Multi-Signatures for Ad-Hoc and Privacy-Preserving Group Signing

Anja Lehmann and Cavit Özbay(✉)

Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany
{anja.lehmann,cavit.oezbay}@hpi.de

Abstract. Multi-signatures allow to combine individual signatures from different signers on the same message into a short aggregated signature. Newer schemes further allow to aggregate the individual public keys, such that the combined signature gets verified against a short aggregated key. This makes them a versatile alternative to threshold or distributed signatures: the aggregated key can serve as group key, and signatures under that key can only be computed with the help of all signers. What makes multi-signatures even more attractive is their simple key management, as users can re-use the same secret key in several and ad-hoc formed groups. In that context, it will be desirable to not sacrifice privacy as soon as keys get re-used and ensure that users are not linkable across groups. In fact, when multi-signatures with key aggregation were proposed, it was claimed that aggregated keys hide the signers' identities or even the fact that it is a combined key at all. In our work, we show that none of the existing multi-signature schemes provide these privacy guarantees when keys get re-used in multiple groups. This is due to the fact that all known schemes deploy deterministic key aggregation. To overcome this limitation, we propose a new variant of *multi-signatures with probabilistic yet verifiable key aggregation*. We formally define the desirable privacy and unforgeability properties in the presence of key re-use. This also requires to adapt the unforgeability model to the group setting, and ensure that key-reuse does not weaken the expected guarantees. We present a simple BLS-based scheme that securely realizes our strong privacy and security guarantees. We also formalize and investigate the privacy that is possible by deterministic schemes, and prove that existing schemes provide the advertised privacy features as long as one public key remains secret.

1 Introduction

When cryptographic signatures and keys are used to protect high-value assets, it is often desirable to protect the access not only with a single, but with multiple keys. One of the most prominent applications of multi-key signing is public ledgers such as Bitcoin. Initially, a naive version of "multi-signatures" was proposed, where a single public key that protects an account gets replaced with a set of public keys, and transactions from that account require a set of respective signatures [1]. The key drawback of this approach is that signature and public

Q. Tang and V. Teague (Eds.): PKC 2024, LNCS 14601, pp. 196–228, 2024.
https://doi.org/10.1007/978-3-031-57718-5_7

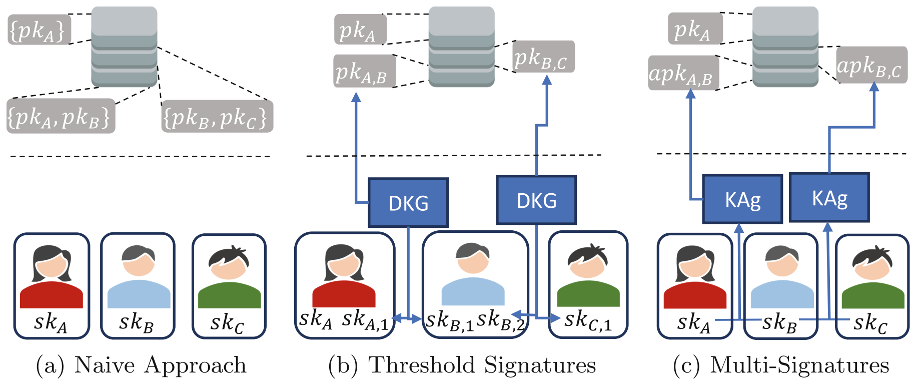


Fig. 1. Group signing approaches for the public ledger use case. Each figure presents a scenario which shows a public ledger with three users, Alice, Bob, and Charlie (from left to right) who hold accounts for user sets $\{Alice\}$, $\{Alice, Bob\}$, and $\{Bob, Charlie\}$. "DKG" and "KAg" stand for a distributed key generation protocol and the key aggregation process of multi-signatures with key aggregation, respectively.

key size, as well as signature verification costs, are linear in the group size. To improve the efficiency of the naive approach, the use of threshold signatures, such as Boldyreva's BLS-based scheme [11] or Schnorr-based FROST [24], was suggested in the Bitcoin standard [39].

Challenges in Key Management. While threshold signatures are more efficient than the naive approach, they come with challenges in key management – in particular in settings where a user is not part of a single signing group only, but wants to sign in *multiple* groups. Such a scenario is depicted in Fig. 1, where Fig. 1a employs the naive approach and Fig. 1b shows the implementation using threshold signatures.

In the naive solution, a group public key is simply the set of individual public keys. Thus, users could use the same secret key for multiple purposes and e.g., rely on a trusted hardware token to protect their individual long-term key. With threshold signatures, users need to manage *individual* key material for each group they are part of. For threshold signatures, this even requires a trusted dealer or an interactive key generation protocol.

In particular when dealing with end-users and not machines, convenient yet secure key management is crucial, and users should ideally be able to use a single long-term key for multiple accounts. In fact, this has very recently triggered a series of works, *multi-verse threshold signatures* (MTS), which allow signers to use a single long-term key to derive multiple "group" keys (or combined signatures) for arbitrary threshold structures [6,18,23]. However, all schemes require some level of interaction in the key aggregation algorithm and have a different focus from our work. We give a more detailed comparison in Sect. 1.3 and also include MTS in Fig. 2 for completeness.

Our work focuses on another – and conceptually much simpler – group signing solution: multi-signatures. They provide the long-term key support of the naive approach while enjoying the same compactness benefits as threshold signatures (but give up on the threshold setting).

Multi-signatures with Key Aggregation. Multi-signatures enable the efficient aggregation of individual signatures of n signers, each having an individual long-term key pair, on the *same* message m. The aggregated signature σ can be efficiently verified to be valid for all n signers. Secure multi-signatures have been constructed mostly from BLS, e.g., [36] and Schnorr signatures [10].

Originally, multi-signatures only considered the aggregation of signatures, but in 2019, Maxwell et al. [31] proposed a protocol that additionally combines the individual public keys into a short aggregated key apk. Verification of a multi-signature σ from n signers then requires only the aggregated public key apk. Computing apk needs to be done only once per group and several signatures of the same "group" can be verified using apk without needing the individual keys. After the initial work, several other schemes with key aggregation have been proposed [9,12,19,20,25,26,28,33,35]. These works were inspired by the public ledger application and have already seen real-world adoption. For instance, there is a proposal to include the multi-signature scheme MuSig2 of Nick et al. [33] into the Bitcoin standard. Another ledger technology, Cardano is also about to support MuSig2 [2]. Figure 1c shows how multi-signatures with key aggregation can be used in a public ledger to form group keys. Instead of creating dedicated secret keys for each group, signers can re-use a single key in multiple groups.

1.1 Ad-Hoc Group Signing with Long-Term Keys

The flexibility and convenience of multi-signatures bear the question of whether they already provide the optimal solution for applications that require compact group signing with long-term keys. We argue that this is not the case for existing multi-signatures, as unforgeability is not guaranteed to hold if keys are re-used, and any reasonable privacy is even impossible.

We start with a discussion of the desirable unforgeability and privacy guarantees in the context of flexible group signing with long-term keys, and explain what is satisfied by existing multi-signatures. For completeness, we also show in Fig. 2 how this compares to threshold signatures (that lack key re-use or simple verification) and the naive solution (allowing key re-use, but lacking compactness/efficiency).

From a practical perspective, we are interested in group signing solutions with the following properties:

Single long-term key: All users have an individual long-term key pair sk_i, pk_i that they generate (and use) autonomously. There is no trusted entity or joint protocol needed for key generation.

Ad-hoc groups: Users can use their single key to dynamically join *groups*, and do so repeatedly. A group is represented through an aggregated public key apk that is derived from a set of public keys $PK = \{pk_i\}$. Signatures that verify under that group key can only be derived from the individual signatures of all members.

Standard verification: Verification of the groups' signature must be (somewhat) compatible with standard verification (e.g. Schnorr, ECDSA, or BLS).

Efficiency: A group's signature and public key size, and verification time must be independent of the group size.

In terms of security, we require the unforgeability of the groups' signatures. The challenge hereby is to understand and formalize the impact of re-using the same individual secret key in different signing groups. For applications such as public ledgers, it will be important that individual signature contributions are strictly bound to the group they were intended for. Otherwise, a signature on $m :=$ "`Send $10 to Eve`" created by Alice to confirm the money transfer from her joint account with Bob, could be re-used to retrieve money from her individual account. Thus, for our group context, we required unforgeability:

Group unforgeability: It is infeasible to create a signature for message m and a group apk, when not all signers provided a signature for *m and apk*.

In existing works, this aspect is often not very explicit. While some target and realize group unforgeability [9,26,33–35,38], other works consider a weaker notion where signature contributions are not bound to a particular group [12, 19,20,25,28]. Thus, only some of the existing schemes provide the security that is necessary for public ledger applications, and developers must carefully read and understand the analyzed unforgeability guarantees. As a side contribution, our work conceptualizes the different unforgeability notions and also shows how weaker schemes can be lifted to achieve group unforgeability.

Note that group unforgeability also implicitly covers non-frameability, i.e., an honest user cannot be framed to be part of a group (i.e., apk) and participated in group signing, when she never did so. This is again crucial when dealing with long-term keys that will be clearly associated to individuals or legal entities, and gets re-used across groups. For the new type of multi-signatures we introduce here, this aspect requires more care and will therefore be discussed more explicitly throughout our work (yet formally is implied by group unforgeability).

The Need for Privacy. While the usage of multiple keys increases security, revealing information about this distribution might not be desired. For instance, revealing how many (and which) keys protect certain assets tells the adversary how many keys he has to compromise and possibly which are used more often and might be an easier or more lucrative target.

Most threshold signatures naturally hide who contributed to a particular signature, and multi-signatures with key aggregation gives rise to similar privacy features as they represent a signer group with a constant size public key. In the case of group signing with an *n-out-of-n* structure, the desire for privacy might be surprising at first. In the end, all signers must contribute to the signature, and one might think there is no need or chance for privacy at all.

However, one must distinguish between in- and outsiders here. Of course, the members of the group want to be aware who their co-signers are and there will be no privacy towards these insiders. To outsiders, i.e., non-group members, who only consume group keys and group signatures, this information is often not necessary. We believe that hiding information about the key structure should be the default, unless required otherwise.

In fact, Maxwell et al. [31] already advertised three privacy properties that multi-signatures with key aggregation allegedly provide: they do not leak any information about 1) the number of individual signers, 2) the identity of individual signers or 3) even whether a key and a corresponding signature are an aggregated key and signature, or standard ones. Previous works on aggregated signatures that implicitly used key aggregation also informally claimed similar features, e.g., that key aggregation hides the structure of the signers [3,37].

Having a single long-term key that is used in different groups now even amplifies the privacy need. Re-using the same key – which is desirable from a usability perspective – must not make the user's signature contributions traceable and linkable across groups. Thus, we set the privacy requirements as follows:

Outsider privacy: An aggregated public key apk (and corresponding signatures) leak no information about the underlying signers.

Unlinkability: The usage of the same long-term key in different groups cannot be linked. In fact, even when repeatedly signing with the same set of signers PK, users can decide to do so under the same apk or generate fresh apk's. Signatures under different apk's are fully unlinkable, i.e., they hide that they were generated by the same set of signers. This unlinkability holds for anyone that is not an insider in both groups.

No Privacy Yet. So far, no formal treatment of the already advertised privacy features for multi-signatures with key aggregation exist. However, even without a formal model, we can see that none of the existing schemes achieves any of the aforementioned privacy properties: all existing schemes have deterministic key aggregation. This makes any privacy properties of the aggregated keys (and associated signatures) impossible in a setting where the adversary knows the signers' individual public keys. Thus, privacy for such schemes can only hold in a model where at least some of the public keys (and their signatures) are considered to be secret. Given that a core benefit of multi-signatures is their flexible use, i.e., using the same long-term key for different group signing activities, assuming that public keys remain secret is clearly neither desirable nor realistic.

In summary, existing multi-signatures – despite being an attractive candidate for group signing – do not provide the privacy (and sometimes even security) guarantees that are needed in applications such as a public ledger. Therefore, our work addresses the following question:

How can we realize compact aggregated signatures that allow for key re-use across groups, yet guarantee strong privacy and security?

	Simple Vrfy	Compact	Key Re-Use	Group Unf	Privacy
Naive Appr. w. Key Prefix	✓	✗	✓	✓	✗
Threshold Signatures	✓	✓	✗	✓	✓
Multi-Signatures with KAg.	✓	✓	✓	✓✗	✗
Multiverse Threshold Sigs	✗	✓	✓	✗	✓✗
randBLS-1 (Section 5.1)	✓	✓	✓	✗	✓
randBLS-2 (Section 5.1)	✓	✓	✓	✓	✓

Fig. 2. Overview of approaches for group signing on public ledgers (see Sect. 1.3 for a more detailed discussion of multiverse threshold signatures). "✓", " ✓✗", and "✗" mean the requirement is satisfied, only satisfied by some existing works, and not satisfied, respectively. randBLS-1 satisfies our strongest privacy notion, i.e., aggregated signatures are fully indistinguishable from standard BLS signatures. randBLS-2 uses key-prefixing to achieve the desired group unforgeability, which comes for the prize of not being identical to standard signatures. Apart from leaking the fact that signatures are aggregated, it still provides the desired privacy guarantees.

1.2 Our Contributions

Our work answers that question by introducing a new variant of multi-signatures that is flexible enough to realize ad-hoc group signing with long-term keys and ensure strong privacy and unforgeability. We formally define the desired security guarantees in a setting where long-term keys get re-used in multiple groups and propose two simple BLS-based constructions that satisfy them. In more detail, our work makes the following contributions:

Multi-signatures with Verifiable Key Aggregation. We introduce a new variant of multi-signatures that comes with verifiable key aggregation (MSvKA). The core idea is to remove the requirement that key aggregation KAg is deterministic, yet keep a way to verify whether an aggregated public key apk belongs to a certain set PK of public keys. We realize that by defining key aggregation to also output a proof π along with the aggregated key. An additional algorithm VfKAg verifies whether apk and PK belong together – but requires π as input. This allows us to later have different security and privacy guarantees for insiders (knowing π and wanting to verify their co-signers) and outsiders (not knowing π).

Unforgeability Framework & Transformations. As our core motivation is the use of multi-signatures for ad-hoc group signing, we define unforgeability for this targeted group context and in the presence of key re-use. Our framework provides a set of definitions, depending on how explicit the user's group intent is supposed to be. The strongest notion in our framework (UNF-3) captures the desired group unforgeability for insiders. It guarantees that if a signer wanted to contribute to a multi-signature for a particular group (expressed via apk), then her signature share cannot be reused in any other context. Our weaker versions (UNF-1/2) allow for more flexibility in the aggregation of individual signature contributions, and will be the right choice when the group context is not known when the individual signatures are computed.

We also show simple transformations to lift schemes with weak unforgeability guarantees to their stronger versions and to translate existing unforgeability results for deterministic schemes into our setting. Further, all our unforgeability definitions also implicitly cover *non-frameability*, i.e., it is guaranteed that an adversary cannot frame an honest user (by producing a malicious proof π) for a group signature she never contributed to.

Privacy Framework. Our core contribution is a definitional framework that defines the privacy guarantees users can expect, despite repeatedly using the same secret key in different groups. We follow the initially advertised properties and formalize three privacy goals: Set Privacy (SetPriv), Membership Privacy (MemPriv), and Full Privacy (FullPriv). All properties guarantee that signers can repeatedly use their long-term secret key in multiple groups without becoming traceable (except to someone who is an insider in all groups). The difference between the definitions is in what the aggregated keys and signatures are supposed to hide beyond that. Our strongest goal FullPriv requires the aggregated values to be fully indistinguishable from standard ones, whereas our weakest guarantee (MemPriv) only focuses on hiding whether a particular user is a member of a group or not, but signatures and keys can leak the group size or the fact that they are aggregates. The stronger properties are harder to achieve and we believe all definitions to have their individual benefits and applications.

All goals are stated in a strong adversarial model, where the adversary can freely interact with all individual signers, knows all their public keys and can request multi-signatures and even be an insider in their groups. This is called the Known-Public-Key (KPK) model. We also show that no multi-signature with deterministic key aggregation can achieve any privacy properties in KPK model.

New BLS Multi-signature with Strong Privacy. As our impossibility result rules out privacy for all existing schemes, we propose a new and simple variant of the BLS multi-signatures from Boneh et al. [12] that turns key aggregation into a probabilistic algorithm. We prove this scheme – called randBLS-1 – to satisfy the strongest FullPriv-KPK privacy and UNF-1 security (in the plain public key model). Using our generic unforgeability transform via key-prefixing, we show how this scheme can be turned into a variant randBLS-2 that achieves UNF-3 security while satisfying MemPriv-KPK and SetPriv-KPK (but no FullPriv privacy anymore, as it leaks the fact that it is an aggregated key/signature).

Weaker Model & Analysis of Existing Schemes. While privacy in the KPK model is the goal we aim for and achieve with our new constructions, it is not achievable by any existing construction due to their deterministic key aggregation: If the adversary knows all public keys, it is trivial to check whether an aggregated public key corresponds to a given set of individual public keys or not. To analyze the privacy properties of existing schemes, we also define the weaker "All-but-one" AbOPK versions of our three privacy properties, where at least one public key must remain secret. This secrecy requirement also comprises all (multi-)

signatures ever generated under that key, and thus any privacy in that model should be interpreted with great care.

We then show that the most common multi-signatures based on BLS and Schnorr signatures achieve the strongest possible privacy guarantees for a deterministic scheme, which is FullPriv-AbOPK. We also translate their known unforgeability results into our framework and discuss how key-prefixing might boost their unforgeability, which results in a slight privacy loss for BLS-based schemes.

1.3 Related Work

We now discuss the related work, with the most related result being for threshold signatures. While multi-signatures might appear to be the special case of *n-out-of-n* threshold signatures, they are actually considerably different. In a standard threshold signature scheme, there exists a dedicated key generation phase for all n signers that then can generate signatures for that particular (sub)group. In a multi-signature, there does not exist a phase that fixes n signers, and using the exact same setup, signers can create multi-signatures for arbitrary signer sets.

Unforgeability Hierarchy for Threshold Signatures. A recent work on threshold schemes by Bellare et al. [8] investigates the different levels of unforgeability they can achieve. They propose stronger notions that a signer, knowing the co-signers when creating a signature, produces signature shares that cannot be used to create a signature for any other signer set. This is similar to the stronger unforgeability notion that already existed for multi-signature and which we capture as MSdKA-UNF-2 and MSvKA-UNF-2/3. Their work focuses solely on unforgeability, but does not consider privacy – which is the focus of our work.

Accountable Subgroup Multi-signatures (ASM). First defined by Micali et al. [32], ASM signatures are a special type of multi-signature that strictly binds a signature to a certain subset of signers. This notion is similar to MSvKA-UNF-2 in Sect. 3, group unforgeability, or the strongest definition of Bellare et al. [8]. Still, ASM schemes focus only on accountability, whereas our focus is on privacy guarantees (in combination with unforgeability). Furthermore, ASM takes the signer subset as an input to the verification algorithm. Although this choice allows flexible threshold structures, it leaves no hope for any privacy property.

Recently, Baldimtsi et al. proposed *subset multi-signatures* [7] which adds a *subset key aggregation* algorithm to the subgroup multi-signatures. Thus, the signature verification only takes an aggregated public key instead of a subset of signers. Aggregated keys can provide threshold-like access structures in a fixed group to require that a message has been signed by a particular subset of the group. Although subset multi-signatures improve ASM's by providing compact public keys and a simple verification algorithm, they sacrifice group unforgeability. Finally, neither of the two schemes achieves the ad-hoc groups that multi-signatures provide: they can only serve subgroups/subsets of a constant signing group which is defined prior to any signing process.

Threshold Signatures with Private Accountability. While traditional threshold signatures offer private signatures where the signatures created by different subgroups are indistinguishable, ASM offer accountability. Boneh and Komlo [13], TAPS, proposes a more flexible option between threshold signatures and ASM. TAPS considers a single signer group and applies a dedicated key generation for this group like threshold signatures. Their main goal is to have privacy for outsiders of the signer group (and to some extent for insiders too), and accountability (with the help of an insider). To do so, they introduce dedicated parties of a Combiner and Tracer with designated secret keys. Li et al. [30] distributes the trust to the combiner and the tracer entities by applying threshold structures and relying on a private blockchain and dedicated hardware extensions.

Our work and TAPS have different privacy concerns: TAPS focuses on hiding which *subset* of a fixed group with an *t-out-of-n* structure created a certain signature. Multi-signatures only operate in *n-out-of-n* setting, so this privacy notion loses its meaning in the context of multi-signatures. We study a privacy notion in a setting with long-term keys to be used in multiple groups. This is an aspect that multi-signatures naturally provide unlike traditional threshold signatures or TAPS. Our privacy notion aims at hiding the structure of these multiple signing groups and re-using keys without being linkable. Finally, neither [13] nor [30] has a simple verification algorithm suitable for replacing a system solely relying on Schnorr or BLS signatures.

Multiverse Threshold Signature (MTS). Baird et al. [6] defined MTS enabling threshold signing with a single long-term secret key and allowing users to create aggregated keys for arbitrary threshold structure *t-out-of-n* and an arbitrary group of signers. Lee [27] proposed another MTS construction that improves key aggregation and signature combining performance. MTS aims at threshold schemes and is more flexible in that respect than our multi-signatures. They do not formalize or aim at privacy properties though, which is the focus of our work. Further, they only aim at a rather weak form of unforgeability that is similar to our $\mathsf{UNF}\text{-}1$ notion, as it allows signature shares to be re-used in different contexts.

Concurrently, Garg et al. [23] and Das et al. [18] designed threshold signature schemes that work more classically for a fixed group of n parties, but support dynamic t values within that group by making the combine and verification algorithms depend on the threshold t. Thus, there is only a single group in which users have individual long-term keys (generated in a joined manner though), but the threshold in the combination and verification can vary. Although [23] informally discusses possible privacy extensions, this is not formalized and the given constructions do not prioritize privacy protection. Further, their unforgeability is again similar to our weakest notion only. While this is a desired feature for their setting, we aim at more restrictive signing, as our focus is on re-using the same key in *different* groups. Finally, we point out that none of these works meet our simplicity requirement as they use different verification algorithms than BLS signatures, making them unsuitable as a direct replacement in existing systems.

Signatures with Re-randomizable Keys. Fleischhacker et al. [22] proposed the concept of signatures with re-randomizable keys, where both the public and secret key allow for consistent re-randomization. These signatures naturally lend themselves for privacy-preserving applications and have sparked a line of research [4,15,17,21]. Building multi-signatures on top of signature schemes with re-randomizable keys could be an alternative way to achieve the functionality and privacy we aimed for, and might be an interesting direction for future work.

As demonstrated by our construction, it is not *necessary* though that the underlying key pairs allow for such re-randomization. From a practical perspective, our approach has two main advantages: 1) it requires a single re-randomization of the aggregated key instead of re-randomizing all public and secret keys for each group; 2) the long-term secret keys can be exposed through a plain sign-API, without the need of re-randomizing the secret key before each use.

2 Preliminaries

In this chapter, we state the notations and core building blocks we use throughout the paper. We also give a definition of multi-signatures with deterministic key aggregation and their known unforgeability models here.

Cyclic Groups and Pairing Groups. Throughout the paper we notate a prime order cyclic group generator GGen that outputs $\mathcal{G} = (\mathbb{G}, g, p)$ and bilinear pairing generator BGGen that outputs $\mathcal{BG} = (e, \mathbb{G}, \hat{\mathbb{G}}, g, \hat{g}, p)$ for the input security parameter. Formal definitions of these algorithms are in the full the paper [29].

We directly define the co-CDH assumption [12] on pairing groups.

Definition 1 (Co-CDH Assumption). *For all PPT adversaries $\mathcal{A}$ it holds that* $\Pr[\mathcal{BG} \leftarrow \mathsf{BGGen}(1^\lambda); a, b \leftarrow \mathbb{Z}_p; \hat{A} \leftarrow \mathcal{A}(\mathcal{BG}, g^a, g^b, \hat{g}^b) : \hat{A} = \hat{g}^{ab}] \leq \mathsf{negl}(\lambda)$

Traditional Signature Algorithms. Throughout the paper, we refer to traditional BLS and Schnorr signing algorithms using the following syntax. The public parameters of the BLS scheme and the Schnorr scheme contain the descriptions of a prime order bilinear group $\mathcal{BG}$ and a prime order group $\mathcal{G}$, respectively.

$\mathsf{BLSSign}(sk, m)$: Outputs $\mathsf{H}_0(m)^{sk}$.
$\mathsf{SchnorrSign}(sk, m)$: For $r \leftarrow \mathbb{Z}_p$, $R \leftarrow g^r$, and $c \leftarrow \mathsf{H}_0(R, g^{sk}, m)$, outputs $\sigma \leftarrow (z, R)$ where $z = r + c \cdot sk$.

Generalized Forking Lemma. The unforgeability proof of our new multi-signature requires the generalized forking lemma [5], and we refer the reader to the full paper [29] for its definition.

2.1 Multi-Signatures with Deterministic Key Aggregation

In this section, we define the existing variant for multi-signature with deterministic key aggregation (MSdKA) and two different versions of the unforgeability property that have been proposed in the literature.

There is actually no common and unified definition in the literature yet, e.g., works such as [12,16] do not make key aggregation or signature combination explicit at all. As both play a key role, we model them explicitly: key aggregation through function KAg and signature combination via the algorithm $\mathsf{Combine}$. Also, we use the name $\mathsf{MulSign}$ instead of Sign, since we later want to express compatibility between a standard signing and the multi-sign operation.

Definition 2 (MSdKA with deterministic key aggregation). *Multi-signature* MSdKA *is a tuple of algorithms* $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vf})$ *such that:*

$\mathsf{Pg}(1^\lambda) \rightarrow pp$: *Outputs public parameters pp for security parameter* 1^λ. *We only make pp explicit in key generation and assume it to be an implicit input to all other algorithms.*

$\mathsf{Kg}(pp) \rightarrow (sk, pk)$: *Probabilistic key generation, outputs key pair* (sk, pk).

$\mathsf{KAg}(PK) \rightarrow apk$: *Deterministic key aggregation, that on input a set of public keys* $PK = \{pk_i\}$, *outputs an aggregated public key apk.*

$\mathsf{MulSign}(sk_i, PK, m) \rightarrow s_i$: *(Possibly interactive) algorithm, that on input the secret key* sk_i, *message* m *and a set of public keys* $PK = \{pk_i\}$ *outputs a signature share* s_i.

$\mathsf{Combine}(PK, \{s_i\}_{pk_i \in PK}) \rightarrow \sigma$: *On input a set of public keys* $PK = \{pk_i\}$ *and set of shares* $\{s_i\}_{pk_i \in PK}$ *outputs a combined signature* σ *for* PK.

$\mathsf{Vf}(apk, \sigma, m) \rightarrow b$: *Verifies if* σ *is a valid signature on* m *for apk.*

A MSdKA must be correct, meaning that every combined multi-signature verifies correctly under the *apk* that belongs to the set of public keys the signature was created for. The correctness definition is available in the full version of the paper [29], and it also relies on the deterministic behaviour of the key aggregation.

Unforgeability Notions. For multi-signatures with deterministic key aggregation, there are two different variants for unforgeability. Both variants consider a single honest user with key pk^* that signs together with other users that are fully controlled by the adversary. The task of the adversary is to come up with a valid and non-trivial forgery (m, σ, PK), i.e., σ must verify correctly under $apk = \mathsf{KAg}(PK)$ with $pk^* \in PK$ that includes the honest signer. The difference in both variants is how they define a *trivial* forgery.

The first definition, denoted as $\mathsf{MSdKA\text{-}UNF\text{-}1}$ and first proposed by [36], only considers the message as authenticated information. That is re-using a signature obtained via $\mathcal{O}^{\mathtt{MulSign}}$ for some (m, PK) and turning it into a valid signature for (m, PK') with $PK \neq PK'$ is *not* considered a valid forgery. A stronger version is $\mathsf{MSdKA\text{-}UNF\text{-}2}$ (first used in [32]) which requires the tuple (m, PK) to be fresh. This ensures that each signature contribution of the honest signer is bound to the dedicated set PK it was intended for. Both games rely on the determinism of KAg and are shown in Fig. 3 and formally defined as follows:

Definition 3 (MSdKA Unforgeability-x). *A multi-signature scheme* Π *is* x-*unforgeable if for all PPT adversaries* $\mathcal{A}$ $\Pr[\mathsf{Exp}_{\Pi,\mathcal{A}}^{\mathsf{MSdKA\text{-}UNF\text{-}x}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *for the experiment from Fig. 3.*

```
$\mathsf{Exp}^{\mathsf{MSdKA\text{-}UNF\text{-}x}}_{\mathsf{MSdKA},\mathcal{A}}$
$pp \leftarrow \mathsf{Pg}(1^\lambda), (pk^*, sk^*) \leftarrow \mathsf{Kg}(pp), Q \leftarrow \emptyset$
$(\sigma, m, PK) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{MulSign}}}(pp, pk^*)$
return 1 if $\mathsf{Vf}(\mathsf{KAg}(PK), \sigma, m) = 1 \wedge$
$pk^* \in PK \ \wedge \ \mathsf{fresh}(m, PK, Q) = 1$
```

```
$\mathcal{O}^{\mathtt{MulSign}}(PK_i, m_i)$
if $pk^* \notin PK_i$ then return $\perp$
$Q \leftarrow Q \cup \{(m_i, PK_i)\}$
$s \leftarrow \mathsf{MulSign}(sk^*, PK_i, m_i)$
return $s$
```

UNF-x	UNF-1	UNF-2
$\mathsf{fresh}(m, PK, Q) = 1$ **if**	$(m, \cdot) \notin Q$	$(m, PK) \notin Q$

Fig. 3. Unforgeability for MSdKA schemes with deterministic key aggregation.

For MSdKA, MSdKA-UNF-2 corresponds to the group unforgeability notion we discussed in Sect. 1.1. It is easy to see that MSdKA-UNF-2 implies MSdKA-UNF-1. Furthermore, MSdKA-UNF-2 is strictly stronger than MSdKA-UNF-1, which immediately follows from the fact that there are schemes that satisfy the weaker but not the stronger notion, such as BLS multi-signatures [12]. One can use a signature share of some pk for message m to create a BLS multi-signature for any set PK where $pk \in PK$. In the full paper [29], we discuss the relation between these notions, and provide a generic transformation which lifts a MSdKA-UNF-1 scheme with certain properties to a MSdKA-UNF-2 scheme, using key-prefixing.

Unforgeability (so far) Requires Deterministic KAg. Requiring deterministic key-aggregation seems to be mainly an artifact of the absence of a formal treatment of unforgeability of signatures with aggregated public keys. Even though the verification algorithm takes an aggregated public key apk as input, the unforgeability of these schemes has still been analyzed in the traditional multi-signature model (with no key aggregation!), where the adversary is asked to output n individual public keys, out of which at least one must be honest. Verification of the adversary's forgery is then done for the aggregated public key that is re-computed from these keys, which requires that key aggregation is deterministic.

This reveals an interesting conflict of unforgeability and privacy: For unforgeability it will be desirable – and in fact necessary – to check whether an aggregated key apk belongs to a certain set PK. This motivates the new type of multi-signature that makes this verification explicit, and allows for probabilistic schemes that will be necessary for any reasonable level of privacy.

3 Multi-Signatures with Verifiable Key Aggregation

As motivated in our introduction (and formally shown in the next section), the current definition of multi-signatures with explicit *deterministic* key aggregation makes it impossible to achieve any form of privacy when the adversary knows all the individual public keys. We therefore introduce a more generic variant of multi-signatures where key aggregation can be probabilistic and that allows for explicit verification of whether an apk is valid for a particular PK.

We start by defining the new syntax and also show how every multi-signature scheme with deterministic key aggregation can be recast in this syntax. We then define different types of unforgeability, which now comes with one more flavour as there is no unique binding between *apk* and PK anymore. Our unforgeability model also ensures that the more flexible verification cannot be misused to frame honest users, i.e., to incorrectly claim that an honest user is part of a group and corresponding signatures she never contributed to. The introduction of our privacy framework for such multi-signatures is given in the following section.

3.1 Syntax and Correctness

The first change to remove the requirement of *deterministic* key aggregation is to make KAg (possibly) probabilistic. This would not be sufficient, though, as we want to keep accountability for insiders, i.e., we need to check whether an aggregated key *apk* belongs to a certain set of public keys PK.

To allow this, we change the definition of key aggregation KAg to not only output the aggregated key *apk* but also a proof π. We further add an algorithm VfKAg that allows to verify whether *apk* belongs to PK using π. Thus, any insider knowing all keys and π can still verify the correctness of the key (towards them there is no key privacy), whereas outsiders only knowing *apk* and PK can (depending on the scheme) not tell whether they belong together or not.

Having no unique mapping between *apk* and PK anymore, as well as having a proof π, also requires changes to $\mathsf{MulSign}$ and $\mathsf{Combine}$. Whereas the MSdKA version gives only PK as input to $\mathsf{MulSign}$ (as it uniquely defines *apk*), we will need to give the sign algorithm both PK and *apk*. We decided not to give π as input or enforce $\mathsf{MulSign}$ to check whether the key *apk* is correct. Instead, we assume signers to verify the correctness of the aggregated key explicitly (via VfKAg), and only run $\mathsf{MulSign}$ on verified keys. At some point in signing, this value π will be required though (in our concrete construction it will be the randomness used in key aggregation). This is happening in $\mathsf{Combine}$ which we give π as additional input. As $\mathsf{Combine}$ is run only once for a multi-signature, this choice has benefits for efficiency and practical security considerations, as it reduces the number of parties that need to keep and use π.

Definition 4 (MSvKA with verifiable key aggregation). *A multi-signature* MSvKA *is a tuple of algorithms* $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{VfKAg}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vf})$ *s.t.:*

$\mathsf{Pg}(1^\lambda) \rightarrow pp$**:** *On input security parameter* 1^λ*, it outputs public parameters pp.*

$\mathsf{Kg}(pp) \rightarrow (sk, pk)$**:** *Probabilistic key generation, outputs a key pair* (sk, pk)*.*

$\mathsf{KAg}(PK) \rightarrow (apk, \pi)$**:** *(Possibly probabilistic) key aggregation, that on input a set of public keys* $PK = \{pk_i\}$*, outputs an aggregated public key apk and a proof of aggregation* π*.*

$\mathsf{VfKAg}(PK, apk, \pi) \rightarrow b$**:** *Checks if* π *is a valid proof of aggregation for* PK *and apk and outputs the boolean result for it.*

$\mathsf{MulSign}(sk_i, PK, apk, m) \rightarrow s_i$**:** *(Possibly interactive) algorithm, that on input the secret key* sk_i*, message* m*, a set of public keys* $PK = \{pk_i\}$ *and aggregated key apk outputs a signature share* s_i*.*

$\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK}) \to \sigma$**:** *On input a set of public keys* $PK = \{pk_i\}$ *and set of shares* $\{s_i\}_{pk_i \in PK}$ *outputs a combined signature* σ *for* PK*.*
$\mathsf{Vf}(apk, \sigma, m) \to b$*: Verifies if* σ *is a valid signature on* m *for* apk*.*

The correctness definition of MSvKA is in the full paper [29] and now covers both VfKAg and Vf. Our new definition is a more general variant of multi-signatures, and any previous scheme with deterministic key aggregation can be turned into our more general variant as stated below. We will later show that this transformation also preserves the unforgeability.

Construction 1 (MSdKA to MSvKA Transformation). *For a* MSdKA *scheme* Π*, the* MSvKA *version with explicit key verification,* Π' *is defined as follows. The algorithms* $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{Vf})$ *of* Π' *are identical to* Π*. The remaining algorithms are:*

$\Pi'.\mathsf{KAg}(PK)$: Set $apk \leftarrow \Pi.\mathsf{KAg}(PK)$. Set $\pi = \perp$ and output (apk, π).
$\Pi'.\mathsf{VfKAg}(PK, apk, \pi)$: If $apk = \Pi.\mathsf{KAg}(PK) \neq \perp$ output 1, else 0.
$\Pi'.\mathsf{MulSign}(sk_i, PK, apk, m)$: Output $s_i \leftarrow \Pi.\mathsf{MulSign}(sk_i, PK, m)$.
$\Pi'.\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK})$: Outputs $\sigma \leftarrow \Pi.\mathsf{Combine}(PK, \{s_i\}_{pk_i \in PK})$.

3.2 Unforgeability Notions

We again define all unforgeability definitions through a single game, where only the freshness predicate differs depending on the unforgeability level. The main game structure is similar to the definitions for deterministic schemes (Sect. 2.1): the adversary gets a public key pk^* for an honest signer and oracle access to sk^* via the signing oracle $\mathcal{O}^{\mathtt{MulSign}}$. This oracle expects a message m, set of public keys PK – now along with the aggregated public key apk and a proof π which shows that apk and PK belong together and contain pk^*. After checking the validity of the provided proof, the oracle computes and returns the honest user's signature share. Further, when the adversary outputs his forgery, he must now provide the aggregated key apk along with a proof π for the claimed group PK. This is necessary as the winning condition will directly use apk when verifying the signature, and we need to check that apk belongs to the group PK that includes the honest signer, as otherwise "forging" would be trivial.

The strongest notion of our framework (MSvKA-UNF-3) guarantees that if a signer wanted to contribute to a multi-signature for a particular group (expressed via apk), then her signature share cannot be reused in any other context. This is what we referred to as group unforgeability.

When aiming at a threshold/quorum setting of signatures, MSvKA-UNF-3 might not be desired though: therein a number of signers will sign the same message, and as soon as the necessary amount exists, the aggregation into a group signature should be possible. (So the different thresholds/quorums will be represented by a set of possible apk's instead of a single on.) In such a scenario, the users are not aware of their "co-signers" upon creation of their individual signatures. To not exclude such applications, we also translate the classic notion

```
$\mathsf{Exp}^{\mathsf{MSvKA\text{-}UNF\text{-}x}}_{\Pi,\mathcal{A}}$
$pp \leftarrow \mathsf{Pg}(1^\lambda), (pk^*, sk^*) \leftarrow \mathsf{Kg}(pp), Q \leftarrow \emptyset$
$(\sigma, m, apk, \pi, PK) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{MulSign}}}(pp, pk^*)$
return $1$ if $\mathsf{Vf}(apk, \sigma, m) = 1$
  $\wedge\ \mathsf{VfKAg}(PK, apk, \pi) = 1\ \wedge\ pk^* \in PK$
  $\wedge\ \mathsf{fresh}(m, PK, apk, Q) = 1$
```

```
$\mathcal{O}^{\mathtt{MulSign}}(PK_i, apk_i, \pi_i, m_i)$
if $pk^* \notin PK_i \vee \mathsf{VfKAg}(PK_i, apk_i, \pi_i) \neq 1$
    then return $\perp$
$Q \leftarrow Q \cup \{(m_i, PK_i, apk_i)\}$
$s \leftarrow \mathsf{MulSign}(sk^*, PK_i, apk_i, m_i)$
return $s$
```

UNF-x	UNF-1	UNF-2	UNF-3
$\mathsf{fresh}(m, PK, apk, Q) = 1$ **if**	$(m, \cdot, \cdot) \notin Q$	$(m, PK, \cdot) \notin Q$	$(m, PK, apk) \notin Q$

Fig. 4. Unforgeability for MSvKA schemes with verifiable key aggregation.

of unforgeability (which was MSdKA-UNF-1 for deterministic schemes) to our setting, which yields the weakest definition denoted as MSvKA-UNF-1. Therein, there is no binding of a signature share or multi-signature to a particular *apk* but the property ensures that an adversary cannot create a message-signature pair that frames an honest user that has not signed the message.

For completeness, we also translate the existing unforgeability notion MSdKA -UNF-2 that binds signatures to the set of signers, i.e., PK (but not necessarily to *apk*) to our setting as MSvKA-UNF-2.

Non-frameability. Our unforgeability notion also guarantees non-frameability, i.e., an honest user cannot be framed (via VfKAg) for a signature she never contributed to. This aspect is modelled by the winning condition that comprises both verify algorithms, Vf (for signatures) and VfKAg (for the aggregated key). The adversary could always win if he is able to output a "forgery" σ for a key *apk* that he knows all secret keys for (then computing σ is trivial), yet he manages to produce a correct proof π s.t. $\mathsf{VfKAg}(PK, apk, \pi) = 1$ with $pk^* \in PK$. That is, $\mathcal{A}$ also wins if he produces a fraudulent proof π that frames the honest user.

Definition 5 (MSvKA Unforgeability-x). *A multi-signature scheme Π is* x*-unforgeable if for all PPT adversaries $\mathcal{A}$ in the experiment from Fig. 4 it holds that:* $\Pr[\mathsf{Exp}^{\mathsf{MSvKA\text{-}UNF\text{-}x}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Relations and Transformations. It is easy to see that MSvKA-UNF-2 is strictly stronger than MSvKA-UNF-1. For MSvKA schemes that have deterministic key aggregation (which is still allowed, but not enforced) MSvKA-UNF-2 and MSvKA-UNF-3 are equivalent, whereas MSvKA-UNF-3 is strictly stronger than MSvKA-UNF-2 for schemes with probabilistic KAg. We further show how known unforgeability results for deterministic schemes can be translated into our setting, and how MSvKA unforgeability can be lifted from UNF-1 to UNF-3. An overview of these results is given in Fig. 5.

Translating MSdKA *into* MSvKA *Unforgeability.* We start by showing that the transformation given in Construction 1 not only transforms the syntax but also preserves the unforgeability. The simple proof is available in [29].

Theorem 1. *If Π' is the transformation from Construction 1 applied on a* MSdKA *scheme Π, then the following holds:*

- *If Π is* MSdKA-UNF-*2 secure, then Π' is* MSvKA-UNF-*3 secure,*
- *If Π is* MSdKA-UNF-*1 secure, then Π' is* MSvKA-UNF-*1 secure.*

Lifting MSvKA-UNF-*1 to* MSvKA-UNF-*3 Security.* A natural question is how weaker versions can be lifted to the strongest one. An immediate idea is to sign (m, PK, apk) instead of m only. Intuitively, this scheme would be UNF-3 secure as any forgery for a message $m' \neq m$ for the same PK and apk would also become a forgery for the UNF-1 secure scheme for the message (m', PK, apk). However, this scheme is not useful, as it requires the knowledge of the signer set PK during signature verification. This would immediately destroy the efficiency and privacy features that comes with the key aggregation.

We resolve this by merely signing (m, apk), and requiring an additional property on the underlying key aggregation mechanism. Similar to the binding property of commitments, we call this property *key binding*. Key binding requires that it is hard to find two distinct signer sets PK and PK' for an aggregated key apk. This property is formally defined in the full paper [29]. We show that this additional assumption is sufficient for the simple key-prefixing transformation to lift an UNF-1 secure scheme to UNF-3 security:

Construction 2 (UNF-1 to UNF-3 Transformation). *Let Π be a* MSvKA-UNF-*1 scheme. Then, we define Π' as stated follows: the algorithms* (Pg, Kg, KAg, Combine) *of Π' are exactly as in Π, and the remaining algorithms are:*

$\Pi'.\mathsf{MulSign}(sk_i, PK, apk, m)$**:** Returns $s_i \leftarrow \Pi.\mathsf{MulSign}(sk_i, PK, apk, (apk, m))$.
$\Pi'.\mathsf{Vf}(apk, \sigma, m)$**:** Outputs $b \leftarrow \Pi.\mathsf{Vf}(apk, \sigma, (apk, m))$.

Theorem 2. *If Π is a multi-signature that is* MSvKA-UNF-*1 secure and key binding, then Π' from Construction 2 is* MSvKA-UNF-*3 secure.*

The simple proof is delegated to the full paper [29]. In a nutshell, if the MSvKA-UNF-3 adversary can provide a valid forgery on fresh (m, PK, apk), then we are able to find either a valid MSvKA-UNF-1 forgery on message (m, apk) or a valid collision (PK, PK', π, π') against the key binding property of the scheme.

4 Privacy Framework for MSvKA

Being equipped with a definition of multi-signatures that allows probabilistic key aggregation, we can now turn to the privacy properties that have already been advertised for such schemes and are necessary for our envisioned application of privacy-preserving group signing. In this section, we provide a formal privacy framework, roughly following what was claimed in [31].

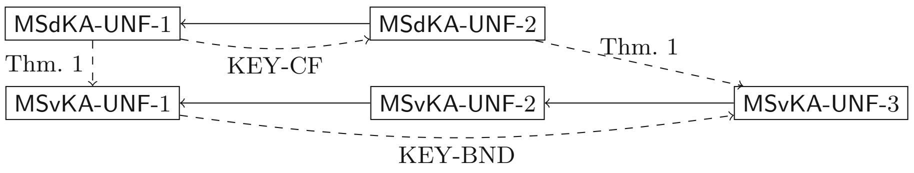


Fig. 5. Relation among unforgeability definitions. $A \overset{x}{\dashrightarrow} B$ means there is a generic construction of B from A relying on properties and/or theorems x. $A \rightarrow B$ means A implies B (any scheme has A, also has B). Finally, KEY-CF is nothing but the deterministic version of the key binding property (KEY-BND) which is available in the full paper [29].

Privacy Goals. We propose a hierarchy of definitions that aim at different strengths of privacy protection, which can be hiding the individual signers or, in the strongest variant, even hiding the fact that the signature and key are aggregated ones.

Full Privacy (FullPriv): One cannot tell whether a key and corresponding signature are a multi-signature with an aggregated key or stem from a standard signature algorithm.

Set Privacy (SetPriv): An aggregated key and corresponding signatures do not leak information about the underlying signer set (but can leak whether it is an aggregated one).

Membership Privacy (MemPriv): An aggregated key and corresponding signatures do not leak information about individual signers (but can leak the size of the group).

We show that FullPriv is the strongest notion and implies SetPriv, which in turn is strictly stronger than MemPriv. An advantage of FullPriv is that it allows for seamless integration into existing applications, as signatures and public keys have exactly the same form as standard ones. We decided to also formalize the weaker notions, as such strong FullPriv privacy might not be achievable (in particular for UNF-3-secure BLS signatures) or even desirable in some applications, e.g., when it should be clearly visible that the signature is a combined one. Both, SetPriv and MemPriv already capture the essential privacy guarantees we aimed for in the context of privacy-friendly group signing, and it will depend on the particular application which of the three properties is the "right" one.

Adversary Model. Given the ad-hoc nature of multi-signatures, the adversary should be able to interact with the individual signers freely, learn all their public keys, see their (multi)-signatures and even become insiders in some of her groups. All that must not allow the adversary to identify the user in groups he is not an insider in. This is what we capture as Known Public-keys (KPK) model for all three privacy properties. We will make this KPK-model explicit, as we will later also introduce a weaker "All-but-One" (AbOPK) model where at least one public key and associated signatures must remain secret. This weaker model is introduced to argue about the privacy of existing multi-signatures, as none of them satisfies the strong KPK version due to the deterministic key aggregation.

Unlinkability. Privacy in the KPK models presented in this section also captures unlinkability of individuals and groups. That is, if a signer re-uses the same key in several groups, the adversary cannot link her across the groups, unless he is an insider to all of them. Further, even the exact same group of signers can create different apk's and signatures when desired. Only users that know the corresponding proof(s) π can tell that they originate from the same group, whereas anyone not being privy of the proofs cannot tell whether two multi-signatures stem from the same group of signers or not. Such unlinkability is guaranteed by any of the three properties mentioned above, the difference is merely whether signatures/keys also hide the group size or the fact that they are an aggregated one.

4.1 Security Games

The goal of our security games is to capture the privacy guarantees towards an *outsider* of a particular (challenge) group, while giving the adversary as much knowledge and power of the individual signers and their participation in other signing groups. An outsider of a group defined through the aggregated key apk knows all individual public keys, and can see combined signatures for arbitrary messages of his choice under that apk – but he does not learn the aggregation proof π for apk nor actively participates in the group signing for this key.

While the adversary must be an outsider to the challenge group (there is no privacy to insiders), he can be an insider in other groups that have a partial or even full overlap with some of the signers of the challenge group. Such an insider might learn the aggregation proofs, see the signing protocol transcripts or even be an active signer in groups that have an overlap with the challenge group.

A typical way to model these insider capabilities of the adversary, is to provide oracle access to all honest entities and their secret keys. Here, this would require to define oracles for key aggregation, multi- (and individual) signing for all possible group and corruption settings.

Another approach is to be as generous as possible, and give the adversary all (secret) keys not strictly necessary to achieve the desired security property. The knowledge of these keys then enable the adversary to internally run all interactions with the honest parties himself. The advantage is that it avoids the need to define a multitude of oracles and keep track of the made queries, which keeps the games much simpler. It also directly highlights the keys or values that are crucial for the targeted property.

In our work we follow the later approach, and give the adversary not only the public keys but even the secret keys of all honest entities. Thus, the only oracle we need to provide in our games is for the challenge group.

We start with the presentation of our definitions for set privacy (SetPriv) and membership privacy (MemPriv). Both require the indistinguishability of two

```
$\mathsf{Exp}^{\mathsf{X}}_{\Pi,\mathcal{A}}(\lambda)$
$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, abort if $n \not> 0$
$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$
$(S_0, S_1) \leftarrow \mathcal{A}(SK\setminus\{sk_1\}, PK\setminus\{pk_1\})$, , abort if $1 \notin S_j$ for $j \in \{0,1\}$
abort if $|S_j \setminus S_{1-j}| \neq 1$ for $j \in \{0,1\}$
$(apk_b, \pi_b) \leftarrow \mathsf{KAg}(PK_{S_b})$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{Chl}}(\cdot)}(apk_b)$, return 1 if $b = b^*$

$\mathcal{O}^{\mathtt{Chl}}(m)$
$\Sigma \leftarrow \{\mathsf{MulSign}(sk_i, PK_{S_b}, apk_b, m)\}_{sk_i \in SK_{S_b}}$
return $\sigma \leftarrow \mathsf{Combine}(PK_{S_b}, \pi_b, \Sigma)$
```

Fig. 6. This is the game for our Set and Membership Privacy definitions ($\mathsf{X} \in \{\mathsf{MemPriv}, \mathsf{SetPriv}\}$). The additional part for $\mathsf{X} = \mathsf{MemPriv}$ is shown in dashed box. Grayed parts correspond to additions for the All-but-One-Public-Key model (AbOPK) model which will be introduced in Sect. 6.

aggregated public keys and associated signatures, and only differ in the restriction on the challenge groups. Thus, we capture both through the same game and only need to include an extra restriction when expressing the MemPriv version.

Set Privacy. Our SetPriv definition captures that an aggregate key and signature do not leak *any* information about the underlying signer group. This includes membership of individual signers but also the group size, both are required to remain hidden. The corresponding game runs in three stages:

In the first stage, the adversary just outputs a value n, which sets the number of individual keys in the system. The challenger internally generates all key pairs and returns all key pairs (SK, PK) to the adversary. The knowledge of all secret and public keys allows $\mathcal{A}$ to generate aggregated keys (and corresponding proofs) for arbitrary groups, as well as generate individual and combined signatures for these aggregated keys.

In the second stage, the adversary is asked to output two challenge sets S_0 and S_1, which are the indices of the honest signers generated earlier. The challenger chooses a random bit b and uses S_b to generate the challenge public key apk_b and proof π_b. The adversary receives apk_b (but not π_b) and gets access to a challenge oracle $\mathcal{O}^{\mathtt{Chl}}$ which returns multi-signatures for apk_b (and π_b) for arbitrary messages chosen by $\mathcal{A}$.

As the adversary knows the secret keys of all signers, it can create aggregated keys, signatures, and signing protocol transcripts for both S_0 and S_1 himself. This models that the challenge public key and signatures must be unlinkable to the keys and signatures that originate from the same set of signers.

Finally, the task of the adversary is to output a bit b^* and he wins if $b^* = b$, i.e., if he can guess to which group of signers the challenge public key and signatures belong. A scheme is said to satisfy our SetPriv definition if $\mathcal{A}$'s winning probability is negligibly better than guessing.

Membership Privacy. This property is defined identically to $\mathsf{SetPriv}$, but we no longer require the aggregated key and signature to hide the underlying group size. Thus, what membership privacy focuses on is hiding the identity of an individual signer within a group. To capture this (and not more), the definition must not allow an adversary to use any other difference between the two signer groups to infer information about a single user. We model this by asking the adversary to output two groups S_0 and S_1 that are identical, except for one user. That is, both groups must have the same size $|S_0| = |S_1| = k$ and $k-1$ members of the two sets must be the same. This condition is checked via the line in the dashed box when the adversary outputs its challenge sets. Apart from that extra check, the game and the winning condition are the same as in $\mathsf{SetPriv}$.

Definition 6 ($\{\mathsf{SetPriv}, \mathsf{MemPriv}\}$-$\mathsf{KPK}$)**.** *A* MSvKA *scheme Π has property* $\mathsf{X} \in \{\mathsf{SetPriv}, \mathsf{MemPriv}\}$ *in the* KPK *model, if for all PPT adversaries $\mathcal{A}$ in* $\mathsf{Exp}^{\mathsf{X}}_{\Pi,\mathcal{A}}$ *from Fig. 6 :* $|\Pr[\mathsf{Exp}^{\mathsf{X}}_{\Pi,\mathcal{A}}(\lambda) = 1] - 1/2| \leq \mathsf{negl}(\lambda)$.

Full Privacy. We now turn to our strongest privacy property. Intuitively, this property guarantees that if the multi-signature uses the same verification algorithm as their "regular signature" analogue, they do not even leak information about whether the signature and key are aggregated ones or not.

This is a bit tricky to define though, as the definition of multi-signature schemes does not contain an algorithm for creating "standard" signatures. Hence, we first need to consider an additional algorithm that captures such a signing procedure with individual keys, which we call Sign. For the majority of existing schemes, this sign algorithm will be the standard BLS or Schnorr algorithm. The keys for the standard sign algorithm are the ones generated via $\mathsf{MSvKA.Kg}$ and it also uses the same verification algorithm $\mathsf{MSvKA.Vf}$.

The detailed model is given in Definition 7 and starts by letting the adversary determine the number of signers for which the challenger then generates the individual keys. As in our previous models, $\mathcal{A}$ immediately gets the key pairs of the signers and these can be used to run KAg, $\mathsf{MulSign}$, or Sign before deciding upon his challenge group. The main difference is in the challenge. Here the adversary is only asked to output a single challenge group S^* and either receives the aggregated public key $pk^* = apk^*$ of that group (if $b = 0$) or a freshly chosen individual public key $pk^* = pk$ (if $b = 1$). Consequently, the challenge oracle now either returns an aggregated signature for apk^* or a standard signature under sk, depending on the challenge bit.

Definition 7 ($\mathsf{FullPriv}$-KPK)**.** *A* MSvKA *scheme Π is fully private for algorithm* Sign *in the* KPK *model if for all PPT adversaries $\mathcal{A}$ in* $\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\Pi,\mathcal{A}}$ *defined in Fig. 7 it holds that* $|\Pr[\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\Pi,\mathcal{A}}(\lambda) = 1] - 1/2| \leq \mathsf{negl}(\lambda)$.

4.2 Impossibility Results and Relations

Before we investigate the relations among our different definitions, we want to stress the following obvious – yet impactful – impossibility result:

```
$\mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi,\mathcal{A}}(\lambda)$
$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, abort if $n \not> 0$
$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$
$S^* \leftarrow \mathcal{A}(SK\backslash\{sk_1\}, PK\backslash\{pk_1\})$    abort if $1 \notin S^*$
if $b = 0$ then
    $(apk^*, \pi^*) \leftarrow \mathsf{KAg}(PK_{S^*}), pk^* \leftarrow apk^*$
if $b = 1$ : $(sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{Chl}}(\cdot)}(pk^*)$, return 1 if $b = b^*$

$\mathcal{O}^{\mathtt{Chl}}(m)$
if $b = 1$ : return $\sigma \leftarrow \mathsf{Sign}(sk, m)$
$\Sigma \leftarrow \{\mathsf{MulSign}(sk_i, PK_{S^*}, apk^*, m)\}_{sk_i \in SK_{S^*}}$
return $\sigma \leftarrow \mathsf{Combine}(PK_{S^*}, \pi^*, \Sigma)$
```

Fig. 7. Our FullPriv game capturing that aggregate signatures and keys are indistinguishable from standard ones. Grayed parts correspond to additions for the All-but-One-Public-Key model (AbOPK) model which will be introduced in Sect. 6.

Theorem 3. *No* MSvKA *schemes with deterministic key aggregation can satisfy the privacy properties* (FullPriv, SetPriv, MemPriv) *in the* KPK *model.*

Proof. As we will show that FullPriv and SetPriv are strictly stronger than MemPriv later in this section, we just prove that an MSvKA with deterministic key aggregation cannot satisfy MemPriv-KPK. We build a MemPriv-KPK adversary $\mathcal{A}$ as follows. $\mathcal{A}$ chooses $n = 3$, learns $\{(sk_i, pk_i)_{i=1,2,3}\}$, and submits the challenge sets $S_0 = \{1, 2\}$ and $S_1 = \{1, 3\}$. When $\mathcal{A}$ gets the challenge aggregated key apk_b, it checks whether $apk' = apk_b$ for $(apk', \_) := \mathsf{KAg}(\{pk_1, pk_2\})$. If the equality holds, the adversary outputs 0, and 1 otherwise. Due to the deterministic KAg, there is a unique aggregated key per signing group, and the adversary wins with probability 1. □

Note that the impossibility result from above immediately rules out the strongest privacy notions for all existing multi-signatures that follow the classic (deterministic) definition from Sect. 2.1, which we have shown to be translatable into the MSvKA framework in Construction 1.

Our adversarial model, granting the adversary access to all secret keys, is stronger than the real-world scenario we have envisioned. Thus, one may question whether the impossibility result for deterministic schemes is a consequence of this (too) strong model, and they could actually satisfy a relaxed yet equally meaningful security notion. It is easy to see that this is not the case, as the attack solely uses knowledge of the public keys and the fact that *apk* is deterministically derived from them. Thus, even a significantly weaker model, where we don't give the adversary any secret keys or even oracle access to them, could still not be satisfied by any deterministic scheme.

Relations Among Games. We now show that FullPriv is strictly stronger than SetPriv, which in turn is strictly stronger than MemPriv. We omit the dedicated mentioning of the KPK model here, as our results also hold for the AbOPK model that we introduce later in this work. An overview of these results is available in Fig. 8.

Theorem 4 (FullPriv $\Rightarrow$ SetPriv). *For any* MSvKA *scheme it holds that* FullPriv *implies – and is strictly stronger than –* SetPriv*.*

We need to prove two statements here: the first is that every FullPriv-secure scheme is also SetPriv-secure; the second is that there are schemes that achieve the SetPriv notion, but not FullPriv. The full proof is given in the full paper [29].

The first is intuitively rather straightforward. If the aggregated key and signatures are fully indistinguishable from standard signatures and keys, then the aggregated values can not leak any information about the contained individual signers or group size. The proof is given in the full version of the paper [29] and is slightly more elaborate, as both games have different structures and challenges.

To show that there are schemes that achieve the SetPriv but not FullPriv, we start with a scheme Π that satisfies both and transform it into Π' that loses the FullPriv property but is still SetPriv secure. The idea is rather simple: Let Π' be exactly as Π, with the only difference that the apk' returned from $\Pi'.\mathsf{KAg}$ adds an extra bit, i.e., $apk' = apk||1$. All algorithms of Π' that work with the aggregated key remove the last bit from apk' and then run identically as Π. This makes the aggregated public key clearly distinguishable from a standard one, so Π' loses the FullPriv property. As the extra bit is independent of the contained signers it does not give the adversary in the SetPriv game any advantage.

Theorem 5 (SetPriv $\Rightarrow$ MemPriv). *For any* MSvKA *scheme it holds that* SetPriv *implies – and is strictly stronger than –* MemPriv*.*

We again need to prove this in two steps. The first is proving that every SetPriv-secure scheme is also MemPriv-secure. This is straightforward, as both properties are expressed through the same security game, except that MemPriv makes an additional limitation on $\mathcal{A}$'s choice of challenge sets.

For the proof that MemPriv does not imply SetPriv, we must come up with a scheme that satisfies the former but not the latter. We start with a scheme Π that has both properties and change that into Π'. Π' behaves as Π, except that key aggregation now appends the set size to the aggregated key:

$$\Pi'.\mathsf{KAg}(PK) : (apk, \pi) \leftarrow \Pi.\mathsf{KAg}(PK); c := |PK| \ ; \ \mathbf{return}\ (apk' := apk||c, \pi)$$

The algorithms of Π' taking $apk' = apk||c$ as input, remove the group size c again and invoke the algorithms of Π on apk. In the MemPriv game, both challenge sets S_0 and S_1 must have the same size, and thus this leaked group size does not give the adversary any advantage, i.e., Π' is still MemPriv-secure. In the SetPriv game, the adversary can now win trivially by submitting two challenge groups of different sizes, regardless of the security of Π. We give a full proof of this idea in the full version of the paper [29].

5 Our Multi-Signature Constructions

We now present our multi-signatures, which are the first schemes that achieve privacy in the KPK model. Our first scheme (randBLS-1) is a simple modification

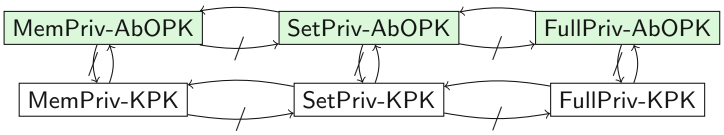


Fig. 8. Relation between our definitions under different models. $A \rightarrow B$ means A implies B (any scheme has property A, also has property B). $A \not\rightarrow B$ means A does not imply B (there exists a scheme s.t. has property A, but not property B). Properties with green boxes are achievable by BLS-dMS and MuSig multi-signature schemes. (Color figure online)

of the BLS multi-signature from Boneh et al. [12] and satisfies the strongest privacy guarantee FullPriv-KPK. Regarding unforgeability, it only achieves MSvKA-UNF-1 security which is the same security level as the original scheme. Using our UNF-1 to UNF-3 transformation from Sect. 3, we turn this into a variant (randBLS-2) which has the strongest unforgeability – but for the price of losing the FullPriv property, as this now requires key-prefixing (which is not considered standard in BLS signatures). This randBLS-2 scheme still satisfies the SetPriv-KPK and MemPriv-KPK properties, which again improves the state of the art for MSvKA-UNF-3 secure scheme. In fact, our randBLS-2 scheme still provides aggregated signatures that are indistinguishable from standard BLS signatures *with* public-key prefixing. Thus, in applications where such prefixing is done already – such as in public ledgers – this construction blends in perfectly.

5.1 Our randBLS-1 Construction

Existing constructions cannot achieve any privacy property in the KPK setting, due to their deterministic key aggregation. Thus, the main task is to turn key aggregation into a probabilistic algorithm that allows the verifiability of a group only with the knowledge of a dedicated proof π. We achieve this by a simple twist in the BLS-based multi-signature scheme of Boneh et al. [12].

Our scheme has identical key generation and signature verification algorithms as the original BLS signature and BLS multi-signatures. The main difference is that we include a random r in the exponent hash $\mathsf{H}_1(pk, PK, r)$ that is used for key aggregation and signature combination. This random r is our proof π, and verifying an aggregated key is recomputing the product using the same ("random") hash. We further move the exponentiation with this hash from MulSign to Combine, which is due to our choice to only include π in Combine but not in MulSign (which was mainly for efficiency purposes and has no impact on the achievable unforgeability notion).

Construction 3 (randBLS-1)**.** *Our first construction* randBLS-1 *uses a bilinear group generator* BGGen*, two hash functions* $\mathsf{H}_0 : \{0,1\}^* \rightarrow \mathbb{G}$ *and* $\mathsf{H}_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ *and is defined as follows:*

$\mathsf{Pg}(1^\lambda)$**:** Return $(e, \mathbb{G}, \hat{\mathbb{G}}, g, \hat{g}, p) \leftarrow \mathsf{BGGen}(1^\lambda)$.
$\mathsf{Kg}(pp)$**:** Return $sk \leftarrow \mathbb{Z}_p^*$, $pk \leftarrow \hat{g}^{sk}$.

$\mathsf{KAg}(PK)$**:** $r \leftarrow \{0,1\}^\lambda$, $apk \leftarrow \prod_{pk_i \in PK} pk_i^{\mathsf{H}_1(pk_i, PK, r)}$. Return $(apk, \pi := r)$.
$\mathsf{VfKAg}(PK, apk, \pi)$**:** $apk' \leftarrow \prod_{pk_i \in PK} pk_i^{\mathsf{H}_1(pk_i, PK, \pi)}$. Return $apk = apk'$.
$\mathsf{MulSign}(sk_i, PK, apk, m)$**:** Return $s_i \leftarrow \mathsf{H}_0(m)^{sk_i}$.
$\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK})$**:** Return $\sigma \leftarrow \prod_{pk_i \in PK} s_i^{\mathsf{H}_1(pk_i, PK, \pi)}$.
$\mathsf{Vf}(apk, \sigma, m)$**:** Return $e(\sigma, \hat{g}) = e(\mathsf{H}_0(m), apk)$.

Unforgeability of randBLS-1. The unforgeability of BLS multi-signatures in the plain public-key model relies on the non-linear mapping in the key aggregation algorithm, and we must ensure that our randomization technique does not introduce a weakness. As the involved randomness in key aggregation is chosen by the adversary in our unforgeability games, he can try to perform attacks similar to rogue-key attacks to form an aggregated key which is independent of the challenge public key pk^*. This is not the case for our scheme, as we use the involved randomness as an additional input to the random oracle H_1. Even if the adversary chooses the randomness maliciously, he still cannot manipulate H_1's output to have a specific algebraic form to cancel pk^* out from the aggregated keys. Regarding the different unforgeability levels, MSvKA-UNF-1 is the best we can hope for, as MulSign is entirely independent of PK and apk.

Theorem 6 (Unforgeability of randBLS-1). *The* randBLS-1 *multi-signature scheme in Construction 3 is* MSvKA-UNF-1 *secure in the ROM for q_{H_0} and q_{H_1} oracle queries for random oracles H_0 and H_1 if the co-CDH assumption, Definition 1, holds and $p > 8q_{H_1}/\mu(\lambda)$.*

Proof (Sketch). Our proof closely follows the unforgeability proof of deterministic BLS multi-signatures from [12]. We aim to build a co-CDH adversary using an efficient forger $\mathcal{A}$. Let us recap a proof strategy for original BLS signatures first [14]: Given a co-CDH problem instance $(A, B, \hat{B})$, the proof simulates an unforgeability game for $pk^* \leftarrow \hat{B}$. We set a $\mathsf{H}_0(m)$ query to $\mathsf{H}_0(m) \leftarrow A$, and hope to get a forgery σ for the message m. For a valid forgery, we have $e(A, pk^*) = e(\sigma, \hat{g})$, so σ is the solution for the given co-CDH instance.

The challenge in the multi-signature case is that the adversary does not output a forgery for pk^* directly, but an aggregated signature for some apk that contains pk^*. Thus, we need a way to get a valid signature for some $(pk^*)^c$ where c is a non-zero value (that will be known to the reduction). Our proof strategy is as follows: We first build an algorithm $\mathcal{B}$ which behaves as the co-CDH solver that uses the original BLS forger we explained above. This algorithm $\mathcal{B}$ only plays an intermediate role, and its task is to get an aggregated signature and key for some message in the simulated unforgeability game that embeds the co-CDH challenge. Our VfKAg() algorithm ensures that the set PK and the aggregated key apk that are chosen by the adversary satisfy that $pk^* \in PK$ and $apk = \prod_{pk_i \in PK} pk_i^{a_i}$ for $a_i = \mathsf{H}_1(pk_i, PK, \pi)$. We then use the Generalized Forking Lemma on the algorithm $\mathcal{B}$ to get two forgeries σ and σ' for aggregated keys apk and apk' such that $apk/apk' = (pk^*)^c$ for some non-zero c value. In particular, the forking lemma gives us two forgeries for the same set PK and the proof π. Further, all a_i values above are set to the same value except the $a_i = \mathsf{H}_1(pk_i, PK, \pi)$ for

$pk_i = pk^*$. For $pk_i = pk^*$, the random oracle is programmed to another value in the second forgery, so when we compute apk/apk' all values except pk^* cancel out. Finally, as we know that $e(A, apk) = e(\sigma, \hat{g})$ and $e(A, apk') = e(\sigma', \hat{g})$ holds, we also know that $e(A, apk/apk') = e(\sigma/\sigma', \hat{g})$, and thus the solution for the given co-CDH instance is $(\sigma/\sigma')^{1/c}$. The formal proof is in the full paper [29].

Privacy of $\mathsf{randBLS}$-1. Our $\mathsf{randBLS}$-1 construction achieves the strongest privacy notion $\mathsf{FullPriv}$ in the KPK model, i.e., produces indistinguishable aggregated keys and signatures from standard ones generated with $\mathsf{BLSSign}$. This immediately implies that the notions of $\mathsf{MemPriv}$ and $\mathsf{SetPriv}$ are satisfied too.

Theorem 7 (Privacy of $\mathsf{randBLS}$-1). *The* $\mathsf{randBLS}$-1 *scheme in Construction 3 is* $\mathsf{FullPriv}$-KPK *secure for* $\mathsf{Sign} = \mathsf{BLSSign}$ *in ROM for* H_1 *as a random oracle.*

Proof (Sketch). In the $\mathsf{FullPriv}$-KPK game, the adversary receives either an aggregated key and signatures (if $b = 0$) or a freshly sampled public key with the corresponding signatures (if $b = 1$) and must not be able to determine b. We prove this property through a series of games, where we end in a game where $\mathcal{A}$ always receives a freshly chosen (standard) key and signatures thereof.

First, we show that for an aggregated key apk^* in our scheme, we can generate a corresponding *aggregated secret key* $ask^* := \sum_{pk_i \in PK_{S^*}} sk_i \cdot \mathsf{H}_1(pk_i, PK_{S^*}, \pi^*)$ that we can use to answer the signing queries using the plain BLS signing algorithm $\mathsf{BLSSign}(ask^*, \cdot)$, instead of aggregating individual signatures. Subsequently, we show through several steps that this ask^* value is indistinguishable from a freshly sampled secret key. It is easy to see that ask^* (and apk^*) are uniformly random to a party who does not know the corresponding proof π^*, due to the random oracle involved in the computation. We then show that the proof π^* remains unknown to $\mathcal{A}$ even after outputting the challenge key and the corresponding signatures that implicitly contain π^*, which again stems from the random oracle property of H_1. In the final game, we replace (ask^*, apk^*) with a freshly sampled secret key, i.e., the game behaves identically for $b = 0$ and $b = 1$ and thus cannot reveal any information about the challenge bit b. The full proof is given in the full version of the paper [29].

5.2 Our randBLS-2 Construction

We now show how we can increase unforgeability to MSvKA-UNF-3, for the price of reducing privacy to $\mathsf{SetPriv}$-KPK. This is done by simply applying the generic UNF-1 to UNF-3 transformation (from Construction 2) to $\mathsf{randBLS}$-1. That is, the $\mathsf{MulSign}$ algorithm of our second scheme $\mathsf{randBLS}$-2 now strictly binds each signature to the intended apk by including the key in the hash.

Construction 4 ($\mathsf{randBLS}$-2). *The* $\mathsf{randBLS}$-2 *is identical to* $\mathsf{randBLS}$-1, *except for the following two algorithms:*

$\mathsf{MulSign}(sk_i, PK, apk, m)$**:** Return $s \leftarrow \mathsf{H}_0(apk, m)^{sk_i}$
$\mathsf{Vf}(apk, \sigma, m)$**:** Return $e(\sigma, \hat{g}) = e(\mathsf{H}_0(apk, m), apk)$.

Using Theorem 2, we conclude that the randBLS-2 scheme is MSvKA-UNF-3 secure if randBLS-1 is MSvKA-UNF-1 secure and key binding. The former was shown in Theorem 6, and thus what remains to be shown is that randBLS-1 is key binding, i.e., an adversary cannot come up with two sets $PK \neq PK'$ that map to the same aggregated key apk.

The simple proof of Theorem 8 is given in the full paper [29] and mainly relies on the fact that each aggregated key is sampled uniformly random by the random choice of r and H_1 being a random oracle, which ensures that collisions occur with negligible probability only. We can then conclude the Corollary 1.

Theorem 8. *The* randBLS-1 *scheme in Construction 3 is key-binding in the ROM for* H_1 *as a random oracle.*

Corollary 1 (Unforgeability of randBLS-2). *The* randBLS-2 *scheme in Construction 4 is* MSvKA-*UNF-3 secure in the ROM for* H_0 *and* H_1 *as random oracles if the co-CDH assumption holds and* $2^\lambda > 8q_H/\mu(\lambda)$ *for* q_H *oracle queries.*

As we now let Vf check the pairing for $\mathsf{H}_0(apk, m)$ we can no longer achieve FullPriv for the *standard* BLSSign algorithm anymore (which uses $\mathsf{H}_0(m)$). However, we still use the FullPriv game as a simple way to prove that the SetPriv-KPK is satisfied: we have shown in Theorem 4 that SetPriv is implied if a scheme satisfies FullPriv-KPK against *some* Sign algorithm. Thus, we simply prove FullPriv-KPK for the modified verification equation (but run for an individual signature, not an aggregated one) and then conclude SetPriv (which in turn implies MemPriv) from there. The proof of the Theorem 9 is the same as the proof of Theorem 7 except that we use a key-prefixed version of BLSSign instead of the standard one, and given in the full paper [29]. Using Theorem 4, we conclude the Corollary 2.

Theorem 9. *The* randBLS-2 *scheme in Construction 4 is* FullPriv-KPK *for the signing algorithm* $\mathsf{Sign}(sk, m) := \mathsf{H}_0(\hat{g}^{sk}, m)^{sk}$ *in ROM for the random oracle* H_1*.*

Corollary 2 (Privacy of randBLS-2). *The* randBLS-2 *scheme in Construction 4 is* SetPriv-KPK *(and thus* MemPriv-KPK*) secure if* H_1 *is a random oracle.*

6 Weaker Privacy and Analysis of Existing Constructions

We have already shown that all existing multi-signature schemes cannot satisfy the privacy properties defined in Sect. 4, due to their deterministic key aggregation. As this is in contrast to what has been claimed, we investigate the weaker privacy guarantees that such deterministic systems can provide. We start by introducing our weaker "All-but-One-PK" (AbOPK) model that adapts the FullPriv, SetPriv and MemPriv definitions by restricting the adversary to knowing all individual public keys, except of one. We then analyze the most common BLS- and Schnorr-based multi-signatures and prove that they do achieve privacy in this weaker model. An overview of the security and privacy of the existing schemes and our new constructions is given in Fig. 9.

6.1 Privacy Model for Deterministic Schemes: AbOPK

As stated in Theorem 3, none of our privacy definitions is achievable when key aggregation is deterministic: the adversary can win each game trivially by comparing the aggregated key(s) he can compute for the challenge set(s) with the key he received from the challenger. To define the desirable privacy properties in such a deterministic environment, we need to capture and exclude the trivial yet inherent attacks imposed by this setting. This requires two changes:

– The adversary must not know all public keys anymore, i.e., at least one public key must remain secret.
– The adversary must not be able to receive aggregate keys or multi-signatures of the challenge group(s) outside the challenge oracle. This immediately rules out any unlinkability guarantees.

We realize both in our "All-but-One-PK" (AbOPK) model that we can apply to all three privacy games. In the AbOPK version of our games, the adversary will no longer receive all public keys, but all but one. Without loss of generality, we set pk_1 to be the unknown key. We also follow the modeling choice from Sect. 4 and generously give the adversary the secret key to every public key it is allowed to know. Thus, when generating all key pairs in our games, denoted as (SK, PK), the adversary now only gets $(SK \setminus \{sk_1\}, PK \setminus \{pk_1\})$. The AbOPK limitation has a strong impact on the overall privacy guarantees, as the adversary in all our games is now prevented from interacting with the holder of the "secret" public key at all.

Definition 8 (AbOPK Models). *A* MSvKA *scheme Π has the property* $\mathsf{X} \in \{\mathsf{SetPriv}, \mathsf{MemPriv}, \mathsf{FullPriv}\}$ *in the* AbOPK *model, if for all PPT adversaries $\mathcal{A}$ it holds that* $|\Pr[\mathsf{Exp}^{\mathsf{X\text{-}AbOPK}}_{\Pi,\mathcal{A}}(\lambda) = 1] - 1/2| \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{X\text{-}AbOPK}}_{\Pi,\mathcal{A}}$ *is defined in Figs. 6 and 7.*

Impact of AbOPK. The most obvious change in our AbOPK model is that the adversary no longer receives all keys (SK, PK) in the second stage of our games, but only gets $(SK \setminus \{sk_1\}, PK \setminus \{pk_1\})$.

Another impact of our AbOPK models is that the challenge sets, that the adversary must output in all three games, become more restrictive. As the entire privacy now relies on the secrecy of pk_1, this public key must of course be part of the challenge sets – otherwise the adversary knows again all keys that will be aggregated into apk^*. This is modeled through additional abort conditions which check that 1 (as key index) is contained in all challenge sets. Putting both limitations – on the challenge sets and keys – together, this means that the adversary can never run any key aggregation or multi-signature algorithms for a *particular* challenge group (i.e., either S_0 and S_1 in the MemPriv, SetPriv games or S^* in the FullPriv game). Thus, also schemes that create aggregate keys and signatures that are linkable for each group PK can satisfy these weaker AbOPK privacy notions.

Obviously, the stronger KPK models are indeed strictly stronger than the weaker AbOPK ones. For completeness, we prove the relations in the full paper [29].

Scheme	Unforg.	FullPriv	SetPriv	MemPriv
BLS-dMS [12]	UNF-1	AbOPK	AbOPK	AbOPK
BLS-dMS + KeyPrefix (Cons. 2)	UNF-3	–/(AbOPK*)	AbOPK	AbOPK
MuSig [31]	UNF-3	AbOPK*	AbOPK	AbOPK
Our Work (randBLS-1)	UNF-1	KPK	KPK	KPK
Our Work (randBLS-2)	UNF-3	–/(KPK*)	KPK	KPK

Fig. 9. Comparison of existing and our new multi-signatures, regarding their unforgeability and privacy. *Note that there is a difference how "standard" the key-prefixing is that all UNF-3-secure schemes require. For Schnorr signatures, such prefixing, i.e., including the public key in the message hash, is often considered to be the standard – which is why MuSig achieves both UNF-3 and FullPriv security. For BLS signatures, key-prefixing is less standard, and thus all UNF-3 secure schemes that use such prefixing immediately lose the FullPriv privacy. If one considers including $(a)pk$ in the hash as standard for BLS too, then both UNF-3 secure BLS schemes also satisfy FullPriv.

6.2 Analysis of BLS and Schnorr Multi-Signatures

In this section, we summarize the analysis of the most common multi-signatures schemes, BLS-based by Boneh et al. [12] and Schnorr-based MuSig by Maxwell et al. [31]. We also note that our theorems and proofs can be easily adapted to other Schnorr multi-signatures with the same key aggregation technique (e.g., [25,33]).

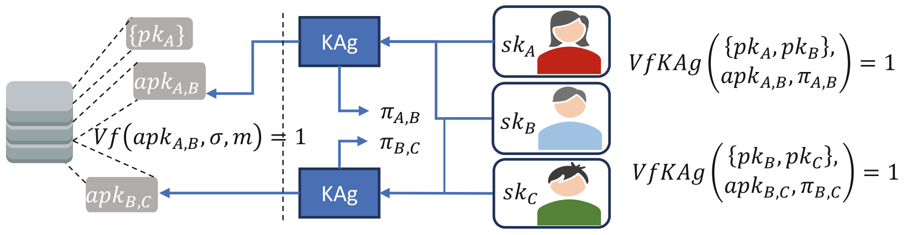
We show that these schemes satisfy FullPriv-AbOPK privacy, i.e., produce multi-signatures and aggregate keys that are indistinguishable from individual ones (derived via SchnorrSign and BLSSign, respectively), if at least one public key remains unknown to the adversary. The formal theorems related to these properties and the proof of those theorems are in the full paper [29].

Originally, BLS-based multi-signature was proven to be MSdKA-UNF-1 secure and MuSig to be MSdKA-UNF-2 secure. Using our transformations from Sect. 3, we can conclude that the BLS-based scheme is MSvKA-UNF-1 secure and MuSig is MSvKA-UNF-3 secure. The difference in the unforgeability level originates from the fact, that Schnorr-schemes widely use the key-prefixed version, even in the stand-alone setting, and we followed that choice in our analysis. The BLS-based scheme needs to apply key-prefixing, which then allows us to prove UNF-3 security via our transformation in Theorem 2. This incurs a slight loss in privacy, as this scheme then only satisfies SetPriv-AbOPK.

7 Applications

We conclude by showing how our MSvKA schemes can be applied in the use case that was one of the main motivations of our work: public ledgers. Another application we envision is privacy-preserving authentication from hardware tokens, and refer to the full version of the paper [29] for a discussion of how this can be realized from our multi-signatures.

For the public ledger use case, we follow the simple example from the introduction. That is, we have three users – Alice, Bob, and Carol – each having their



{pk_A}
apk_A,B
Vf(apk_A,B, σ, m) = 1
apk_B,C
KAg
π_A,B
π_B,C
KAg
sk_A
sk_B
sk_C
VfKAg({pk_A, pk_B}, apk_A,B, π_A,B) = 1
VfKAg({pk_B, pk_C}, apk_B,C, π_B,C) = 1

Fig. 10. Group signing using MSvKA in a public ledger.

own individual key pair, where one account is owned by Alice alone, the second by Alice and Bob, and the third by Bob and Carol. There are in fact different ways how our scheme could be used in this setting. The solution we sketch here is and is summarized in Fig. 10 is, to us, the most natural one.

Individual Setup. Every user generates their individual key pair, as $\mathsf{Kg}(pp) \rightarrow (sk_X, pk_X)$ for $X \in \{A, B, C\}$. We assume all public keys are publicly known to everyone, and the individual account by Alice is associated with pk_A.

Group Setup. When Alice wants to generate a joint account with Bob, either of them can trigger the key aggregation. Assuming this is done by Alice, she runs:

$$\mathsf{KAg}(PK_{A,B}) \rightarrow (apk_{A,B}, \pi_{A,B})$$

where $PK_{A,B} := \{pk_A, pk_B\}$. Alice locally stores the tuple $(Alice/Bob, apk_{A,B}, \pi_{A,B})$ and sends $(apk_{A,B}, \pi_{A,B})$ to Bob. Bob, upon receiving the tuple, now verifies that this aggregated public key is correctly formed by running:

$$\mathsf{VfKAg}(PK_{A,B}, apk_{A,B}, \pi_{A,B}) \rightarrow b$$

If $b = 1$, Bob stores $(Alice/Bob, apk_{A,B}, \pi_{A,B})$ and puts $apk_{A,B}$ on the ledger for the shared account.

For the shared account of Bob and Carol, the same procedure is used, and the resulting $apk_{B,C}$ gets associated with their account. At the end, Alice keeps $(Alice/Bob, apk_{A,B}, \pi_{A,B})$, Bob has $(Alice/Bob, apk_{A,B}, \pi_{A,B})$, $(Bob/Carol, apk_{B,C}, \pi_{B,C})$ and Carol stores $(Bob/Carol, apk_{B,C}, \pi_{B,C})$.

One could also assign one member of each group as the designated combiner, then only this party needs to keep the associated π. The values established for each group are not security-critical – their leakage have no impact on the guaranteed unforgeability, only on privacy. There is no privacy towards anyone knowing the proof π, so the value should be treated with some care but clearly does not have to be protected at the same level as the user's long-term secret key. What is important here is that each party stores the approved apk for each group, as this will be needed as trusted input for each signature contribution.

Group Signing. Whenever Alice and Bob want to make a transaction from their shared account protected with $apk_{A,B}$, both parties need to provide their signature contribution using their long-term key. Let us assume that Alice initiated the transaction. She computes her share as:

$$\mathsf{MulSign}(sk_A, PK_{A,B}, apk_{A,B}, m) \rightarrow s_A$$

Alice informs Bob about this request and sends him (s_A, m). If Bob agrees, he first computes his share and then combines both:

$$\mathsf{MulSign}(sk_B, PK_{A,B}, apk_{A,B}, m) \rightarrow s_B; \quad \mathsf{Combine}(PK_{A,B}, \pi_{A,B}, \{s_A, s_B\}) \rightarrow \sigma$$

Bob then sends (m, σ) to the ledger to release the transaction. Signatures for Bob/Carol are done analogously, and signatures for pk_A are just standard signatures.

Verification. Anyone on the ledger can verify the correctness of the transaction, e.g., for $apk_{A,B}$ by running $\mathsf{Vf}(apk_{A,B}, \sigma, m) \rightarrow b$. This verification does not require any individual keys or even knowledge of who the underlying signers are.

Privacy Guarantees. If a scheme with FullPriv privacy, such as randBLS-1 is used, then no one (except Alice and Bob) can even notice that $apk_{A,B}$ is an aggregated public key. If a scheme with Set/MemPriv privacy, such as randBLS-2 is used, then an outsider can see that this is an account controlled by multiple parties. However, the members of that group and the size of the group are still fully hidden. Our notion guarantees that privacy to anyone who is not part of the group, i.e., even a malicious Carol knowing all public keys and having a joint account with Bob cannot recognize that Bob also controls $apk_{A,B}$.

If key-prefixing in BLS is also done for standard signatures on the ledger, then aggregated keys and signatures from randBLS-2 are again fully indistinguishable from standard ones.

Unforgeability and Non-frameability Guarantees. For this application, our strongest notion UNF-3 is needed, which is satisfied only by randBLS-2. It ensures that all signature contributions are strictly bound to the context. That is, e.g., signature shares for $apk_{A,B}$ can neither be used for Alice's private account nor for the shared account Bob has with Carol.

Our schemes hide all information about their signers by default but also come with dedicated key verification. It is therefore important that this verification cannot be misused for framing attacks. Assume that Alice and Bob make a dubious transaction from their account $apk_{A,B}$, and later want to claim that this was actually done by Carol and Dave, e.g., by coming up with a proof π^* such that $\mathsf{VfKAg}(\{pk_C, pk_D\}, apk_{A,B}, \pi^*) = 1$. As non-frameability is guaranteed by all unforgeability notions, this is infeasible for every secure MSvKA scheme.

Acknowledgements. This research was partially funded by the HPI Research School on Data Science and Engineering. It was also supported by the German Federal Ministry of Education and Research (BMBF) through funding of the ATLAS project under reference number 16KISA037.

References

1. Bitcoin wiki. https://en.bitcoin.it/wiki/Multi-signature
2. Iohk musig2 implementation. https://github.com/input-output-hk/musig2
3. Ambrosin, M., Conti, M., Ibrahim, A., Neven, G., Sadeghi, A.R., Schunter, M.: SANA: secure and scalable aggregate network attestation. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016)
4. Backes, M., Hanzlik, L., Kluczniak, K., Scneider, J.: Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys (2018), report Number: 191
5. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Proceedings of the 15th ACM conference on Computer and communications security - CCS '08, p. 449. Alexandria, Virginia, USA (2008)
6. Baird, L., et al.: Threshold signatures in the multiverse. In: 2023 IEEE Symposium on Security and Privacy (SP) (2023)
7. Baldimtsi, F., et al.: Subset-optimized BLS Multi-signature with Key Aggregation (2023). https://eprint.iacr.org/2023/498, report Number: 498
8. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Advances in Cryptology – CRYPTO 2022 (2022)
9. Bellare, M., Dai, W.: Chain reductions for multi-signatures and the hbms scheme. In: Advances in Cryptology – ASIACRYPT 2021 (2021)
10. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006 (2006)
11. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_3
12. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
13. Boneh, D., Komlo, C.: Threshold signatures with private accountability. In: Advances in Cryptology – CRYPTO 2022 (2022). https://doi.org/10.1007/978-3-031-15985-5_19
14. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Advances in Cryptology - ASIACRYPT 2001 (2001)
15. Celi, S., Griffy, S., Hanzlik, L., Kempner, O.P., Slamanig, D.: SoK: Signatures With Randomizable Keys (2023), https://eprint.iacr.org/2023/1524, publication info: Preprint
16. Crites, E., Komlo, C., Maller, M.: How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures (2021). https://eprint.iacr.org/2021/1375, report Number: 1375

17. Das, P., Faust, S., Loss, J.: A Formal Treatment of Deterministic Wallets. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, pp. 651–668. Association for Computing Machinery, New York, November 2019. https://doi.org/10.1145/3319535.3354236, https://dl.acm.org/doi/10.1145/3319535.3354236
18. Das, S., Camacho, P., Xiang, Z., Nieto, J., Bunz, B., Ren, L.: Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold (2023). https://eprint.iacr.org/2023/598, report Number: 598
19. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the Security of Two-Round Multi-Signatures. In: 2019 IEEE Symposium on Security and Privacy (SP) (2019)
20. Drijvers, M., Gorbunov, S., Neven, G., Wee, H.: Pixel: multi-signatures for consensus. In: Proceedings of the 29th USENIX Conference on Security Symposium (2020)
21. Eaton, E., Lepoint, T., Wood, C.A.: Security Analysis of Signature Schemes with Key Blinding (2023). https://eprint.iacr.org/2023/380, report Number: 380
22. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient Unlinkable Sanitizable Signatures from Signatures with Re-Randomizable Keys (2015), report Number: 395
23. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hinTS: Threshold Signatures with Silent Setup (2023). https://eprint.iacr.org/2023/567, report Number: 567
24. Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Selected Areas in Cryptography (2021)
25. Kılınç Alper, H., Burdges, J.: Two-round trip schnorr multi-signatures via delinearized witnesses. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 157–188. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_7
26. Le, D.P., Yang, G., Ghorbani, A.: DDH-based Multisignatures with Public Key Aggregation (2019). https://eprint.iacr.org/2019/771, report Number: 771
27. Lee, K.: Decentralized Threshold Signatures for Blockchains with Non-Interactive and Transparent Setup (2023). https://eprint.iacr.org/2023/1206, report Number: 1206
28. Lee, K., Kim, H.: Two-round multi-signatures from okamoto signatures. Mathematics **11**(14) (2023). https://doi.org/10.3390/math11143223
29. Lehmann, A., Özbay, C.: Multi-signatures for ad-hoc and privacy-preserving group signing. Cryptology ePrint Archive, Paper 2023/1884 (2023). https://eprint.iacr.org/2023/1884. https://eprint.iacr.org/2023/1884
30. Li, M., Zhang, M., Wang, Q., Ding, H., Meng, W., Zhu, L., Zhang, Z., Lin, X.: Decentralized Threshold Signatures with Dynamically Private Accountability, August 2023. http://arxiv.org/abs/2304.07937. arXiv:2304.07937 [cs]
31. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Des. Codes Crypt. **87**(9), 2139–2164 (2019)
32. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: Proceedings of the 8th ACM conference on Computer and Communications Security (2001)
33. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: simple two-round schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 189–221. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_8
34. Pan, J., Wagner, B.: Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In: Advances in Cryptology – EUROCRYPT 2023 (2023)

35. Pan, S., Chan, K.Y., Cui, H., Yuen, T.H.: Multi-signatures for ECDSA and its applications in blockchain. In: Information Security and Privacy (2022)
36. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_13
37. Syta, E., Tamas, I., Visher, D., Wolinsky, D.I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., Ford, B.: Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning, May 2016. arXiv:1503.08768 [cs]
38. Tessaro, S., Zhu, C.: Threshold and multi-signature schemes from linear hash functions. In: Advances in Cryptology – EUROCRYPT 2023 (2023), https://doi.org/10.1007/978-3-031-30589-4_22
39. Wuille, P., Nick, J., Towns, A.: Validation of taproot scripts. https://github.com/bitcoin/bips/blob/e918b50731397872ad2922a1b08a5a4cd1d6d546/bip-0342.mediawiki