses $\sigma^* = (\sigma'^*, dpk^*)$, and computes

$$\begin{aligned} &PK^* := (X_i^*)_{i\in[n]}, m^* := \mathsf{H}_{\mathsf{com}}(L^*), \\ &\mathsf{Index}^* \leftarrow \mathsf{GetIndex}(PK^*), PK'^* := \textstyle\bigcup_{i\in\mathsf{Index}^*}\{X_i^*\} \cup \{dpk^*\}. \end{aligned} \tag{4}$$

Finally, $\mathcal{B}$ returns $(PK'^*, \bar{m}^* := (m^*, apk^*), \sigma'^*)$ as its own forgery.

Next we will show that $\mathcal{B}$ breaks the MS-UNF-1 security of MS, as long as $\mathcal{A}$ breaks the AS-UNF-3 security of ASvKA and the above bad events do not occur. Assume that $\mathcal{A}$'s output $(\sigma^*, L^*, \pi^*, apk^*)$ is a valid AS-UNF-3 forgery, which implies that

(i) $PK^* \subseteq LPK$,
(ii) $\mathsf{ASvKA.KAgVf}(PK^*, apk^*, \pi^*) = 1$,

(iii) $\mathsf{ASvKA.Vrfy}(L^*, apk^*, \sigma^*) = 1$,
(iv) $\exists i \in [n]$, s.t. $X_i^* = X^*$ and $(m_i^*, L^*, apk^*) \notin Q$.

We need to show that $\mathcal{B}$'s output $(PK'^*, \bar{m}^* = (m^*, apk^*), \sigma'^*)$ is a valid MS-UNF-1 forgery, i.e., the following conditions hold:

(i′) $pk_{\mathsf{MS}}^* = X^* \in PK'^*$,
(ii′) $\mathsf{MS.Vrfy}(PK'^*, \bar{m}^*, \sigma'^*) = 1$,
(iii′) $(\bar{m}^* = (m^*, apk^*), \cdot) \notin Q_{\mathsf{MS}}$, where Q_{MS} denotes the set Q in the MS-UNF-1 experiment (cf. Fig. 7), and records all tuples $(\bar{m} = (m, apk), PK')$ that $\mathcal{B}$ has queried its own first round $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m} = (m, apk), PK')$ for some k and also queried its own R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this k.

Similar to the proof of Theorem 1, (i′) follows from (iv) and (4), (ii′) follows from (iii) and (4), and it remains to show (iii′).

Suppose there exists a contradiction that (iii′) does not hold, i.e., $(\bar{m}^* = (m^*, apk^*), \cdot) \in Q_{\mathsf{MS}}$. Namely, $\mathcal{B}$ has queried its own first round oracle $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m}^* = (m^*, apk^*), \cdot)$ for some k, and also queried its own R-th round $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this k. By our construction of $\mathcal{B}$, it must hold that $\mathcal{A}$ has queried the first round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}(k, L, apk, \pi)$ for some (k, L, apk, π) in which $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m}^* = (m^*, apk^*), \cdot)$, and $\mathcal{A}$ has also queried the R-th round $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ on this k in which $\mathcal{B}$ also queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign}R}$ on this k.

With a similar analysis as that in the proof of Theorem 1, we can show that for $\mathcal{A}$'s query $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}(k, L, apk, \pi)$ in which $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m}^* = (m^*, apk^*), \cdot)$, it must hold that $L = L^*$ and $apk = apk^*$. This is because in $\mathcal{A}$'s $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}(k, L, apk, \pi)$ query, $\mathcal{B}$ should query $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m} = (m, apk), \cdot)$ for $m := \mathsf{H_{com}}(L)$.

Since $\mathcal{B}$ queries $\mathsf{MS}.\mathcal{O}_{\text{MulSign1}}(k, \bar{m}^* = (m^*, apk^*), \cdot)$, we have $m^* = m$ and $apk^* = apk$, and $m^* = \mathsf{H_{com}}(L^*) = \mathsf{H_{com}}(L) = m$ further implies $L^* = L$ assuming that MsgColl does not occur. So we have $L = L^*$ and $apk = apk^*$, i.e., $\mathcal{A}$ has queried the first round oracle $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}(k, L^*, apk^*, \pi)$ for some k and also queried the R-th round oracle $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ on this k. Then similar to the proof of Theorem 1, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign1}}$ query, for all $i \in [n]$ s.t. $X_i^* = X^*$, the tuples (k, m_i^*, L^*, apk^*) will be added to Q_1, and furthermore, during this $\mathsf{ASvKA}.\mathcal{O}_{\text{AggSign}R}$ query, the tuples (m_i^*, L^*, apk^*) will be added to Q, which contradicts with (iv). This shows that (iii′) must hold.

Hence, (i′), (ii′) and (iii′) hold simultaneously, and thus $\mathcal{B}$'s output is a valid MS-UNF-1 forgery. Consequently, we get $\mathsf{Adv}^{\text{MS-UNF-1}}_{\mathsf{MS},\mathcal{B}}(\kappa) \geq \mathsf{Adv}^{\text{AS-UNF-3}}_{\mathsf{ASvKA},\mathcal{A}}(\kappa) - (q + 2q^2)/2^\kappa$, which concludes the theorem. □

C.3 Proof of Theorem 4 (Full Privacy of PP-BAS-1)

The proof is essentially the same as the proof of Theorem 3, except that we use the Sign' algorithm instead of the standard BLS scheme. More precisely,

when answering $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries for $\mathcal{A}$, we compute σ as an aggregate-signature for $L_{S^*} = \{(X_i, m_i)\}_{i\in S^*}$, i.e., $\sigma := \tilde{z}\cdot\prod_{i\in\mathsf{Index}} z_i = \mathsf{H}_{\mathsf{sig}}(apk^*, m)^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} \mathsf{H}_{\mathsf{sig}}(apk^*, m)^{sk_i}$ in Game_0 and Game_1, and compute $\sigma := \mathsf{H}_{\mathsf{sig}}(apk^*, m)^{ask^*}$ in Game_2 and Game_3. Finally, in Game_3, we have $pk^* = apk^* := g^{ask^*}$ and $\sigma := \mathsf{H}_{\mathsf{sig}}(apk^*, m)^{ask^*}$ for $ask^* \leftarrow_{\$} \mathbb{Z}_p$, which are exactly the public key and signature of the Sign' algorithm defined above. With a similar analysis, we can get $\mathsf{Adv}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(\kappa) \leq q/2^{\kappa}$. □

C.4 Proof of Theorem 5 (Full Privacy of PP-SpeedyASvKA)

Let $\mathcal{A}$ be a PPT adversary against the AS-FullPriv of PP-SpeedyASvKA. Our proof proceeds with a sequence of games, where Game_0 is the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 0$ (cf. Fig. 3) and Game_3 is the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 1$. Let Ev_i be the event that $\mathcal{A}$ outputs $b^* = 1$ in Game_i.

- Game_0 : It is the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 0$. More precisely, let $(pk_i = (X_i, pop_i), sk_i = x_i)$ be the key pair of signer $i \in [n]$. In this game, upon receiving a signer set S^* from $\mathcal{A}$, we set pk^* as an aggregated public key for $PK_{S^*} = (X_i)_{i\in S^*}$, i.e., $apk^* := g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} X_i$, where $dsk^* := \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ with $\pi^* \leftarrow_{\$} \{0,1\}^{\kappa}$ and $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK_{S^*})$, and returns $pk^* := apk^*$ to $\mathcal{A}$. Then when answering $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries, we compute σ as an aggregate-signature for $L_{S^*} = \{(X_i, m_i)\}_{i\in S^*}$, i.e., $\sigma := (\hat{R}, z)$ with $\hat{R} := \bar{R}\cdot\bar{S}^b$ and $z := \sum_{i\in\mathsf{Index}} z_i + \tilde{z} = \sum_{i\in\mathsf{Index}}(r_i + b\cdot s_i + c\cdot sk_i) + c\cdot dsk^*$, where $\bar{R} := \prod_{j\in\mathsf{Index}} R_j$, $\bar{S} := \prod_{j\in\mathsf{Index}} S_j$ for $R_j := g^{r_j}, S_j := g^{s_j}$ with $r_j, s_j \leftarrow_{\$} \mathbb{Z}_p$, $b := \mathsf{H}_{\mathsf{non}}(apk^*, m, \bar{R}, \bar{S})$ for $m := \mathsf{H}_{\mathsf{com}}(L_{S^*})$, and $c := \mathsf{H}_{\mathsf{sig}}(apk^*, m, \hat{R})$, and returns $\sigma := (\hat{R}, z)$ to $\mathcal{A}$.
- Game_1: This game introduces abort conditions for $\mathcal{A}$'s random oracle queries to $\mathsf{H}_{\mathsf{dum}}$ and $\mathsf{H}_{\mathsf{non}}$. More precisely, if $\mathcal{A}$ makes a $\mathsf{H}_{\mathsf{dum}}$ query on $(\pi^*, \cdot)$, or makes a $\mathsf{H}_{\mathsf{non}}$ query on $(\cdot, \cdot, \bar{R}, \bar{S})$ for any $(\bar{R}, \bar{S})$ involved in the $\mathcal{O}_{\mathrm{CHL}}$ queries, Game_1 aborts. Since π^* is only used in the computation of $dsk^* = \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ in the game, by the random oracle model, $\mathcal{A}$ learns nothing about π^* and can hardly query $\mathsf{H}_{\mathsf{dum}}$ on $(\pi^*, \cdot)$. Besides, since $\mathcal{A}$ only obtains $\sigma = (\hat{R} = \bar{R}\cdot\bar{S}^b, z)$ from the $\mathcal{O}_{\mathrm{CHL}}$ queries, $\mathcal{A}$ does not get $\bar{R}, \bar{S}$ individually and can hardly query $\mathsf{H}_{\mathsf{non}}$ on $(\cdot, \cdot, \bar{R}, \bar{S})$. So we have $|\Pr[Ev_1] - \Pr[Ev_0]| \leq q^2/2^{\kappa}$, where q denotes the number of queries that $\mathcal{A}$ makes.
- Game_2 : This game is almost the same as Game_1, except how we compute $pk^* = apk^*$ and how we compute $\sigma = (\hat{R}, z)$ for $\mathcal{A}$'s $\mathcal{O}_{\mathrm{CHL}}(\{m_i\}_{i\in S^*})$ queries. More precisely, we first computes an *aggregated secret key* $ask^* := dsk^* + \sum_{i\in\mathsf{Index}} sk_i$, where $\mathsf{Index} \leftarrow \mathsf{GetIndex}(PK_{S^*})$. Then we use ask^* to compute $pk^* = apk^* := g^{ask^*}$, which is just a conceptual change since $apk^* = g^{ask^*} = g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} g^{sk_i} = g^{dsk^*} \cdot \prod_{i\in\mathsf{Index}} X_i$. Moreover, when answering $\mathcal{O}_{\mathrm{CHL}}$ queries, we compute $\sigma := (\hat{R}, z)$ with $\hat{R} := g^{\hat{r}}$ and $z := \hat{r} + c\cdot ask^*$, where $\hat{r} := \sum_{i\in\mathsf{Index}}(r_i + b\cdot s_i)$ for the $r_i, s_i \leftarrow_{\$} \mathbb{Z}_p$ and $b := \mathsf{H}_{\mathsf{non}}(apk^*, m, \bar{R}, \bar{S})$. This

change is also conceptual, since $\hat{R} = g^{\hat{r}} = \prod_{i \in \mathsf{Index}} g^{r_i} \cdot \prod_{i \in \mathsf{Index}} g^{b \cdot s_i} = \bar{R} \cdot \bar{S}^b$ and $z = \hat{r} + c \cdot ask^* = \sum_{i \in \mathsf{Index}}(r_i + b \cdot s_i) + c \cdot (dsk^* + \sum_{i \in \mathsf{Index}} sk_i) = \sum_{i \in \mathsf{Index}}(r_i + b \cdot s_i + c \cdot sk_i) + c \cdot dsk^*$. Thus $\Pr[Ev_2] = \Pr[Ev_1]$.

– Game_3 : This game is almost the same as Game_2, except that we sample $ask^* \leftarrow_\$ \mathbb{Z}_p$ uniformly (instead of computing $ask^* := dsk^* + \sum_{i \in \mathsf{Index}} sk_i$) and sample $\hat{r} \leftarrow_\$ \mathbb{Z}_p$ uniformly for the $\mathcal{O}_{\mathrm{CHL}}$ queries (instead of computing $\hat{r} := \sum_{i \in \mathsf{Index}}(r_i + b \cdot s_i)$).

  Note that by the changes in Game_1, $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{dum}}$ on $(\pi^*, \cdot)$ and never queries $\mathsf{H}_{\mathsf{non}}$ on $(\cdot, \cdot, \bar{R}, \bar{S})$, thus both $dsk^* = \mathsf{H}_{\mathsf{dum}}(\pi^*, PK_{S^*})$ and $b = \mathsf{H}_{\mathsf{non}}(apk^*, m, \bar{R}, \bar{S})$ are uniformly random over $\mathbb{Z}_p$ from $\mathcal{A}$'s view, and so are ask^* and $\hat{r}$. Consequently, the change is conceptual and $\Pr[Ev_3] = \Pr[Ev_2]$.

  Finally, we note that Game_3 is exactly the $\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}$ experiment with $b = 1$. This is because in Game_3, we have $pk^* = apk^* := g^{ask^*}$ and $\sigma := (\hat{R} = g^{\hat{r}}, z = \hat{r} + c \cdot ask^*)$ for $ask^* \leftarrow_\$ \mathbb{Z}_p$, $\hat{r} \leftarrow_\$ \mathbb{Z}_p$ and $c := \mathsf{H}_{\mathsf{sig}}(apk^*, m, \hat{R})$, which are exactly the public key and signature of the Schnorr scheme (cf. Subsect. B.2).

Overall, we have $\mathsf{Adv}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(\kappa) = \big|\Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(0) \Rightarrow 1] - \Pr[\mathsf{Exp}^{\mathsf{AS\text{-}FullPriv}}_{\mathsf{ASvKA},\mathcal{A},n}(1) \Rightarrow 1]\big| = \big|\Pr[Ev_0] - \Pr[Ev_3]\big| \leq q^2/2^\kappa$ and conclude the theorem. □

Table of Contents