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

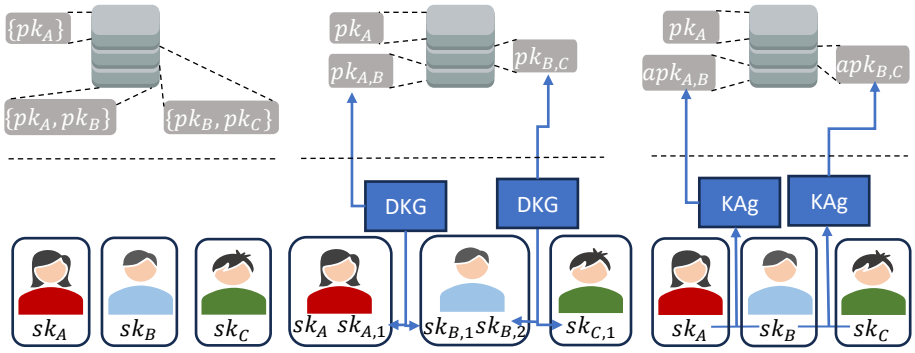
Anja Lehmann and Cavit Özbay

Hasso-Plattner-Institute, University of Potsdam
{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



(a) Naive Approach (b) Threshold Signatures (c) Multi-Signatures

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 Figure 1, where Figure 1a employs the naive approach and Figure 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, 23, 18]. 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 Section 1.3 and also include MTS in Table 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. [30] 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 [12, 19, 20, 25, 28, 9, 26, 32, 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. [32] 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 Table 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 $m :=$ "`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 require a strong variant of 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, 32, 35, 34, 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. [30] 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 need for privacy. Re-using the same key – which will be 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 having a formal model, it is easy to 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:

	Simple Vrfy	Compact	Key Re-Use	Group Unf	Privacy
Naive Approach 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 Sec. 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.

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

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 what we call the Known-Public-Key (KPK) model. We also show that no multi-signature with deterministic key aggregation can achieve any of the privacy properties in this strong 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 guarantee 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. [31], 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 Section 3, group unforgeability, or the strongest definition of Bellare et al. [8]. Still, ASM schemes focus only on accountability, whereas our focus is on achievable 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, aims to find a more flexible option for both privacy and accountability features of threshold signatures. Similar to threshold signatures, TAPS considers a single signer group and applies a dedicated key generation for this group. 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 each holding designated secret keys.

Li et al. (DeTAPS, [29]) aimed to distribute the trust to the combiner and the tracer entities by applying threshold structures. It results in requirements such as running a private blockchain and dedicated hardware extensions for combiners.

Our privacy concerns differ from those of TAPS: the focus in TAPS is on hiding which *subset* of a fixed group with an *t-out-of-n* structure created a certain signature. Given that multi-signatures exclusively operate in *n-out-of-n* setting, the TAPS privacy notion loses its meaning in the context of multi-signatures. The privacy notion studied in our work assumes a setting where long-term keys exist and can be used in multiple groups, which is an aspect which multi-signatures naturally provide in contrast to traditional threshold signatures or TAPS. Our privacy notion aims at hiding the structure of these multiple signing groups and re-using keys without being linkabl.

In terms of constructions, neither [13] nor [29] has a simple verification algorithm suitable for replacing a system solely relying on Schnorr or BLS signatures.

Threshold signatures with long-term secret keys. There are several recent works that extend threshold signatures to the setting where users have long-term keys, just as in our work, and also rely on BLS-based (threshold) signatures. Baird et al. [6] defined *multiverse threshold signature* (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 as the main contribution. 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 UNF-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 four works meet our simplicity requirement as they use a different verification algorithm 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 [21, 4, 17, 15]. 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. Full definitions of these algorithms are in Appendix A.

For simplicity, we directly define the co-CDH assumption [12] on pairing groups that we will use.

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

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 contain the description of a prime order bilinear group $\mathcal{BG}$ and the public parameters of a Schnorr scheme contains the description of a prime order group $\mathcal{G}$.

$\mathsf{BLSSign}(sk, m)$: 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], for completeness we give the generalized forking lemma in Appendix A.

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 [16, 12] 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). *A multi-signature MSdKA is a tuple of algorithms $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vf})$ such that:*

$\mathsf{Pg}(1^\lambda) \to pp$**:** *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.*

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

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

$\mathsf{MulSign}(sk_i, PK, m) \to s_i$**:** *(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.*

$\mathsf{Combine}(PK, \{s_i\}_{pk_i \in PK}) \to \sigma$**:** *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.*

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 in Definition 11 in Appendix B.1, and it also relies on the deterministic behaviour of the key aggregation.

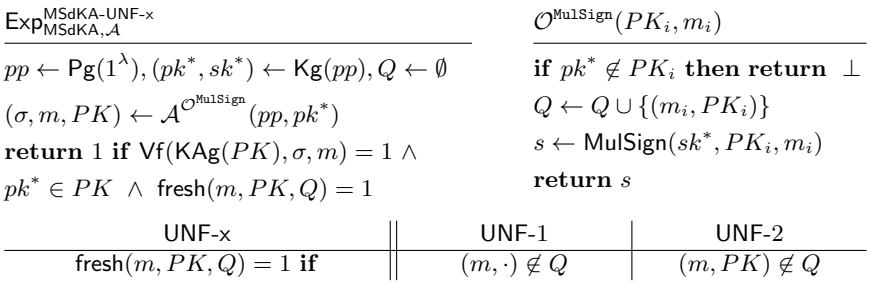


Fig. 3: Unforgeability for MSdKA schemes with deterministic key aggregation.

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 MSdKA-UNF-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 MSdKA-UNF-2 (first used in [31]) 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 Figure 3 and formally defined as follows:

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

For MSdKA, MSdKA-UNF-2 corresponds to the group unforgeability notion we discussed in Section 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 Appendix C, 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 recomputed 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 MulSign and Combine. Whereas the MSdKA version gives only PK as input to 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 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 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 Combine which we give π as additional input. As 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.:*

$\mathsf{Pg}(1^\lambda) \to pp$**:** *On input security parameter* 1^λ*, it outputs public parameters* pp*.*
$\mathsf{Kg}(pp) \to (sk, pk)$**:** *Probabilistic key generation, outputs a key pair* (sk, pk)*.*
$\mathsf{KAg}(PK) \to (apk, \pi)$**:** *(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*.*
$\mathsf{VfKAg}(PK, apk, \pi) \to b$**:** *Checks if* π *is a valid proof of aggregation for* PK *and* apk *and outputs the boolean result for it.*
$\mathsf{MulSign}(sk_i, PK, apk, m) \to s_i$**:** *(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*.*

For completeness, the correctness of MSvKA is defined as in Appendix B.2 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) *Let* Π *be a* MSdKA *multi signature with deterministic key aggregation, then* Π' *defined as follows is the general* MSvKA *version with explicit key verification. The algorithms* $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{Vf})$ *of* Π' *are the same as in* Π*, and 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 (Sec. 2.1):

the adversary gets a public key pk^* for an honest signer and oracle access to sk^* via the signing oracle $\mathcal{O}^{\texttt{MulSign}}$. 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 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 Figure 4 it holds that:* $\Pr\left[\mathsf{Exp}^{\mathsf{MSvKA\text{-}UNF\text{-}x}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

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

$\mathsf{Exp}^{\mathsf{MSvKA\text{-}UNF\text{-}x}}_{\Pi,\mathcal{A}}$

$pp \leftarrow \mathsf{Pg}(1^\lambda), (pk^*, sk^*) \leftarrow \mathsf{Kg}(pp), Q \leftarrow \emptyset$

$(\sigma, m, apk, \pi, PK) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{MulSign}}}(pp, pk^*)$

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.

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 Figure 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 in Appendix D.1.

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 Definition 14 of Appendix D. 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:

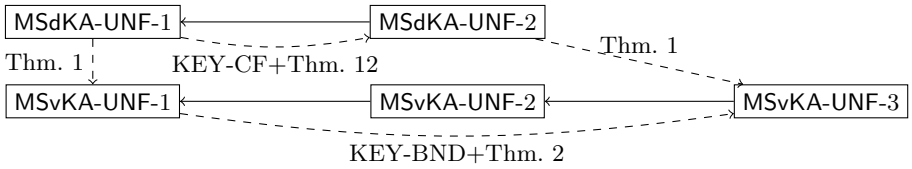


Fig. 5: Relation among unforgeability definitions. $A \overset{x}{\dashrightarrow} B$ means there is a generic construction of B from A relying on properties and/or theorems x. $A \rightarrow B$ means A implies B (any scheme has A, also has B).

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 Appendix D.2. 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 [30].

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.

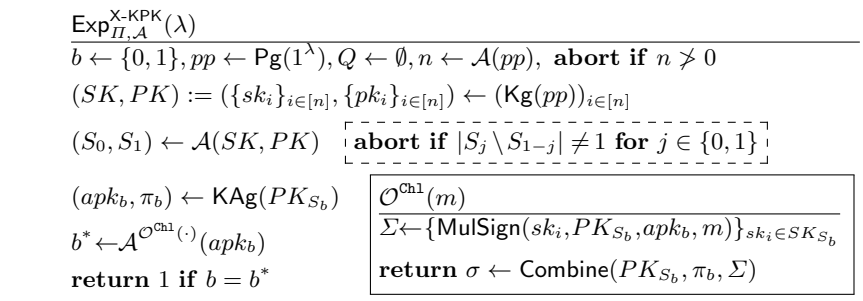


Fig. 6: Game for our Set Privacy ($\mathsf{X} = \mathsf{SetPriv}$) and Membership Privacy ($\mathsf{X} = \mathsf{MemPriv}$) definitions in the KPK model. The dashed box shows the additional condition for the Membership Privacy definition.

While the adversary must be an outsider to the challenge group (there is no privacy to insiders), he should be able to 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 be able to 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 ($\mathsf{SetPriv}$) and membership privacy ($\mathsf{MemPriv}$). Both require the indistinguishability of two 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 $\mathsf{MemPriv}$ version.

Set Privacy. Our $\mathsf{SetPriv}$ 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 corrresponding 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 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. We add this check through the additional 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 SetPriv.

Definition 6 ($\{$SetPriv, MemPriv$\}$-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 :* $\left|\Pr\left[\mathsf{Exp}^{\mathsf{X}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \leq \mathsf{negl}(\lambda)$.

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

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\Pi,\mathcal{A}}(\lambda)$

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$S^* \leftarrow \mathcal{A}(SK, PK)$

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 $b = 1$ **then**

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 in the KPK model capturing that aggregate signatures and keys are indistinguishable from standard ones.

schemes, this sign algorithm will be the standard BLS or Schnorr algorithm. The keys for the standard sign algorithm are the ones generated via MSvKA.Kg and it also uses the same verification algorithm MSvKA.Vf.

The detailed model is given in Def. 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, 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 (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* $\left|\Pr\left[\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \leq \mathsf{negl}(\lambda)$.

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:

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 Section 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.

Theorem 4 (FullPriv ⇒ 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 Appendix E.1.

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 Appendix E.1 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.

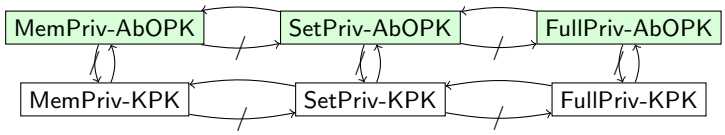


Fig. 8: Relation between our definitions under different models. $A \rightarrow B$ means A implies B (any scheme has property A, also has property B). $A \not\rightarrow B$ means A does not imply B (there exists a scheme s.t. has property A, but not property B). Properties with green boxes are achievable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

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:

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

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 Appendix E.2.

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 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 Section 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* $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$ *and* $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$ *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 so that the resulting aggregated key would be independent of the challenge public key pk^*. It is easy to see that this is not the case for our scheme, as we use the involved randomness as an additional input to the random oracle H_1. Hence, 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 $\mathsf{VfKAg}()$ 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 $e(A, apk') = e(\sigma', \hat{g})$ must hold, 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 full proof is in Appendix F.1.

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

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

Proof (Sketch). In the 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 Appendix F.2.

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 SetPriv-KPK. This is done by simply applying the generic UNF-1 to UNF-3 transformation (from Construction 2) to randBLS-1. That is, the MulSign algorithm of our second scheme randBLS-2 now strictly binds each signature to the intended apk by including the key in the hash.

Construction 4 (randBLS-2) *The* randBLS-2 *is identical to* 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.

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

The simple proof is given in Appendix F.3 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 now conclude the following:

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 next level SetPriv-KPK of privacy is satisfied: we have shown in Theorem 15 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 following theorem 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 Appendix F.4.

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.

Using Theorem 15 we can conclude:

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 & Analysis of Existing Constructions

We have already shown that all existing multi-signature schemes cannot satisfy the privacy properties defined in Section 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 Table 11.

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.

$\underline{\mathsf{Exp}^{\mathsf{X\text{-}AbOPK}}_{\Pi,\mathcal{A}}(\lambda)}$

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp),$ **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 $1 \notin S_j \vee |S_j \setminus S_{1-j}| \neq 1$ **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** $b = b^*$

$\underline{\mathcal{O}^{\mathtt{Chl}}(m)}$

$\Sigma \leftarrow \{\mathsf{MulSign}(sk_i, PK_{S_b}, apk_b, m)\}_{sk_i \in SK_{S_b}}$

return $\sigma \leftarrow \mathsf{Combine}(PK_{S_b}, \pi_b, \Sigma)$

Fig. 9: Game for our Set Privacy ($\mathsf{X} = \mathsf{SetPriv}$) and Membership Privacy ($\mathsf{X} = \mathsf{MemPriv}$) definitions in the AbOPK model. The dashed box shows the additional condition for the Membership Privacy definition.

$\underline{\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\Pi,\mathcal{A}}(\lambda)}$

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp),$ **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$S^* \leftarrow \mathcal{A}(SK \setminus \{sk_1\}, PK \setminus \{pk_1\})$ **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** $b = b^*$

$\underline{\mathcal{O}^{\mathtt{Chl}}(m)}$

if $b = 1$ **then**

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. 10: Game for our Full Privacy definition in the AbOPK model.

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 Section 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 $\left|\Pr\left[\mathsf{Exp}^{\mathsf{X\text{-}AbOPK}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \leq \mathsf{negl}(\lambda)$ where $\mathsf{Exp}^{\mathsf{X\text{-}AbOPK}}_{\Pi,\mathcal{A}}$ is defined in Figures 9 and 10.*

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 Appendix E.3.

6.2 Analysis of BLS & 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. [30]. We also note that our theorems and proofs can be easily adapted to other Schnorr multi-signatures that have the same key aggregation technique (e.g., [25, 32]).

BLS Multi-Signatures. In this section, we analyze the BLS multi-signature scheme from Boneh et al. [12], restate the already known unforgeability properties in our model and prove that it can either satisfy UNF-1 and FullPriv-AbOPK, or UNF-3 and SetPriv-AbOPK. The Construction 5 below implicitly applies our MSdKA-to-MSvKA transformation to the original protocol from [12].

Construction 5 (BLS-dMS [12]) BLS-dMS *uses hash functions* $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$, *a bilinear group generator* BGGen *and is defined as follows:*

Pg, Vf: same as randBLS-1.

Kg(pp)**:** Return (sk, pk) for $sk \leftarrow \mathbb{Z}_p$ and $pk \leftarrow \hat{g}^{sk}$.

KAg(PK)**:** Return $(apk, \perp)$ for $apk \leftarrow \prod_{pk_i \in PK} pk^{\mathsf{H}_1(pk_i, PK)}$.

VfKAg(PK, apk, π)**:** Return 1 if $(apk, \perp) = \mathsf{KAg}(PK)$. Return 0 otherwise.

MulSign(sk_i, PK, apk, m)**:** Return $s_i \leftarrow \mathsf{H}_0(m)^{sk_i \cdot \mathsf{H}_1(pk_i, PK)}$.

Combine$(PK, \pi, \{s_i\}_{pk_i \in PK})$**:** Return $\sigma \leftarrow \prod_{pk_i \in PK} s_i$.

Unforgeability. Boneh et al. [12] proved the security of their scheme in a model that is equivalent to what we define as MSdKA-UNF-1 security. Thus, BLS-dMS is MSvKA-UNF-1 secure by Theorem 1.

Clearly, BLS-dMS cannot achieve any of the unforgeability notions beyond MSvKA-UNF-1: e.g., let $s_i = \mathsf{H}_0(m)^{sk_i \cdot \mathsf{H}_1(pk_i, PK)}$ be a share of the honest user i for the group PK. Let further PK' be a set of public keys such that $PK' \neq PK$ and $pk_i \in PK'$. Then pk_i's signature share on message m for PK' can be computed as $s'_i \leftarrow s_i^{\mathsf{H}_1(pk_i, PK') \cdot t}$ for $t \leftarrow \mathsf{H}_1(pk_i, PK)^{-1}$. Thus, BLS-dMS is not MSvKA-UNF-2, and consequently neither UNF-3 secure.

Corollary 3. BLS-dMS *scheme from Construction 5 is* MSvKA-UNF-*1 secure (under the assumptions from [12]), but not* MSvKA-UNF-*2/3 secure.*

While BLS-dMS was designed for the flexible, i.e., non-group specific usage, for which UNF-1 is the right unforgeability level, one can also easily lift the scheme to achieve the stronger UNF-3 unforgeability guarantees needed for group signing. This is again done via key-prefixing, and we present a generic key-prefixing transformation (Cons. 7) for MSdKA which is analogous to the transformation in Construction 2 for MSvKA. In Appendix C, we show that this transformation is applicable to BLS-dMS to lift MSdKA-UNF-1 to MSdKA-UNF-2 security. We have further shown our MSdKA-to-MSvKA transformation turns MSdKA-UNF-2 into MSvKA-UNF-3 security (Thm. 1). Thus, we can conjecture the following:

Corollary 4. BLS-dMS *scheme with key-prefixing (*BLS-dMS-*2) is* MSvKA-UNF-*3 secure (under the assumptions from [12]) and assuming* H_1 *to be a random oracle.*

The price we pay for this transformation is that we lose any hope for the FullPriv property, as signatures cannot be verified with the standard verification algorithm anymore, but also require key-prefixing. In practical terms, the sacrifice is rather minor though. Although we do not provide full proof, we point out that BLS-dMS-2 is still SetPriv-AbOPK secure (and FullPriv-AbOPK for a non-standard sign algorithm). One can use the following proof strategy for this statement: Having FullPriv privacy for *any* Sign algorithm implies SetPriv, and we show that this holds for $\mathsf{Sign}'(sk, m) = \mathsf{H}_0(\hat{g}^{sk}, m)^{sk}$, which includes key-prefixing in the plain signatures.

Privacy. We now show that the UNF-1 secure version of the BLS-dMS scheme satisfies FullPriv-AbOPK privacy, i.e., produces multi-signatures and aggregate keys that are indistinguishable from individual ones (derived via BLSSign, from Sec. 2), if at least one public key remains unknown to the adversary.

Theorem 10. BLS-dMS *scheme in Construction 5 is* FullPriv-AbOPK *in the ROM for* H_1 *as a random oracle and for* Sign = BLSSign.

Proof (Sketch). The full proof is in Appendix G.1 and closely follows the privacy proof of our randBLS-1 scheme. The main difference between our randBLS-1

scheme and the BLS-dMS is that the aggregated key in BLS-dMS does not contain a randomly chosen π value. This value was crucial for the privacy of our scheme, and the only reason we can argue privacy of the deterministic variant is that we are now in the weaker AbOPK model which relies on the public key pk_1 to be secret. Thus, pk_1 essentially takes over the role of π in this proof, and we show in a series of games how the challenge aggregated key and signatures can be replaced by a freshly sampled public key and standard signatures.

Schnorr Multi-Signature. We now analyze Schnorr multi-signature scheme MuSig within our unforgeability and privacy hierarchies. Recently, Schnorr multi-signatures have seen an increasing interest, and several variants of Schnorr-based multi-signatures exist, e.g., [12, 30, 9, 16, 25, 32]. In general, these works aim to improve either efficiency, e.g. by reducing the number of rounds in MulSign, or security, e.g. by providing tighter security bounds. We choose the MuSig scheme [30] to analyze. The structure of MuSig is similar to the traditional multi-signature scheme of Bellare and Neven [10], and thus has the best chances for full privacy. Furthermore, MuSig is the first multi-signature scheme with aggregation that has been proven secure (by Boneh et al. [12] and Maxwell et al. [30], independently). We also note that our theorems and proofs can be easily adapted to other Schnorr multi-signatures that have the same key aggregation technique ([25, 32]). We present MuSig scheme adapted to our MSvKA syntax in Construction 6, which again implicitly uses the MSdKA-to-MSvKA transform from Sec. 2.1.

Construction 6 (MuSig [30]) MuSig *uses a group generator* GGen*, three hash functions* $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$ *and it is defined as follows.*

$\mathsf{Pg}(1^\lambda)$**:** Returns $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(\lambda)$.

$\mathsf{Kg}, \mathsf{KAg}, \mathsf{VfKAg}$**:** same as in BLS-dMS of Cons. 5 (but now in $\mathbb{G}$)

$\mathsf{MulSign}(sk_i, PK, apk, m)$**:** Signing algorithm is composed of 3 rounds.

Round 1. Choose $r_i \leftarrow \mathbb{Z}_p$, and compute $R_i \leftarrow g^{r_i}$,$t_i \leftarrow \mathsf{H}_2(R_i)$. Send t_i to other signers and wait for t_j for all $j \neq i$.
Round 2. Send R_i to other signers and wait for R_j for all $j \neq i$. If there exists a j such that $t_j \neq \mathsf{H}_0(R_j)$ then abort.
Round 3. Compute $(apk', \_) \leftarrow \mathsf{KAg}(PK)$, $R \leftarrow \prod_{pk_i \in PK} r_i$, and $c \leftarrow \mathsf{H}_0(R, apk', m)$. Finally, return $s_i \leftarrow (s_i', R_i)$ for $s_i' \leftarrow r_i + c \cdot sk_i \cdot \mathsf{H}_1(pk_i, PK)$.

$\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK})$**:** Parse s_i as (s_i', R_i). Return $\sigma \leftarrow (s, R)$ for $s \leftarrow \sum_{pk_i \in PK} s_i'$ and $R \leftarrow \prod_{pk_i \in PK} R_i$.

$\mathsf{Vf}(apk, \sigma, m)$**:** Parse σ as (s, R). Return 1 if $g^s = apk \cdot R^{\mathsf{H}_0(R, apk, m)}$ else 0.

Unforgeability. The original MuSig has been proven to be MSdKA-UNF-2 secure by Maxwell et al. [30] and Bellare and Dai [9]. By relying on Theorem 1, we conjecture the following:

Corollary 5. *The* MuSig *scheme in Construction 6 is* MSvKA-UNF*-3 secure.*

Scheme	Unforg.	FullPriv	SetPriv	MemPriv
BLS-dMS [12]	UNF-1	AbOPK	AbOPK	AbOPK
BLS-dMS + KeyPrefix (Cons. 2)	UNF-3	–/(AbOPK*)	AbOPK	AbOPK
MuSig [30]	UNF-3	AbOPK*	AbOPK	AbOPK
Our Work (randBLS-1)	UNF-1	KPK	KPK	KPK
Our Work (randBLS-2)	UNF-3	–/(KPK*)	KPK	KPK

Fig. 11: 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.

One point to note is that this scheme already applies key prefixing, and thus naturally achieves our strongest notions.

Privacy. Just as BLS-dMS, the Schnorr multi-signature scheme in Construction 6 also achieves FullPriv-AbOPK, i.e., produces keys and signatures that are indistinguishable from standard SchnorrSign ones. This again crucially relies on (at least) one public key to remain secret.

In contrast to BLS, the MuSig scheme enjoys both MSvKA-UNF-3 and FullPriv-AbOPK security at the same time. However, the reason that it achieves FullPriv-KPK is slightly subjective, as it depends on what one considers the "standard" signature analogue for a multi-signature. As the literature widely uses the key-prefixed Schnorr, even in the stand-alone setting, we followed that choice and thus we can show indistinguishable from such standard signatures. If key-prefixing becomes more standard in BLS too, then we can immediately claim FullPriv-AbOPK for BLS-dMS-2 as well.

Theorem 11. MuSig *scheme in Construction 6 is* FullPriv-AbOPK *in ROM for* H_1 *as a random oracle and for* Sign = SchnorrSign.

Proof (Sketch). The proof is almost analogous to the one for BLS-dMS, as key aggregation is identical in both schemes. What remains to be shown is that the distributed computation of $R = \prod_{pk_i \in PK} g^{r_i}$ in MuSig is indistinguishable from $R = g^r$ in SchnorrSign. This is straightforward, as the adversary never sees the individual contributions of the different signers for the challenge aggregated key. Thus FullPriv-AbOPK follows. The full proof of this theorem is available in Appendix G.2.

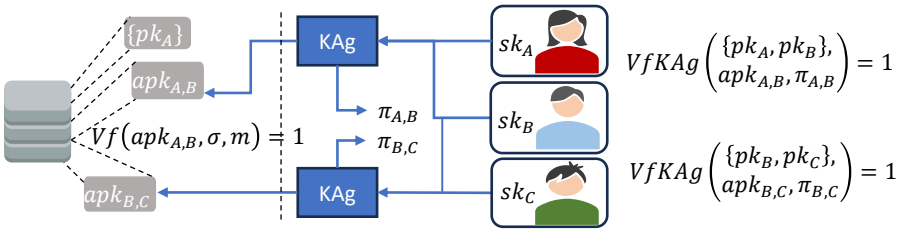


Fig. 12: Group signing using MSvKA in a public ledger.

7 Applications & Discussion

In this section, we sketch two applications that are enabled through our new multi-signatures and discuss which of the privacy and security properties are needed for the different scenarios. At the end, we also discuss open problems.

7.1 Public Ledgers

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 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 Figure 12 is, to us, the most natural one.

Individual Setup. Every user generates their individual key pair, as $\mathsf{Kg}(pp) \to (sk_X, pk_X)$ 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. We assume this is done by Alice, who runs:

$$\mathsf{KAg}(\{pk_A, pk_B\}) \to (apk_{A,B}, \pi_{A,B})$$

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:

$$\mathsf{VfKAg}(\{pk_A, pk_B\}, apk_{A,B}, \pi_{A,B}) \to b$$

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 to be the designated combiner, then only this party needs to keep the associated π. Note that the values established for each group are not security-critical – their leakage would 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:

$$\mathsf{MulSign}(sk_A, \{pk_A, pk_B\}, apk_{A,B}, m) \to s_A$$

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

$$\mathsf{MulSign}(sk_B, \{pk_A, pk_B\}, apk_{A,B}, m) \to s_B$$

$$\mathsf{Combine}(\{pk_A, pk_B\}, \pi_{A,B}, \{s_A, s_B\}) \to \sigma$$

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 $\mathsf{Vf}(apk_{A,B}, \sigma, m) \to b$. 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.

7.2 Privacy-Preserving Authentication from Hardware Tokens

Apart from using multi-signatures to build groups of different users to cater for an increased level of security, it can also be used to improve the individual key management of end-users which we want to briefly sketch here. Users can use a hardware token that contains a single (certified) key pair (sk_T, pk_T) and use that to bootstrap unlinkable but hardware-protected key pairs for strong user authentication: When the user wants to create an account with a service provider, she creates a fresh and service-specific key pair (sk_i, pk_i), which can be stored on an untrusted client. The user registers with the aggregated key apk_i derived from pk_T and pk_i with the service and consequently always needs both underlying signing keys, i.e., in particular the hardware-protected one, to access her account. If the service provider wants to be assured that the key apk_i is indeed (partially) protected by hardware, the user sends both individual public keys and the certificate of pk_T to the service provider, which uses VfKAg to verify that apk_i is properly formed.

This allows the user to rely on a single hardware token to create many unlinkable account keys. If the scheme is at least MemPriv-KPK private, then an adversary learning two account public keys $apk_i \neq apk_j$ cannot tell whether they belong to the same user or not – this even holds if $\mathcal{A}$ knows the user's long-term hardware key pk_T. Having FullPriv-KPK even fully hides the fact that the user is using such a scheme. In terms of security, the level UNF-3 is necessary here, as it ensures that all of the user's signatures are strictly bound to the account key she wants to authenticate for, and cannot be used by a malicious service provider in a phishing attempt.

In this context, one could even argue that the MemPriv-AbOPK privacy suffices, as the service-specific public keys serve as internal "randomizers" by the honest users and service providers will only need them to register and verify apk_i. However, we still caution the use of the AbOPK model, as it relies on the secrecy of public keys, which – even when being reasonable for a particular application in theory – might be hard to guarantee in practice.

7.3 Discussion

Finally, we discuss open problems and possible directions for future work.

Schnorr Multi-Signatures with Improved Privacy. Our work shows how strong privacy can be achieved for BLS-based multi-signatures, and we expect that the randomization technique can be applied analogously for Schnorr-based schemes as well. This also requires to prove that the modified schemes are unforgeable, which we leave as interesting future work.

Privacy Against Insiders. Our work focuses on privacy against outsiders, as this seems to be the most basic requirement. One could further consider a more fine-grained security notion that also considers some form of privacy towards insiders. More precisely, our definition relies on an explicit algorithm $\mathsf{VfKAg}(PK, apk, \pi)$ to verify whether an apk corresponds to the group PK. For some applications, it might be desirable to remove the requirement of knowing the full set PK and instead only verify whether a particular key pk is contained in apk or not.

Unforgeability. So far, all MSvKA-UNF-3 constructions including ours rely on key-prefixing to bind a signature to a particular apk. Although it is not a problem for the use cases that already employ key-prefixing such as public ledgers, there may be use cases that do not internally employ key-prefixing. An open problem is whether we can have MSvKA-UNF-3 secure schemes (also for Schnorr-signatures) without key prefixing.

Further, our definition of multi-signature MSvKA brought a new intermediate level of unforgeability, MSvKA-UNF-2. Neither of the analyzed or proposed schemes sits at that level though, they are either UNF-1 or UNF-3 secure. It thus remains an open question, whether there are constructions that satisfy the MSvKA-UNF-2 notion, and are possibly able to benefit from the relaxed requirements compared to UNF-3.

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., Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Threshold signatures in the multiverse. In: 2023 IEEE Symposium on Security and Privacy (SP) (2023)
7. Baldimtsi, F., Chalkias, K.K., Garillot, F., Lindstrom, J., Riva, B., Roy, A., Sonnino, A., Waiwitlikhit, P., Wang, J.: 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 '06 (2006)
11. Boldyreva, A.: Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. pp. 31–46. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2002)
12. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Advances in Cryptology – ASIACRYPT 2018. Springer International Publishing (2018)
13. Boneh, D., Komlo, C.: Threshold signatures with private accountability. In: Advances in Cryptology – CRYPTO 2022 (2022)
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. pp. 651–668. CCS '19, Association for Computing Machinery, New York, NY, USA (Nov 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: Advances in Cryptology – CRYPTO 2021 (2021)
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. Li, M., Zhang, M., Wang, Q., Ding, H., Meng, W., Zhu, L., Zhang, Z., Lin, X.: Decentralized Threshold Signatures with Dynamically Private Accountability (Aug 2023), http://arxiv.org/abs/2304.07937, arXiv:2304.07937 [cs]
30. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (2019)
31. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: Proceedings of the 8th ACM conference on Computer and Communications Security (2001)
32. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: Simple Two-Round Schnorr Multisignatures. In: Advances in Cryptology – CRYPTO 2021 (2021)
33. Nick, J., Ruffing, T., Seurin, Y., Wuille, P.: Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)
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: Advances in Cryptology - EUROCRYPT 2007 (2007)
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)
39. Wuille, P., Nick, J., Towns, A.: Validation of taproot scripts, https://github.com/bitcoin/bips/blob/e918b50731397872ad2922a1b08a5a4cd1d6d546/bip-0342.mediawiki

A Further Preliminaries

Here we provide formal definitions that are deferred in Section 2.

Cyclic Groups. We will use a group generator GGen as follows for the Schnorr-based scheme.

Definition 9 (Group Generator). *A group generator* GGen *is a p.p.t. algorithm outputs a prime order group description* $\mathcal{G} \leftarrow (\mathbb{G}, g, p)$ *s.t.* $\langle g \rangle = \mathbb{G}$ *is a group of prime order* p *and* $\lceil \log_2 p \rceil = \lambda$.

Pairing Groups. For BLS, we will use a bilinear pairing generator BGGen as follows.

Definition 10 (Bilinear Pairing). *For* $\langle g \rangle = \mathbb{G}$, $\langle \hat{g} \rangle = \hat{\mathbb{G}}$ *and* $\mathbb{G}_T$ *which are groups of prime order* p, $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$ *is a bilinear pairing if it is efficiently computable and* bilinear*:* $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab} = e(g^b, \hat{g}^a)$ $\forall a, b \in \mathbb{Z}_p$, *and* non-degenerate*:* $\langle e(g, \hat{g}) \rangle = \mathbb{G}_T$, *so* $e(g, \hat{g}) \neq 1_{\mathbb{G}_T}$. *A bilinear group generator* BGGen *is an algorithm which outputs a bilinear pairing description* $\mathcal{BG} = (e, \mathbb{G}, \hat{\mathbb{G}}, g, \hat{g}, p)$ *such that* $\lceil \log_2 p \rceil = \lambda$ *the requirements above hold.*

Generalized Forking Lemma. We give the detailed definition of generalized forking lemma here. Let $\mathcal{A}$ be an adversary that interacts with a random oracle $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p$ on input par. Let $f = (\rho, h_1, ..., h_{q_H})$ denote the randomness involved in the execution of $\mathcal{A}$ such that ρ is the random input of $\mathcal{A}$ and h_j is the response of H for j'th query where q_H is the maximum number of queries. Let Ω denote the set of all vectors f. We say that $\mathcal{A}$ is successful if $\mathcal{A}$ outputs a tuple $(J, \{\phi_j\}_{j \in J})$ where J is a non-empty multi-set of indices such that $J \subseteq [q_h]$. $\mathcal{A}$ outputs $(J, \{\phi_j\}_{j \in J})$ for $J = \emptyset$ if it fails. Let f_i denote $(\rho, h_1, ..., h_{i-1})$. Let ϵ denote the success probability of $\mathcal{A}$'. On input par, $\mathcal{GF}_{\mathcal{A}}$ acts as follows.

Lemma 1 (Generalized Forking Lemma). *Let* IG *be a randomized algorithm that generates* par *and* $\mathcal{A}$ *be a randomized algorithm that makes at most* q_H *oracle queries such that* $\mathcal{A}$ *succeeds with probability* ϵ. *If* $p > 8nq_H/\epsilon$, *then* $\mathcal{GF}_{\mathcal{A}}$ *succeeds with probability* $\mathsf{frk} > \epsilon/8$.

B Correctness Definitions

The correctness definitions that belong to the traditional multi-signatures and ours are provided in this section.

B.1 Correctness Definition of MSdKA

The correctness of a MSdKA scheme is defined as follows.

Definition 11 (MSdKA-Correctness). *A multi-signature scheme* MSdKA *is correct if for all* λ, *for all* $pp \leftarrow \mathsf{Pg}(1^\lambda)$, *for all messages* m, *for all* n, *for all* $(sk_i, pk_i) \leftarrow \mathsf{Kg}(pp)$ *for* $i \in [n]$, *for all* $s_i \leftarrow \mathsf{MulSign}(sk_i, \{pk_i\}_{i \in [n]}, m)$,

$$\mathsf{Vf}(\mathsf{KAg}(\{pk_i\}_{i \in [n]}), \mathsf{Combine}(\{pk_i\}_{i \in [n]}, \{s_i\}_{i \in [n]}), m) = 1$$

$\mathcal{GF}_{\mathcal{A}}(\mathsf{par})$

$f = (\rho, h_1, ..., h_{q_H}) \leftarrow \Omega$
$(J, \{\phi_j\}_{j\in J}) \leftarrow \mathcal{A}(\mathsf{par}, f)$
if $j = \emptyset$ **then return** $\perp$
Let $J = \{j_1, ..., j_n\}$ s.t. $j_1 \leq ... \leq j_n$
$out = \{(h_j, \phi_j)_{j\in J}\}, out' = \emptyset$
for i **in** $\{1, ..., n\}$
  $succ_i \leftarrow 0, k \leftarrow 0, k_{max} \leftarrow 8nq_h/\epsilon \cdot \ln(8n/\epsilon)$
  while $succ_i = 0 \wedge k_i \leq k_{max}$ **do**
    $f' \leftarrow \Omega,$ s.t. $f'_i = f_i$
    $(J', \{\phi'_j\}_{j\in J'})\mathcal{A}(\mathsf{par}, f')$
    if $h_{j_i} \neq h_{j'_i} \wedge J \neq \emptyset \wedge j_i \in J'$
      then $out' = out' \cup \{(h'_{j_i}, \phi'_{j_i})\}, succ_i = 1$
if $succ_i = 1$ for $i \in \{1, ..., n\}$ **then return** (out, out')
else return $\perp$

Fig. 13: Algorithm $\mathcal{GF}_{\mathcal{A}}$ for Generalized Forking Lemma

	MSdKA-UNF-1 (fresh m)	MSdKA-UNF-2 (fresh (m, PK))
with Key Aggr.	$[12, 19, 20, 25, 28]$	$[9, 26, 32, 35, 34, 38]$
without Key Aggr.	$[16, 5, 36]$	$[9, 10, 30, 33]$

Fig. 14: Categorization of previous works w.r.t their targeted unforgeability definitions

B.2 Correctness Definition of MSvKA

The correctness of a MSvKA scheme is defined as follows.

Definition 12 (MSvKA-Correctness). *A multi-signature scheme* MSvKA *is correct if for all λ, for all m, for all n, for all $pp \leftarrow \mathsf{Pg}(1^\lambda)$ for all $(sk_i, pk_i) \leftarrow \mathsf{Kg}(pp)$ for $i \in [n]$, for all $(apk, \pi) \leftarrow \mathsf{KAg}(\{pk_i\}_{i\in[n]})$, for all $s_i \leftarrow \mathsf{MulSign}(sk_i, \{pk_i\}_{i\in[n]}, apk, m)$ for $i \in [n]$,*

$$\mathsf{VfKAg}(\{pk_i\}_{i\in[n]}, apk, \pi) = 1 \wedge \mathsf{Vf}(apk, \mathsf{Combine}(\{pk_i\}_{i\in[n]}, \pi, \{s_i\}_{i\in[n]}), m) = 1$$

C MSdKA Unforgeability Relations and Transformations

This section contains full proofs and/or additional definitions related to unforgeability notions of MSdKA and their relations. Also, Figure 14 is presented to recap the position of existing schemes.

Relations and Transformations. A natural question is whether we can have black-box construction of a MSdKA-UNF-2 scheme from a MSdKA-UNF-1 one. If the message space allows, the straightforward way is to sign (PK, m) instead of m. The problem with that approach is that it immediately rules out most of the efficiency and/or privacy properties we would gain by applying key aggregation, since the set of public keys is then necessary to verify whether the signature is valid. Thus, to have any hope of having MSdKA-UNF-2 while also keeping any features that come with key aggregation, one needs to bind the signature to *apk* instead of PK. In fact, previous works that achieve the MSdKA-UNF-2 notion, follow this approach and mostly sign (apk, m) [32, 9, 16].

As a side result, we present a generic transformation based on the idea of key-prefixing for MSdKA-UNF-2 security. To capture this idea in a generic transformation, an additional property on *apk* and the underlying key aggregation mechanism. We call this property *key collision-freeness* and it has been internally used in [9, 32]. Intuitively, it assures that it is infeasible for an adversary to find two non-empty sets of public keys PK_0 and PK_1 such that $PK_0 \neq PK_1$ and $\mathsf{KAg}(PK_0) = \mathsf{KAg}(PK_1)$. This notion is formally defined and proved below for common key aggregation mechanism that is used by the literature. We also show that our transformation holds and provide other side-results related to key collision-freeness.

Definition 13 (MSdKA Key Collision-Freeness). *A multi-signature scheme* MSdKA *is key collision-free if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$,* $\Pr\left[\mathsf{Exp}^{\mathsf{MSdKA}\text{-}KEY\text{-}CF}_{\mathsf{MSdKA},\mathcal{A}}(\lambda) = 1\right] \leq \mu(\lambda)$.

$\mathsf{Exp}^{\mathsf{MSdKA}\text{-}KEY\text{-}CF}_{\mathsf{MSdKA},\mathcal{A}}(\lambda)$

$pp \leftarrow \mathsf{Pg}(1^\lambda), (PK_0, PK_1) \leftarrow \mathcal{A}(pp)$

return 1 **if** $PK_0 \neq PK_1 \wedge \mathsf{KAg}(PK_0) = \mathsf{KAg}(PK_1)$

Construction 7 (MSdKA-UNF-1 to MSdKA-UNF-2 Transformation) *Let a* MSdKA-UNF-*1 multi-signature scheme Π have the aggregated key space $\mathcal{AP}$ and message space $\mathcal{M} = \mathcal{AP} \times \mathcal{M}^*$. Then, we define Π' with message space $\mathcal{M}^*$ as follows. The algorithms* $(\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{Combine})$ *of Π' are exactly as in Π, and the remaining algorithms are:*

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

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

Theorem 12. *If Π is a multi-signature scheme that is* MSdKA-UNF-*1 secure and key collision-free, then Π' from Construction 7 is* MSdKA-UNF-*2 secure.*

Proof. Let $\mathcal{F}'$ be a MSdKA-UNF-2 forger against Π'. Let (σ, m, PK) be $\mathcal{F}'$'s forgery. An MSdKA-UNF-1 forger $\mathcal{F}$ against Π works as follows. It runs $\mathcal{F}'$ and answers all queries using MSdKA-UNF-1 challenger as it is defined in Construction 7. Intuitively, if $\mathcal{F}'$ forgery is for not a fresh (m, apk), it means that PK

in the forgery is different from the one $\mathcal{F}$ queried, so we find a key collision. Otherwise, we directly have the forgery for MSdKA-UNF-1.

Game_0 is equivalent to the original MSdKA-UNF-2 experiment.

Game_1: Let event F be the event that there is a previous query for (m, PK') such that $PK \neq PK'$ and $\mathsf{KAg}(PK) = \mathsf{KAg}(PK')$. If the event F happens we abort. By the winning condition $PK' \neq PK$. We can build an adversary $\mathcal{A}$ against key collision freeness of the scheme using such PK'. Hence, $\Pr[F] \leq \mu_{\text{MSdKA-KEY-CF}}(\lambda)$.

$$|\Pr[W_0] - \Pr[W_1]| \leq \mu_{\text{MSdKA-KEY-CF}}(\lambda)$$

We can simply build an MSdKA-UNF-1 forger $\mathcal{F}$ against Π by simulating the public parameters and the challenge public key identically to the MSdKA-UNF-2 forger $\mathcal{F}'$ against Π'. To simulate $\mathcal{F}'$'s signing query $\mathcal{O}^{\mathtt{MulSign}}_{\Pi'}(PK_i, m_i)$, we make a $\mathcal{O}^{\mathtt{MulSign}}_{\Pi}(PK_i, (\mathsf{KAg}(PK_i), m_i))$ query to MSdKA-UNF-1 challenger. We know that there is no signing query to MSdKA-UNF-1 challenger for (m, apk) where $apk \leftarrow \mathsf{KAg}(PK)$ in Game_1. Thus, $(\sigma, (apk, m), PK)$ is a valid MSdKA-UNF-1 forgery for MS.

$$\Pr[W_1] \leq \mu_{\text{MSdKA-UNF-1}}(\lambda)$$

Thus, we either have a key collision, or MSdKA-UNF-1 forgery. □

Key Collision-freeness and DL-based Schemes We now show that key collision-freeness is naturally achieved by DL-based schemes, such as BLS ([12]) and Schnorr-based ([12, 32, 25]) constructions that use the key aggregation technique initially proposed in [30]. Thus, we now consider an MSdKA scheme that works in a cyclic group $(\mathbb{G}, g, p)$ of prime order p with generator g and that uses a hash function $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$, and where pp contains $((\mathbb{G}, g, p), \mathsf{H}_1)$. We make a minor change to the previous schemes, [12, 25, 32, 30], and assume that the secret key is sampled from $\mathbb{Z}_p^*$ instead of $\mathbb{Z}_p$. By this change, we eliminate the case that a pk may be the identity element of the group, so it is easier to determine the distribution of key aggregation.

Construction 8 (DL-based Key Aggregation) *Let a DL-based multi-signature scheme Π_{DL} have the following algorithms for key generation and key aggregation [30] where $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function:*

$\mathsf{Kg}(pp)$*: For* $sk \leftarrow \mathbb{Z}_p^*$ *and* $pk \leftarrow g^{sk}$*, outputs* (sk, pk).
$\mathsf{KAg}(PK)$*: Outputs* apk *where* $apk \leftarrow \prod_{pk_i \in PK} pk_i^{\mathsf{H}_1(pk_i, PK)}$.

Theorem 13. *Any DL-based multi-signature scheme* MSdKA *with the key generation and aggregation algorithms defined in Construction 8 is key collision-free if* H_1 *is a random oracle.*

Proof. Intuitively, we show that each apk is uniformly random per PK in ROM. The rest of the proof is about dividing collisions into cases such that each case can occur with negligible probability.

A proof in the random oracle model is provided as follows. A hash query $\mathsf{H}_1(pk, PK)$ is answered as follows. The proof mainly relies on the fact that each apk is uniformly random in ROM, since public keys are generators of the underlying group. Once we know that apk's are uniformly random, the rest follows from the traditional birthday problem.

- if $pk \in PK$ and this is the first query for PK, set $\mathsf{H}_1(pk', PK) \leftarrow \mathbb{Z}_p$ for all $pk' \in PK$.
- if $pk \in PK$ and there is a previous query for PK, answer the query using previously set value.
- if $pk \notin PK$, simulate the query as a uniformly random choice from $\mathbb{Z}_p$.

We consider three cases for the winning event of $\mathcal{A}$.

Event W_1: there does not exist a query for PK_0.
- Let $apk_1 \leftarrow \mathsf{KAg}(PK_1)$ which has been set by the previous hash query for PK_1. As there is no query for PK_0 yet, it is simulated by the verification as $apk_0 \leftarrow pk_0^{\mathsf{H}_1(pk_0, PK)} \cdot Z$ for some $pk_0 \in PK_0$ and for some Z. As $pk_0 \in \mathbb{G}^*$ and $\mathsf{H}_1(pk_0, PK_0) \in \mathbb{Z}_p$ is uniformly random, apk_0 is uniformly random choice from $\mathbb{G}$. Hence, $\Pr[W_1] = 1/p$

Event W_2: there does not exist a query for PK_1.
- Similar to the first case. $\Pr[W_2] = 1/p$.

Event W_3: there exists queries for both PK_0 and PK_1.
- Query for PK_b creates a uniformly random apk_b in $\mathbb{G}$. Hence, for q random oracle queries, $\Pr[W_3] \leq q^2/p$

It is obvious that

$$\mathsf{Exp}^{\mathsf{MSvKA\text{-}KEY\text{-}BND}}_{\Pi,\mathcal{A}} \leq \Pr[W_1] + \Pr[W_2] + \Pr[W_3]$$

Thus, we conclude that $\mathsf{Exp}^{\mathsf{MSvKA\text{-}KEY\text{-}BND}}_{\Pi,\mathcal{A}} \leq (q^2 + 2)/p$. □

D MSvKA Unforgeability Relations and Transformations

This section contains some additional definitions and full proofs of several theorems which are related to the relations and transformations among MSvKA unforgeability definitions.

Key Binding Definition. The formal definition of the key binding property for a MSvKA scheme is as follows.

Definition 14 (MSvKA Key Binding). *A multi-signature scheme Π is key binding if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$,* $\Pr\left[\mathsf{Exp}^{\mathsf{MSvKA}\text{-}KEY\text{-}BND}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mu(\lambda)$.

$\mathsf{Exp}^{\mathsf{MSvKA}\text{-}KEY\text{-}BND}_{\Pi,\mathcal{A}}(\lambda)$

$pp \leftarrow \mathsf{Pg}(1^\lambda), (apk, PK_0, PK_1, \pi_0, \pi_1) \leftarrow \mathcal{A}(pp)$

return 1 **if** $PK_0 \neq PK_1 \ \wedge \ \mathsf{VfKAg}(PK_0, apk, \pi_0) = 1 \ \wedge \ \mathsf{VfKAg}(PK_1, apk, \pi_1) = 1$

D.1 Proof of MSdKA to MSvKA Transformation (Theorem 1)

We show that the Construction 1 brings MSdKA-1 and MSdKA-2 unforgeability properties to MSvKA-1 and MSvKA-3, respectively.

Proof. First, we consider the claim that if Π is MSdKA-UNF-2, then Π' is MSvKA-UNF-3 secure. Let $\mathcal{A}$ be an efficient adversary against MSvKA-UNF-3 of MSvKA above. Then, we build $\mathcal{B}$ against MSdKA-UNF-2 of MSdKA as follows.

We run $\mathcal{A}$ to get the challenge (pp, pk^*) and run $\mathcal{B}$ with the same challenges. The signing queries are answered as follows. Like in the original oracle, we check whether $\mathsf{MSvKA.VfKAg}(PK, apk, \pi) = 1$ or not. If not, the oracle outputs $\perp$. Otherwise, oracle calls $\mathsf{MSdKA.MulSign}(sk^*, PK, m)$ and simulates the query with its result.

In the end, if $\mathcal{A}$ outputs a valid forgery $(\sigma, m, apk, \pi, PK)$, then $\mathcal{B}$ outputs (σ, m, PK). As $\mathsf{MSdKA.KAg}(\cdot)$ is deterministic, there exists only a single aggregated public key for each set PK. Thus, if $(PK, apk, m) \in Q_{\mathsf{MSvKA}}$, then $(PK, m) \in Q_{\mathsf{MSdKA}}$. Furthermore, if $\mathcal{A}$'s output is a valid MSvKA-UNF-3 forgery, then $\mathsf{VfKAg}(PK, apk, \pi) = 1$ which means $apk = \mathsf{MSdKA.KAg}(PK)$. Therefore, we conclude $\mathsf{MSdKA.Vf}(\mathsf{MSdKA.KAg}(PK), \sigma, m) = 1$.

Now, we consider the second claim. Let $\mathcal{A}$ be an efficient adversary against MSvKA-UNF-1 of MSvKA scheme. To build the MSdKA-UNF-1 adversary, we answer the MSvKA adversary's queries in the identical way that we explained above. Similar to the first claim, a valid MSdKA-UNF-1 forgery can be used as MSvKA-UNF-1 forgery, since $(\cdot, \cdot, m) \notin Q_{\mathsf{MSvKA}}$ implies $(\cdot, m) \notin Q_{\mathsf{MSdKA}}$. □

D.2 Proof of MSvKA-UNF-1 to MSvKA-UNF-3 Transformation (Theorem 2)

This section contains the proof of our MSvKA-UNF-1 to MSvKA-UNF-3 transformation's security and the proof of key-binding property of randBLS-1.

Proof. This proof follows almost the same strategy as the MSdKA transformation proof in Appendix C. Let $\mathcal{F}^*$ be a MSvKA-UNF-3 forger against MSvKA^*. Let $(\sigma, m, apk, \pi, PK)$ be $\mathcal{F}^*$'s forgery and Game_0 is equivalent to the original MSvKA-UNF-3 experiment.

Game_1: Let the event F be the event that there is a previous query for (PK', apk, π', m) such that $\mathsf{VfKAg}(PK, apk, \pi) = \mathsf{VfKAg}(PK', apk, \pi') = 1$. If the event F happens we abort. By the winning condition $PK' \neq PK$. We can build an adversary $\mathcal{A}$ against key binding of the scheme using such PK' and π'. Hence, $\Pr[F] \leq \mu_{\text{MSvKA-KEY-BND}}()$.

$$|\Pr[W_0] - \Pr[W_1]| \leq \mu_{\text{MSvKA-KEY-BND}}(\lambda)$$

We build an MSvKA-UNF-1 forger $\mathcal{F}$ against Π by simulating the public parameters and the challenge public key identically to the MSvKA-UNF-3 forger $\mathcal{F}'$ against Π'. To simulate $\mathcal{F}'$'s signing query $\mathcal{O}_{\Pi'}^{\texttt{MulSign}}(PK_i, apk_i, \pi_i, m_i)$, we make a $\mathcal{O}_{\Pi}^{\texttt{MulSign}}(PK_i, apk_i, \pi_i, (apk_i, m_i))$ query to MSvKA-UNF-1 challenger.

We know that there is no signing query to MSvKA-UNF-1 challenger for the message (apk, m) in Game_1. Thus, $(\sigma, (apk, m), apk, \pi, PK)$ is a valid MSvKA-UNF-1 forgery for Π.

$$\Pr[W_1] \leq \mu_{\mathsf{MSvKA\text{-}UNF\text{-}1}}(\lambda)$$

Thus, Π is either not key binding, or not MSvKA-UNF-1 secure. □

E Relations Among Privacy Definitons

In this section, we analyze the relations among privacy games and models. We provide the formal proofs for the relations we claimed in Section 4.2.

E.1 Proof of Theorem 4

The claim of Theorem 4 is that FullPriv property is strictly stronger than SetPriv property. The natural way to prove this claim is to prove two weaker claims: all FullPriv multi-signature schemes are SetPriv, and there is a multi-signature scheme that is SetPriv, but not FullPriv. We prove that these weaker claims hold, and we conclude that Theorem 4 holds.

SetPriv Does Not Imply FullPriv. The following theorem shows that SetPriv does not imply FullPriv.

Theorem 14. *There exists a multi-signature scheme Π' such that it is* SetPriv*, but it does not have* FullPriv *property.*

Proof. Intuitively, we take a multi-signature scheme that has both properties. Then, on top of it, we build a new multi-signature scheme such that it has SetPriv, but does not have FullPriv. The main strategy for creating such a multi-signature scheme is changing key spaces in a way that the distribution of key generation and key aggregation algorithm's outputs will be distinguishable. Let the multi-signature scheme $\Pi = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{VfKAg})$ be a multi-signature scheme which has FullPriv property against a Sign algorithm and MemPriv property with secret key space $\mathcal{S}$ and public key space $\mathcal{P}$. We create Π' with secret key space $\mathcal{S} \times \{0,1\}$ and public key space $\mathcal{P} \times \{0,1\}$ as follows.

$\Pi'.\mathsf{Kg}(pp)$**:** Runs $(sk, pk) \leftarrow \Pi.\mathsf{Kg}(pp)$, and outputs $((sk, 0), (pk, 0))$.
$\Pi'.\mathsf{KAg}(PK')$**:** Let PK be the set of pk for $(pk, \cdot) \in PK'$. Then the algorithm runs $(apk, \pi) \leftarrow \Pi.\mathsf{KAg}(PK)$ and outputs $((apk, 1), \pi)$.
$\Pi'.\mathsf{MulSign}(sk_i', PK', apk', m)$: Computes PK as in $\Pi'.\mathsf{KAg}()$. Then parses apk' and sk_i' as $(apk, \cdot)$ and $(sk_i, \cdot)$, respectively. Finally, outputs $s_i \leftarrow \Pi.\mathsf{MulSign}(sk_i, PK, apk, m)$.
$\Pi'.\mathsf{Combine}(PK', \pi, \{s_i\}_{pk_i' \in PK'})$**:** Computes PK as in $\Pi'.\mathsf{KAg}()$. Then, computes and outputs $\sigma \leftarrow \Pi.\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK})$.
$\Pi'.\mathsf{VfKAg}(PK', apk', \pi)$**:** Parses/computes PK and apk accordingly and outputs $b \leftarrow \Pi.\mathsf{VfKAg}(PK, apk, \pi)$.
$\Pi'.\mathsf{Vf}(apk', \sigma, m)$**:** Parses apk' as $(apk, \cdot)$ and then outputs the bit $b \leftarrow \Pi.\mathsf{Vf}(apk\ \sigma, m)$.

$\mathcal{B}$

$pp \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(\lambda), n \leftarrow \mathcal{B}'(pp)$

$(SK, PK) \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(n), (SK', PK') \leftarrow \{(sk_i, (pk_i, 0)) : pk_i \in PK\}$

$(S_0, S_1) \leftarrow \mathcal{B}'(SK', PK'), apk \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(S_0, S_1)$

$b^* \leftarrow \mathcal{B}'^{\mathcal{O}^{\mathtt{Chl}'}}((apk, 1)), \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(b^*)$

$\mathcal{O}^{\mathtt{Chl}}(m)$

return $\sigma \leftarrow \mathcal{O}^{\mathtt{Chl}'}(m)$

Fig. 15: Adversary $\mathcal{B}$ against $\mathsf{Exp}_{\pi}^{\mathsf{SetPriv}}$ which simulates $\mathsf{Exp}_{\pi'}^{\mathsf{SetPriv}}$ against $\mathcal{B}'$. For notational clarity, the common symbols of both games are differentiated with apostrophes (e.g. $\mathcal{O}^{\mathtt{Chl}}$ and $\mathcal{O}^{\mathtt{Chl}'}$).

Not FullPriv. We show that Π' cannot be FullPriv for any $\mathsf{Sign}(\cdot,\cdot)$ algorithm. In FullPriv game, if $b = 0$, then the adversary gets pk' in the form of $(\cdot, 1)$. Otherwise, pk' is in the form of $(\cdot, 0)$. Thus, the adversary trivially wins the game.

SetPriv. Assume there is an adversary $\mathcal{B}'$ against $\mathsf{Exp}_{\Pi'}^{\mathsf{SetPriv}}$. Then we can build an adversary $\mathcal{B}$ against $\mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}$, using $\mathcal{B}'$. We show how $\mathcal{B}$ can simulate the view of $\mathcal{B}'$ as in Figure 15. $\mathcal{B}$ simulates the additional bits in individual public keys by simply appending 0's to the individual public keys. Once individual public keys are sent to $\mathcal{B}'$, all other outputs of $\mathcal{B}$ are independent of additional bits, and $\mathcal{B}$ succeeds whenever $\mathcal{B}'$ succeeds. □

FullPriv Implies SetPriv. The following theorem shows that FullPriv implies SetPriv.

Theorem 15. *If a multi-signature scheme Π has the* FullPriv *property against any* Sign*() algorithm, then it has the* SetPriv *property.*

Proof. Our proof strategy is as follows. We need to prove that $apk_{S_0} \leftarrow \mathsf{KAg}(S_0)$ and $apk_{S_1} \leftarrow \mathsf{KAg}(S_1)$ (also the corresponding challenge queries) are indistinguishable. We do it by providing two reductions from FullPriv property. First, we change how we set the challenge public key to be sent to the adversary if $b = 1$. We do this change by relying on FullPriv property. Then, we have a game where if $b = 1$, the challenge public key is a fresh individual public key, and if $b = 0$, we output an aggregated public key.

Game_1: As it is shown in Figure 16, we change the way we set the challenge public key if $b = 1$. Another change that we have to make to have an indistinguishable game hop is changing $\mathcal{O}^{\mathtt{Chl}}$. Game_1 responds to $\mathcal{O}^{\mathtt{Chl}}(m)$ queries as $\sigma \leftarrow \mathsf{Sign}(sk_0, m)$ if $b = 1$. We show that this change is indistinguishable by building a FullPriv adversary as in Figure 17a. Both Game_0 and Game_1 plays SetPriv experiment identically if $b_{\mathsf{SetPriv}} = 0$, so our adversary. If $b_{\mathsf{SetPriv}} = 1$, our $\mathcal{A}_{\mathsf{FullPriv}}$ simulates SetPriv experiment to $\mathcal{A}_{\mathsf{SetPriv}}$ relying on FullPriv experiment. If the challenge bit of FullPriv experiment, $b_{\mathsf{FullPriv}} = 0$, then $\mathcal{A}_{\mathsf{FullPriv}}$ simulates Game_0. Otherwise, it simulates Game_1. Thus, if $\mathcal{A}_{\mathsf{SetPriv}}$ can distinguish Game_1

from Game_0 with non-negligible probability, $\mathcal{A}_{\mathsf{FullPriv}}$ can win $\mathsf{FullPriv}$ with non-negligible probability. In particular, $|\Pr[W_1] - \Pr[W_0]| \leq 1/2 \cdot \mu_{\mathsf{FullPriv}}(\lambda)$

Now, we argue that $\Pr[W_1]$ is negligible by relying on $\mathsf{FullPriv}$ property again. We show the $\mathsf{FullPriv}$ adversary in Figure 17b. One could observe that Game_1's challenge phase is already identical to $\mathsf{FullPriv}$ game. We send S_0 as the challenge set to $\mathsf{FullPriv}$ game. If $b_{\mathsf{FullPriv}} = 0$, our adversary simulates Game_1 for $b_{\mathsf{SetPriv}} = 0$. Otherwise, it simulates Game_1 for $b_{\mathsf{SetPriv}} = 1$. Thus, we guess b_{FullPriv} as b^*. Finally, we conclude that $\Pr[W_1] \leq \mu_{\mathsf{FullPriv}}(\lambda)$. □

$\mathsf{Exp}^{\mathsf{SetPriv}}_{\Pi,\mathcal{A}}(\lambda)$//$\mathsf{Game}_1$

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$(S_0, S_1) \leftarrow \mathcal{A}(SK, PK)$

if $b = 0$:

$(apk_0, \pi_0) \leftarrow \mathsf{KAg}(PK_{S_0}), pk \leftarrow apk_0$

if $b = 1$: $(sk_1, pk_1) \leftarrow \mathsf{Kg}(pp), pk \leftarrow pk_1$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}(\cdot)}(pk)$

return 1 **if** $b = b^*$

$\mathcal{O}^{\mathsf{Chl}}(m)$

if $b = 1$ **then**

  return $\sigma \leftarrow \mathsf{Sign}(sk, m)$

$\Sigma \leftarrow \{\mathsf{MulSign}(sk_i, PK_{S_0}, apk_0, m)\}_{sk_i \in SK_{S_0}}$

return $\sigma \leftarrow \mathsf{Combine}(PK_{S_0}, \pi_0, \Sigma)$

Fig. 16: Games of proof of Theorem 15

$\mathcal{A}_{\mathsf{FullPriv}}$ // For Game_0-Game_1 transition

$b_{\mathsf{SetPriv}} \leftarrow \{0,1\}$

if $b_{\mathsf{SetPriv}} = 1$ **then**

  $pp \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(\lambda), n \leftarrow \mathcal{A}_{\mathsf{SetPriv}}(pp)$

  $(SK, PK) \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(n)$

  $(S_0, S_1) \leftarrow \mathcal{A}_{\mathsf{SetPriv}}(SK, PK)$

  $pk \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(S_0)$

  $b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}_{\mathsf{FullPriv}}}_{\mathsf{SetPriv}}(pk), \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(b^*)$

else

  // Run $\mathsf{Exp}^{\mathsf{SetPriv}}_{\Pi}$ identically.

(a) $\mathsf{FullPriv}$ adversary for Game_0-Game_1 transition.

$\mathcal{A}_{\mathsf{FullPriv}}$ // For Game_1

$pp \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(\lambda), n \leftarrow \mathcal{A}_{\mathsf{SetPriv}}(pp)$

$(SK, PK) \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(n)$

$(S_0, S_1) \leftarrow \mathcal{A}_{\mathsf{SetPriv}}(SK, PK)$

$pk \leftarrow \mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(S_0)$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}_{\mathsf{FullPriv}}}_{\mathsf{SetPriv}}(pk)$

$\mathsf{Exp}^{\mathsf{FullPriv}}_{\Pi}(b^*)$

(b) $\mathsf{FullPriv}$ adversary for Game_1.

Fig. 17: $\mathsf{FullPriv}$ adversaries for proof of Theorem 15. For notational clarity, symbols used by both games have sub indexes to indicate which game they belong.

By Theorem 15, FullPriv property acquires another meaning. Our main aim is having FullPriv property under different models for BLSSign and SchnorrSign algorithms. However, having FullPriv property against any Sign() algorithm is useful even if we do not have FullPriv property for these algorithms, as we can use this fact to prove that the scheme has MemPriv property under the same model.

E.2 Proof of Theorem 5

Similar to the proof of Theorem 4 in Appendix E.1, we split Theorem 5 into two weaker claims and prove each of them.

MemPriv does not imply SetPriv. The following theorem shows that MemPriv does not imply SetPriv.

Theorem 16. *There exists a multi-signature scheme Π' which is* MemPriv*, but not* SetPriv*.*

Proof. Intuitively, we take a multi-signature scheme which has both properties. Then, on top of it, we build a new multi-signature scheme such that it has MemPriv, but does not have SetPriv. The main strategy for creating such multi-signature scheme is changing key spaces in a way that aggregated keys will reveal information about the size of underlying set. Let multi-signature scheme $\Pi = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{MulSign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{VfKAg})$ be a multi-signature scheme which has FullPriv property against a Sign algorithm and MemPriv property with secret key space $\mathcal{S}$ and public key space $\mathcal{P}$. We create Π' with secret key space $\mathcal{S} \times \mathbb{Z}_{2^\lambda}$ and public key space $\mathcal{P} \times \mathbb{Z}_{2^\lambda}$ for some security parameter λ as follows.

- $\Pi'.\mathsf{Kg}(pp)$: Runs $(sk, pk) \leftarrow \Pi.\mathsf{Kg}(pp)$ and outputs $((sk, 0), (pk, 0))$.
- $\Pi'.\mathsf{KAg}(PK')$: Let PK be the set of pk for $(pk, \cdot) \in PK'$. Then the algorithm runs $(apk, \pi) \leftarrow \Pi.\mathsf{KAg}(PK)$ and outputs $((apk, |PK'|), \pi)$.
- $\Pi'.\mathsf{MulSign}(sk'_i, PK', apk', m)$: Computes PK as in $\Pi'.\mathsf{KAg}()$. Then parses apk' and sk'_i as $(apk, \cdot)$ and $(sk_i, \cdot)$, respectively. Finally, outputs $s_i \leftarrow \Pi.\mathsf{MulSign}(sk_i, PK, apk, m)$.
- $\Pi'.\mathsf{Combine}(PK', \pi, \{s_i\}_{pk'_i \in PK'})$: Computes PK as in $\Pi'.\mathsf{KAg}()$. Then, computes and outputs $\sigma \leftarrow \Pi.\mathsf{Combine}(PK, \pi, \{s_i\}_{pk_i \in PK})$.
- $\Pi'.\mathsf{VfKAg}(PK', apk', \pi)$: Parses/computes PK and apk accordingly and outputs $b \leftarrow \Pi.\mathsf{VfKAg}(PK, apk, \pi)$.
- $\Pi'.\mathsf{Vf}(apk', \sigma, m)$: Parses apk' as $(apk, \cdot)$ and then outputs the bit $b \leftarrow \Pi.\mathsf{Vf}(apk, \sigma, m)$.

Not SetPriv*.* An adversary $\mathcal{A}_{\mathsf{SetPriv}}$ can easily win SetPriv as follows. $\mathcal{A}_{\mathsf{SetPriv}}$ plays the game with the inputs $n \leftarrow 2$, $S_0 \leftarrow \{1\}$, and $S_1 \leftarrow \{1, 2\}$ without any further oracle queries. The returned public key apk' from the game is in the form of $(\cdot, r)$ where $r = |S_b|$. As two sets have different sizes, $\mathcal{A}_{\mathsf{SetPriv}}$ wins the game with probability 1.

MemPriv. We show that the multi-signature scheme Π' is MemPriv if Π is MemPriv. To do that, we build an adversary $\mathcal{B}$ against $\mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi}$ using an adversary $\mathcal{B}'$ against $\mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi'}$ as in Figure 18. $\mathcal{B}$ simulates the view of $\mathcal{B}'$ perfectly without knowing the challenge bit b_{MemPriv}. As $|S_0| = |S_1|$, the size of the challenge set does not reveal any information that $\mathcal{B}'$ does not have. Thus, $\mathcal{B}$ wins whenever $\mathcal{B}'$ wins. □

$\mathcal{B}$

$pp \leftarrow \mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi}(\lambda), n \leftarrow \mathcal{B}'(pp)$

$(SK, PK) \leftarrow \mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi}(n), (SK', PK') \leftarrow \{(sk_i, (pk_i, 0))\}_{pk_i \in PK}$

$(S_0, S_1) \leftarrow \mathcal{B}'(SK', PK'), apk \leftarrow \mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi}(S_0, S_1)$

$b^* \leftarrow \mathcal{B}'^{\mathcal{O}^{\mathtt{Ch1}'}}((apk, |S_0|))$

$\mathsf{Exp}^{\mathsf{MemPriv}}_{\Pi}(b^*)$

$\mathcal{O}^{\mathtt{Ch1}'}(m)$

return $\sigma \leftarrow \mathcal{O}^{\mathtt{Ch1}}(m)$

Fig. 18: Adversary $\mathcal{B}$ against $\mathsf{Exp}^{\mathsf{MemPriv}}_{\pi}$ which simulates $\mathsf{Exp}^{\mathsf{MemPriv}}_{\pi'}$ against $\mathcal{B}'$. For notational clarity, the common symbols of both games are differentiated with apostrophe (e.g. $\mathcal{O}^{\mathtt{Ch1}}$ and $\mathcal{O}^{\mathtt{Ch1}'}$).

SetPriv implies MemPriv. The following theorem shows that FullPriv implies SetPriv.

Theorem 17. *If a multi-signature scheme Π has the* SetPriv *property, then it has the* MemPriv *property.*

Proof. We show this implication by building an adversary $\mathcal{A}_{\mathsf{SetPriv}}$ on top of an adversary $\mathcal{A}_{\mathsf{MemPriv}}$ in Figure 19. As the original games in Figure 6 shows, SetPriv and MemPriv games only differ for the additional abort condition in MemPriv. Thus, the only additional step that $\mathcal{A}_{\mathsf{SetPriv}}$ performs beyond forwarding messages is checking the validity of challenge sets for MemPriv game. This step is necessary to simulate MemPriv game indistinguishably. Other than that, SetPriv game is already identical to MemPriv game. □

E.3 Relations between KPK and AbOPK Models

For completeness, we state the formal theorem for the relation that KPK model is strictly stronger than AbOPK model. The main difference between different models is the amount of revealed individual public keys. It is straightforward to show that KPK implies AbOPK and the reverse does not hold. It is also easy to see that the implication FullPriv ⇒ SetPriv ⇒ MemPriv hold in the AbOPK model as well. For the following analysis we therefore always prove only the strongest privacy property that is achieved by each scheme.

$\mathcal{A}_{\mathsf{SetPriv}}$

$pp \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(\lambda), n \leftarrow \mathcal{A}_{\mathsf{MemPriv}}(pp), (SK, PK) \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(n)$

$(S_0, S_1) \leftarrow \mathcal{A}_{\mathsf{MemPriv}}(SK, PK)$

abort if $|S_j \setminus S_{1-j}| \neq 1$ **for** $j \in \{0, 1\}$

$pk \leftarrow \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(S_0, S_1), b^* \leftarrow \mathcal{A}_{\mathsf{MemPriv}}^{\mathcal{O}_{\mathsf{SetPriv}}^{\mathtt{Ch1}}}(pk), \mathsf{Exp}_{\Pi}^{\mathsf{SetPriv}}(b^*)$

$\mathcal{O}_{\mathsf{SetPriv}}^{\mathtt{Ch1}}(m)$

return $\sigma \leftarrow \mathcal{O}_{\mathsf{MemPriv}}^{\mathtt{Ch1}}(m)$

Fig. 19: Adversary for the SetPriv $\Rightarrow$ MemPriv implication.

Theorem 18 (KPK $\Rightarrow$ AbOPK)**.** *For any* MSvKA *scheme and* X $\in$ {FullPriv, SetPriv, MemPriv} *it holds that* X-KPK *implies, and is strictly stronger than,* X-AbOPK*.*

Proof. This proof needs the correctness of two sub-claims which we investigate below.

X-KPK Implies X-AbOPK This implication can easily be proven by showing that if a multi-signature scheme is not X-AbOPK, then it is not X-KPK. We can answer the queries of an X-AbOPK adversary using an X-KPK challenger. We can output the guess bit b^* of X-AbOPK adversary to X-KPK challenger directly. The only difference between the models remaining is the abort case in AbOPK. However, as AbOPK abort case is more restricted than KPK, we win against X-KPK challenger whenever X-AbOPK adversary wins against us.

X-AbOPK Does Not Imply X-KPK We do not provide a proof for this statement as existing schemes which we investigate in Section 6 already shows it. □

F Proofs for randBLS-1 and randBLS-2

This section presents the unforgeability and privacy proofs of our randBLS-1 and randBLS-2 schemes.

F.1 Proof of MSvKA-UNF-1 Security (Theorem 6)

As the proof sketch explains, we build an algorithm $\mathcal{B}$ that will play the role of challenger using a co-CDH instance. However, we will first apply a few changes to the unforgeability game through a sequence of games so that $\mathcal{B}$ can simulate the resulting game using the co-CDH instance.

Proof. Game_1: Changes from Game_0 is presented in Figure 20. This game adds a counter to count the H_0 queries and changes the way we answer the H_0 queries so that we know the discrete logarithms of output values. The counter will be used to choose which H_0 query to set an external value A in Game_2. The discrete logarithm values will be used to answer multi-signature queries without knowing

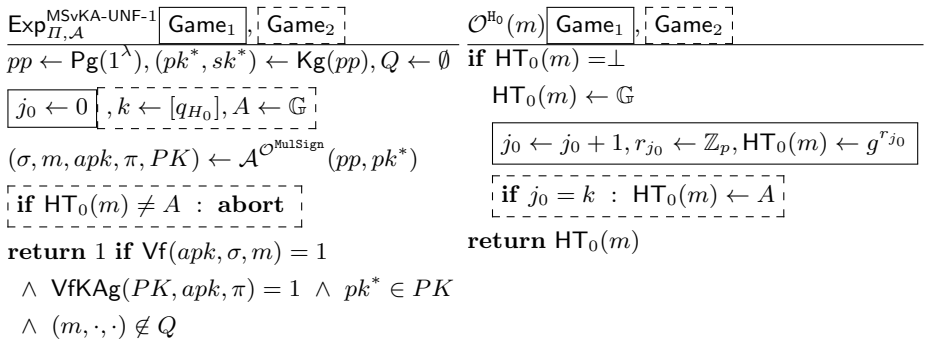


Fig. 20: Games of randBLS-1 Unforgeability Proof

the secret key. Both changes are made to serve the upcoming proof steps and are invisible to the adversary, $\Pr[W_1] = \Pr[W_0]$.

Game_2: In this game, we guess the message that the adversary will perform a forgery and we abort if the forgery was not for that message. We fix the H_0 query for the guessed message to a random value A. This value will be replaced by A value of a co-CDH instance in the upcoming steps. As our guess is independent from the adversary's outputs and we simulate Game_1 perfectly until the abort condition, $\Pr[W_2] = \Pr[W_1]/q_{H_0}$.

Game_3: The previous game hops were similar with the original BLS signature's unforgeability proof. However, this game hop is special to multi-signature case and it is shown in Figure 21. In this game hop, we change how we answer H_1 queries, and it will be useful to have a point that we can fork on in later steps. Namely, whenever there is a new H_1 query for a set that contains pk^*, we set hash table not only for the queried public key, but also for the other public keys from the set. At the end, we set the hash table for pk^*. Later, when we fork from the point where we set the hash table for pk^*, we will be sure that the hash table has the identical values for all other public keys in the set. As these changes are all internal and we keep simulating all queries as uniformly random values from $\mathbb{Z}_p$, $\Pr[W_3] = \Pr[W_2]$.

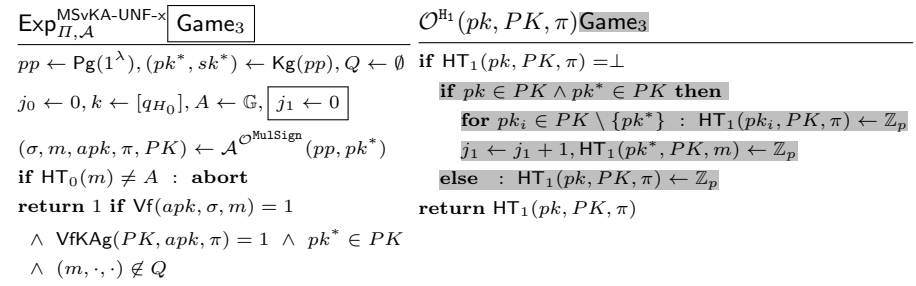


Fig. 21: Games of randBLS-1 Unforgeability Proof

We will build our co-CDH adversary, but we need to explain the algorithm $\mathcal{B}$ first. $\mathcal{B}$ takes a co-CDH instance as an input and randomness $(\rho, h_1, ..., h_{q_{H_1}})$. Then, similar with the proof original BLS signature, it sets pk^* as $\hat{B}$. Also, $\mathcal{B}$ does not sample a group element A and uses the input A value. Since we do not know the corresponding secret key for $\hat{B}$, we need to answer signing queries without a secret key. We can do this easily by using r_{j_0} values from H_0 queries, except the $\mathsf{H}_0(m) = A$ query. However, we do not have to simulate a signing query for this message as we know that there will not be a signing query for this message by Game_2. Thus, we simulate signing queries of Game_3 perfectly. While we answer H_0 queries identically to Game_3, we have a slight change for H_1 queries. We change how we set $\mathsf{H}_1(pk^*, PK, \pi)$ queries. Instead of sampling a fresh random value from the randomness ρ, we use h_{j_1} values. By doing that. we will be able to fork from the points that we set this query. As this change is also indistinguishable, we simulate H_1 queries perfectly, too. Finally, since algorithm $\mathcal{B}$ simulates Game_3 perfectly, we conclude that $\Pr\Big[res \leftarrow \mathcal{B}((A, B, \hat{B}), (\rho, h_1, ..., h_{q_{H_1}})) : res \neq (\emptyset, \perp)\Big] = \Pr[W_3]$.

$\underline{\mathcal{B}((\mathcal{BG}, A, B, \hat{B}), (\rho, h_1, ..., h_{q_{H_1}}))}$
$pp \leftarrow \mathcal{BG}, pk^* \leftarrow \hat{B}, Q \leftarrow \emptyset$
$j_0 \leftarrow 0, k \leftarrow [q_{H_0}], j_1 \leftarrow 0$
$(\sigma, m, apk, \pi, PK) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathtt{MulSign}}}(pp, pk^*)$
if $\mathsf{HT}_0(m) \neq A$: **abort**
return $(\emptyset, \perp)$ **if** $\mathsf{Vf}(apk, \sigma, m) = 0$
$\vee$ $\mathsf{VfKAg}(PK, apk, \pi) = 0$ $\vee$ $pk^* \notin PK$
$\vee$ $(m, \cdot, \cdot) \in Q$
Find j_f s.t. $\mathsf{HT}_1(pk^*, PK, \pi) = h_{j_f}$
else return $(J = \{j_f\}, \{\sigma, h_{j_f}\})$

$\underline{\mathcal{O}^{\mathtt{H}_1}(pk, PK, \pi)}$
if $\mathsf{HT}_1(pk, PK, \pi) = \perp$
if $pk \in PK \wedge pk^* \in PK$ **then**
for $pk_i \in PK \setminus \{pk^*\}$:
$\mathsf{HT}_1(pk_i, PK, \pi) \leftarrow \mathbb{Z}_p$
$j_1 \leftarrow j_1 + 1, \mathsf{HT}_1(pk^*, PK, m) \leftarrow h_{j_1}$
else : $\mathsf{HT}_1(pk, PK, \pi) \leftarrow \mathbb{Z}_p$
return $\mathsf{HT}_1(pk, PK, \pi)$

$\underline{\mathcal{O}^{\mathtt{H}_0}(m)}$
if $\mathsf{HT}_0(m) = \perp$
$j_0 \leftarrow j_0 + 1, r_{j_0} \leftarrow \mathbb{Z}_p, \mathsf{HT}_0(m) \leftarrow g^{r_{j_0}}$
if $j_0 = k$: $\mathsf{HT}_0(m) \leftarrow A$
return $\mathsf{HT}_0(m)$

$\underline{\mathcal{O}^{\mathtt{MulSign}}(PK_i, apk_i, \pi_i, m_i)}$
if $pk^* \notin PK_i \vee \mathsf{VfKAg}(PK_i, apk_i, \pi_i) \neq 1$
then return $\perp$
$Q \leftarrow Q \cup \{(m_i, PK_i, apk_i)\}$
if $\mathsf{HT}_0(m_i) = \perp$: $\mathsf{H}_0(m_i)$
$s \leftarrow B^{r_{m_i}}$ // W.l.o.g. let $\mathsf{H}_0(m_i) = g^{r_{m_i}}$
return s

Fig. 22: Algorithm $\mathcal{B}$

Now, we can explain our co-CDH adversary in Figure 23. $\mathcal{A}_{\text{co-cdh}}$ takes a co-CDH instance $(\mathcal{BG}, A, B, \hat{B})$ and runs $\mathcal{GF}$ with algorithm $\mathcal{B}$ for the input

$(\mathcal{BG}, A, B, \hat{B})$. First, we explain how the successful output of $\mathcal{GF}_{\mathcal{B}}$ can be used to solve the co-CDH instance. For a successful output, we have two signatures σ and σ' such that they are valid for the aggregated public keys apk and apk'. In particular, we know that $e(A, apk) = e(\sigma, \hat{g})$ and $e(A, apk') = e(\sigma', \hat{g})$. Furthermore, we also know that two runs of algorithm $\mathcal{B}$ were identical up to the point j_f which is a point in a H_1 query for some (pk, PK, π). As the runs were identical up to j_f, we know that the forgery is performed for the same (pk, PK, π) and we also know that $\mathsf{H}_1(pk, PK, \pi)$ queries are set to the identical values except $pk = pk^*$. For $pk = pk^*$, $\mathsf{H}_1(pk, PK, \pi)$ queries are set to a and a' for two runs. It means that $apk/apk' = (pk^*)^{a-a'}$ as all other public keys cancel each other. On the other hand, we know that a and a' are not equal by Forking Lemma, so pk^* values cannot be canceled out. By using signature verification equations, we know that $e(A, apk/apk') = e(\sigma/\sigma', \hat{g}) = e(A, (pk^*)^{a-a'})$. Thus, we know that $(\sigma/\sigma')^{1/(a-a')}$ is the solution to the co-CDH instance. Finally, we conclude that $\mathcal{A}_{\text{co-cdh}}$ succeeds whenever $\mathcal{GF}_{\mathcal{B}}$ succeeds, and $\mathcal{GF}_{\mathcal{B}}$ succeeds with the probability $\Pr[W_3]/8$ by Generalized Forking Lemma if $p > 8q_H/\Pr[W_2]$. Thus, $\mathcal{A}_{\text{co-cdh}}$ succeeds with the probability $\Pr[W_0]/(8 \cdot q_{H_0})$. □

$\mathcal{A}_{\text{co-cdh}}(\mathcal{BG}, A, B, \hat{B})$

$res \leftarrow \mathcal{GF}_{\mathcal{B}}(pp, A, B, \hat{B})$, **if** $res = \perp$: **abort**

Parse res as (out, out'), out as $\{\sigma, a\}$ and out' as $\{\sigma', a'\}$

return $(\sigma/\sigma')^{1/(a-a')}$

Fig. 23: Algorithm $\mathcal{A}_{\text{co-cdh}}$

F.2 Proof of FullPriv-KPK Property (Theorem 7)

We prove the theorem through a short sequence of games, where Game_0 denotes the game played against the original randBLS-1. In the final game, the challenge public key and challenge oracle are entirely independent of the adversary's output S^* as well as of the challenge bit b, and thus the adversary can win the FullPriv-KPK game only with probability 1/2. Our main strategy is that setting apk^* to a freshly sampled public key and answering $\mathcal{O}^{\mathtt{Chl}}$ queries related to apk^* using the corresponding secret key and BLSSign algorithm. By doing that, we will end up in a game where regardless of what the challenge bit b is, all queries are answered using a freshly sampled key pair and BLSSign algorithm.

Proof. Game_1: Figure 24 shows the changes between Game_0 and Game_1. To achieve the main goal we explained above, we need several steps. The first step is simulating the FullPriv-KPK game without using key aggregation and multi-signature algorithms. In Game_1, we set an *aggregated secret key* ask^* and apk^* is just computed as a function of ask^*, exponentiation. Using ask^*, $\mathcal{O}^{\mathtt{Chl}}$ queries

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\mathsf{randBLS\text{-}1},\mathcal{A}}(\lambda)$ Game_1

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp),$ **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$S^* \leftarrow \mathcal{A}(SK, PK)$

if $b = 0$ **then**

$\pi^* \leftarrow \{0,1\}^\lambda, i \leftarrow S^*, r \leftarrow \mathsf{H}_1(pk_i, PK_{S^*}, \pi^*)$

$ask^* \leftarrow sk_i \cdot r + \sum_{j\in S^*\setminus\{i\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*}, \pi^*)$

$apk^* \leftarrow \hat{g}^{ask^*}, pk^* \leftarrow apk^*$

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 $b = 1$ **then**

  return $\sigma \leftarrow \mathsf{BLSSign}(sk, m)$

$\sigma \leftarrow \mathsf{BLSSign}(ask^*, m)$

return σ

Fig. 24: Changes between Game_0 and Game_1 of $\mathsf{FullPriv} - \mathsf{KPK}$ experiment.

can be answered using $\mathsf{BLSSign}$ algorithm instead of $\mathsf{MulSign}()$ and $\mathsf{Combine}()$ algorithms even in the case that $b = 0$. The change in the original experiment is invisible to the adversary as we just perform an additional computation that does not impact the output. Furthermore, it is straightforward to observe that the change in $\mathcal{O}^{\mathtt{Chl}}$ is also perfectly indistinguishable to the adversary as the resulting σ value due to the change in Game_1 is identical to the σ value which is computed in Game_0.Thus, we conclude that $\Pr[W_1] = \Pr[W_0]$.

In Game_1, we have a game in which apk^* is computed from a secret key ask^* and $\mathcal{O}^{\mathtt{Chl}}$ queries are answered using the same secret key and $\mathsf{BLSSign}$. In the following steps, we aim to show that the secret key ask^* is indistinguishable from a freshly sampled secret key.

Game_2: To show that ask^* is indistinguishable from a freshly sampled secret key, we need to show that we do not reveal information about π^* to the adversary and it will take two steps. We start by adding an abort condition in this game. Namely, if there was a H_1 query before setting the challenge set S^* which contains π^*, then Game_2 aborts. Otherwise, it plays Game_1 identically. We show that this change is indistinguishable to the adversary as it only occurs with a negligible probability. π^* is a freshly sampled random value, so the probability that there is a previous query for π^* is $q/2^\lambda$. Thus, we conclude that $|\Pr[W_2] - \Pr[W_1]| \leq q/2^\lambda$.

Game_3: In this game, we show that the adversary can learn the hash values related to π^* only with a negligible probability. In Game_2, we already achieve a game hop that the adversary did not learn any hash values related to π^* until the challenge phase. To show that such queries are unlikely to occur even after the challenge phase, we add another abort condition in $\mathcal{O}^{\mathtt{H}_1}$. According to this condition, we abort if there are any hash queries related to π^* after the challenge phase. To check whether a query is before or after the challenge phase, we add the flag $\mathsf{Chlinit}$ which is set to 0 by default and it is set to 1 in the challenge

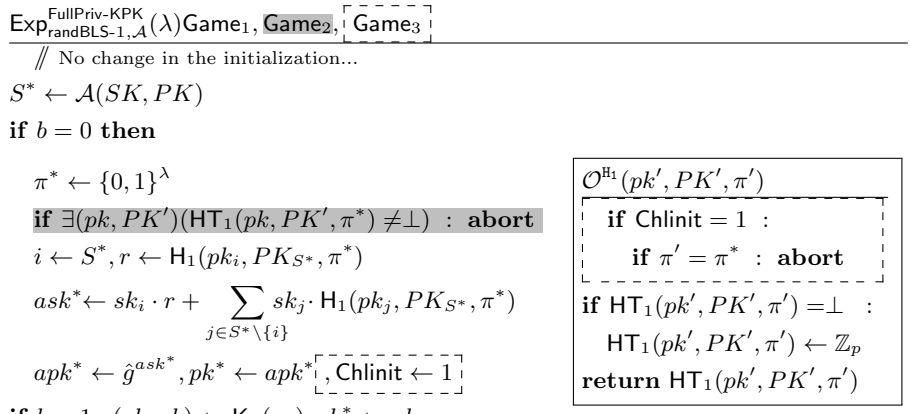


Fig. 25: Games Game_1, Game_2, and Game_3 of FullPriv-KPK proof.

phase. We note that the game itself makes hash queries related to π^* to compute the aggregated secret key ask^*. However, the abort condition is only valid after the flag $\mathsf{Chlinit}$ is set. Thus, these queries do not cause a trivial abort.

We argue that the abort condition that we add holds only with a negligible probability as follows. By Game_2, we already know that there is no such random oracle query until the challenge set has been set. After that, the only value that we use which is dependent on π^* is ask^*. As we know that the adversary did not call the random oracle for π^* before we set ask^*, $r = \mathsf{H}_1(pk_i, PK_{S^*}, \pi^*)$ is uniformly random to the adversary. It follows that $sk_i \cdot r$ is uniformly random and finally the ask^* value is uniformly random. Thus, ask^* does not reveal any information about π^*. Now, we know that the oracles of Game_3 except $\mathcal{O}^{\mathsf{H}_1}$ do not reveal any information about π^*. Hence, the adversary can make an $\mathcal{O}^{\mathsf{H}_1}$ query only with the negligible probability, $|\Pr[W_3] - \Pr[W_2]| \leq q/2^\lambda$.

Game_4: Finally, we can start changing the way we compute ask^* to show its indistinguishability from a freshly sampled key. We change how we compute ask^*. We change $sk_i \cdot r$ to r^* for a uniformly random r^*. As r is uniformly random in $\mathbb{Z}_p$, $sk_i \cdot r$ is perfectly indistinguishable from r^*, so $\Pr[W_4] = \Pr[W_3]$.

Game_5: In this game, we sample ask^* uniformly from $\mathbb{Z}_p$. As r^* is uniformly random in $\mathbb{Z}_p$, $r^* + \sum_{j \in S^* \setminus \{i\}} sk_j \cdot \mathsf{H}_1(pk_j, PK, \pi^*)$ is perfectly indistinguishable from $ask^* \leftarrow \mathbb{Z}_p$, so $\Pr[W_5] = \Pr[W_4]$.

Game_6: Finally, we initialize key pair (ask^*, apk^*) as fresh individual key pair. While Game_5 samples ask^* from $\mathbb{Z}_p$, $\mathsf{Kg}(pp)$ samples it from $\mathbb{Z}_p^*$, which is a negligible difference. In particular, the advantage of the adversary against Game_5 is $|\Pr[W_6] - \Pr[5]| \leq 1/p$.

In the final game Game_6, challenge public key pk^* and corresponding signatures are computed as individual BLS signatures independently form the chal-

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\mathsf{randBLS\text{-}1},\mathcal{A}}(\lambda)$ Game_4, $\boxed{\mathsf{Game}_5}$, $\boxed{\mathsf{Game}_6}$

// No change in the initialization...

$S^* \leftarrow \mathcal{A}(SK, PK)$

if $b = 0$ **then**

$\pi^* \leftarrow \{0,1\}^\lambda$

if $\exists (pk, PK')(\mathsf{HT}_1(pk, PK', \pi^*) \neq \perp)$: **abort**

$i \leftarrow S^*, r^* \leftarrow \mathbb{Z}_p, ask^* \leftarrow r^* + \sum_{j \in S^* \setminus \{i\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*}, \pi^*)$

$\boxed{ask^* \leftarrow \mathbb{Z}_p,} apk^* \leftarrow \hat{g}^{ask^*}, \boxed{(ask^*, apk^*) \leftarrow \mathsf{Kg}(pp),} pk^* \leftarrow apk^*, \mathsf{Chlinit} \leftarrow 1$

if $b = 1 : (sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Ch1}}(\cdot)}(pk^*)$, **return** 1 **if** $b = b^*$

Fig. 26: Games Game_4, Game_5, and Game_6 of FullPriv-KPK proof.

lenge bit b and challenge set S^*. Thus, the adversary has no more advantage than a random guess, so $\Pr[W_6] = 1/2$. Hence, using the sequences of games, we conclude that

$$\left| \mathsf{Exp}^{\mathsf{FullPriv\text{-}KPK}}_{\mathsf{randBLS\text{-}1},\mathcal{A}}(\lambda) - 1/2 \right| \leq 2 \cdot q/2^\lambda + 1/p$$

□

F.3 Proof of randBLS-1's Key Binding Property (Theorem 8)

The proof strategy is similar to the proof in Appendix C for the key collision-freeness. We show that *apk*'s are distributed uniformly using the random oracle, and collisions occur with negligible probabilities in 3 different cases.

Proof. A proof in the random oracle model is provided as follows. A hash query $\mathsf{H}_1(pk, PK, \pi)$ is answered as follows.

- if $pk \in PK$ and this is the first query for such PK and π, then we set $\mathsf{H}_1(pk', PK, \pi) \leftarrow \mathbb{Z}_p$ for all $pk' \in PK$.
- if $pk \in PK$ and there is a previous query for such PK and π, we answer the query using the previously set value.
- if $pk \notin PK$, simulate the query as a uniformly random choice from $\mathbb{Z}_p$.

We consider three cases for the winning event of $\mathcal{A}$.

Event W_1: There does not exist a query for PK_0 and π_0.
- Let $apk_1 \leftarrow \mathsf{KAg}(PK_1)$ which has been set by the previous hash query for PK_1. As there is no query for PK_0 yet, it is simulated by the verification as $apk_0 \leftarrow pk_0^{\mathsf{H}_1(pk_0, PK, \pi_0)} \cdot Z$ for some $pk_0 \in PK_0$ and for some Z. As $pk_0 \in \mathbb{G}^*$ and $\mathsf{H}_1(pk_0, PK_0, \pi_0) \leftarrow \mathbb{Z}_p$ is uniformly random value, apk_0 is uniformly random choice from $\mathbb{G}$. Hence, $\Pr[W_1] = 1/p$

Event W_2: there does not exist a query for PK_1.
- Similar to the first case. $\Pr[W_2] = 1/p$.

Event W_3**:** there exists queries for both PK_0 and PK_1.

– Query for (PK_b, π_b) creates a uniformly random apk_b in $\mathbb{G}$. Hence, for q random oracle queries, $\Pr[W_3] \leq q^2/p$

It is obvious that

$$\mathsf{Exp}^{\mathsf{MSvKA\text{-}KEY\text{-}BND}}_{\Pi,\mathcal{A}} \leq \Pr[W_1] + \Pr[W_2] + \Pr[W_3]$$

Thus, we conclude that $\mathsf{Exp}^{\mathsf{MSvKA\text{-}KEY\text{-}BND}}_{\Pi,\mathcal{A}} \leq (q^2+2)/p$. □

F.4 Proof of randBLS-2's FullPriv-KPK Property (Theorem 9)

Proof. The proof is identical to the proof of randBLS-1's FullPriv-KPK property except that we use the related Sign instead of BLSSign.

Game_1: Let $ask^* \leftarrow \sum_{sk_j \in SK_{S^*}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*}, \pi^*)$. This game is identical to Game_0 except the way we answer $\mathcal{O}^{\mathtt{Chl}}$ queries. If $b = 1$, the oracle queries are answered as they used to be. If $b = 0$, instead of creating a multi-signature, we compute the output as $\sigma \leftarrow \mathsf{Sign}(ask^*, m)$. This change does not affect the adversary's view, so Game_1 is perfectly indistinguishable from Game_0.

$$\Pr[W_1] = \Pr[W_0]$$

The remaining part of the proof is identical to the related part of the proof of Theorem 7.

G Analysis of Existing Schemes

In this section, we provide the detailed proofs of privacy properties of two existing multi-signatures scheme in Section 6.2.

G.1 Proof of FullPriv-AbOPK for BLS-dMS (Theorem 10)

We show that the BLS-dMS scheme satisfies FullPriv-AbOPK privacy, i.e., produces multi-signatures and aggregate keys that are indistinguishable from individual ones (derived via BLSSign, from Sec. 2), if at least one public key remains unknown to the adversary.

Proof. We prove the theorem through a short sequence of games, where Game_0 denotes the game played against the original BLS-dMS. In the final game, the challenge public key and challenge oracle are entirely independent of the adversary's output S^* as well as of the challenge bit b, and thus the adversary can win the FullPriv-AbOPK game only with probability 1/2. Our main strategy is that setting apk^* to a freshly sampled public key and answering $\mathcal{O}^{\mathtt{Chl}}$ queries related to apk^* using the corresponding secret key and BLSSign algorithm. By doing that, we will end up in a game where regardless of what the challenge bit b is, all queries are answered using a freshly sampled key pair and BLSSign algorithm.

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{BLS\text{-}dMS},\mathcal{A}}(\lambda)$ Game_1

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$S^* \leftarrow \mathcal{A}(SK, PK)$, **abort if** $1 \notin S_j$ **for** $j \in \{0,1\}$

if $b = 0$ **then**

$r \leftarrow \mathsf{H}_1(pk_1, PK_{S^*})$

$ask^* \leftarrow sk_1 \cdot r + \sum_{j\in S^*\setminus\{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$apk^* \leftarrow \hat{g}^{ask^*}, pk^* \leftarrow apk^*$

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 $b = 1$ **then**

  return $\sigma \leftarrow \mathsf{BLSSign}(sk, m)$

$\sigma \leftarrow \mathsf{BLSSign}(ask^*, m)$

return σ

Fig. 27: Changes between Game_0 and Game_1 of FullPriv-AbOPK proof of BLS-dMS. Gray highlight is used to indicate the changed code between games.

Game_1: Figure 27 shows the changes between Game_0 and Game_1. To achieve the main goal we explained above, we need several steps. The first step is simulating FullPriv-AbOPK game without using key aggregation and multi-signature algorithms. In Game_1, we set an *aggregated secret key* ask^* and apk^* is just computed as a function of ask^*, exponentiation. Using ask^*, $\mathcal{O}^{\mathtt{Chl}}$ queries can be answered using BLSSign algorithm instead of MulSign() and Combine() algorithms even in the case that $b = 0$. The change in the original experiment is invisible to the adversary as we just perform an additional computation that does not impact the output. Furthermore, it is straightforward to observe that the change in $\mathcal{O}^{\mathtt{Chl}}$ is also perfectly indistinguishable to the adversary as the resulting σ value due to the change in Game_1 is identical to the σ value which is computed in Game_0.Thus, we conclude that $\Pr[W_1] = \Pr[W_0]$.

In Game_1, we have a game in which apk^* is computed from a secret key ask^* and $\mathcal{O}^{\mathtt{Chl}}$ queries are answered using the same secret key and BLSSign. In the following steps, we aim to show that the secret key ask^* is indistinguishable from a freshly sampled secret key.

Game_2: To show that ask^* is indistinguishable from a freshly sampled secret key, we need to show that we do not reveal information about pk_1 to the adversary and it will take two steps. We start by adding an abort condition in this game. Namely, if there was a H_1 query before setting the challenge set S^* which contains pk_1, then Game_2 aborts. Otherwise, it plays Game_1 identically. We show that this change is indistinguishable to the adversary as it only occurs with a negligible probability. pk_1 is a freshly sampled random value, so the probability that there is a previous query for pk_1 is q/p. Thus, we conclude that $|\Pr[W_2] - \Pr[W_1]| \leq q/p$.

Game_3: In this game, we show that the adversary can learn the hash values related to pk_1 only with a negligible probability. In Game_2, we already achieve a

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{BLS\text{-}dMS},\mathcal{A}}(\lambda)\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3$

// No change in the initialization...

$S^* \leftarrow \mathcal{A}(SK \setminus \{sk_1\}, PK \setminus \{pk_1\}), \textbf{abort if } 1 \notin S_j \textbf{ for } j \in \{0,1\}$

if $b = 0$ **then**

if $\exists(PK')(\mathsf{HT}_1(pk_1, PK') \neq \perp)$: **abort**

$r \leftarrow \mathsf{H}_1(pk_1, PK_{S^*})$

$ask^* \leftarrow sk_1 \cdot r + \sum_{j \in S^* \setminus \{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$apk^* \leftarrow \hat{g}^{ask^*}, pk^* \leftarrow apk^*, \mathsf{Chlinit} \leftarrow 1$

if $b = 1 : (sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}(\cdot)}(pk^*), \textbf{return } 1 \textbf{ if } b = b^*$

$\mathcal{O}^{\mathsf{H}_1}(pk', PK')$

if $\mathsf{Chlinit} = 1$:

if $pk' = pk_1$: **abort**

if $\mathsf{HT}_1(pk', PK') = \perp$:

$\mathsf{HT}_1(pk', PK') \leftarrow \mathbb{Z}_p$

return $\mathsf{HT}_1(pk', PK')$

Fig. 28: Games Game_1, Game_2, and Game_3 of FullPriv-AbOPK proof of BLS-dMS.

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{BLS\text{-}dMS},\mathcal{A}}(\lambda)\mathsf{Game}_4, \mathsf{Game}_5, \mathsf{Game}_6$

// No change in the initialization...

$S^* \leftarrow \mathcal{A}(SK \setminus \{sk_1\}, PK \setminus \{pk_1\}), \textbf{abort if } 1 \notin S_j \textbf{ for } j \in \{0,1\}$

if $b = 0$ **then**

if $\exists(PK')(\mathsf{HT}_1(pk_1, PK') \neq \perp)$: **abort**

$r^* \leftarrow \mathbb{Z}_p, ask^* \leftarrow r^* + \sum_{j \in S^* \setminus \{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$ask^* \leftarrow \mathbb{Z}_p, apk^* \leftarrow \hat{g}^{ask^*}, (ask^*, apk^*) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow apk^*, \mathsf{Chlinit} \leftarrow 1$

if $b = 1 : (sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}(\cdot)}(pk^*), \textbf{return } 1 \textbf{ if } b = b^*$

Fig. 29: Games Game_4, Game_5, and Game_6 of FullPriv-KPK proof of BLS-dMS.

game hop that the adversary did not learn any hash values related to pk_1 until the challenge phase. To show that such queries are unlikely to occur even after the challenge phase, we add another abort condition in $\mathcal{O}^{\mathsf{H}_1}$. According to this condition, we abort if there is any hash queries related to pk_1 after the challenge phase. To check whether a query is before or after the challenge phase, we add the flag $\mathsf{Chlinit}$ which is set to 0 by default and it is set to 1 in the challenge phase. We note that the game itself makes hash queries related to pk_1 to compute the aggregated secret key ask^*. However, the abort condition is only valid after the flag $\mathsf{Chlinit}$ is set. Thus, these queries do not cause a trivial abort.

We argue that the abort condition that we add holds only with a negligible probability as follows. By Game_2, we already know that there is no such random oracle query until the challenge set has been set. After that, the only value that we use which is dependent on pk_1 is ask^*. As we know that the adversary did

not call the random oracle for pk_1 before we set ask^*, $r = \mathsf{H}_1(pk_1, PK_{S^*})$ is uniformly random to the adversary. It follows that $sk_1 \cdot r$ is uniformly random and finally the ask^* value is uniformly random. Thus, ask^* does not reveal any information about π^*. Now, we know that the oracles of Game_3 except $\mathcal{O}^{\mathtt{H}_1}$ do not reveal any information about pk_1. Hence, the adversary can make an $\mathcal{O}^{\mathtt{H}_1}$ query only with the negligible probability, $|\Pr[W_3] - \Pr[W_2]| \leq q/p$.

Game_4: Finally, we can start changing the way we compute ask^* to show its indistinguishability from a freshly sampled key. We change how we compute ask^*. We change $sk_1 \cdot r$ to r^* for a uniformly random r^*. As r is uniformly random in $\mathbb{Z}_p$, $sk_1 \cdot r$ is perfectly indistinguishable from r^*, so $\Pr[W_4] = \Pr[W_3]$.

Game_5: In this game, we sample ask^* uniformly from $\mathbb{Z}_p$. As r^* is uniformly random in $\mathbb{Z}_p$, $r^* + \sum_{j \in S^* \setminus \{i\}} sk_j \cdot \mathsf{H}_1(pk_j, PK)$ is perfectly indistinguishable from $ask^* \leftarrow \mathbb{Z}_p$, so $\Pr[W_5] = \Pr[W_4]$.

Game_6: Finally, we initialize key pair (ask^*, apk^*) as fresh individual key pair. While Game_5 samples ask^* from $\mathbb{Z}_p$, $\mathsf{Kg}(pp)$ samples it from $\mathbb{Z}_p^*$, which is a negligible difference. In particular, the advantage of the adversary against Game_5 is $|\Pr[W_6] - \Pr[5]| \leq 1/p$.

In the final game Game_6, challenge public key pk^* and corresponding signatures are computed as individual BLS signatures independently form the challenge bit b and challenge set S^*. Thus, the adversary has no more advantage than a random guess, so $\Pr[W_6] = 1/2$. Hence, using the sequences of games, we conclude that

$$\left|\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{BLS\text{-}dMS},\mathcal{A}}(\lambda) - 1/2\right| \leq (2 \cdot q + 1)/p$$

□

G.2 Proof of FullPriv-AbOPK for MuSig - Theorem 11

Just as BLS-dMS, the Schnorr multi-signature scheme in Construction 6 also achieves FullPriv-AbOPK, i.e., produces keys and signatures that are indistinguishable from standard SchnorrSign ones. This again crucially relies on (at least) one public key to remain secret.

Proof. We prove the theorem through a short sequence of games, where Game_0 denotes the game played against the original MuSig. In the final game, the challenge public key and challenge oracle are entirely independent of the adversary's output S^* as well as of the challenge bit b, and thus the adversary can win the FullPriv-AbOPK game only with probability $1/2$. Our main strategy is that setting apk^* to a freshly sampled public key and answering $\mathcal{O}^{\mathtt{Chl}}$ queries related to apk^* using the corresponding secret key and SchnorrSign algorithm. By doing that, we will end up in a game where regardless of what the challenge bit b is, all queries are answered using a freshly sampled key pair and SchnorrSign algorithm.

Game_1: Figure 30 shows the changes between Game_0 and Game_1. To achieve the main goal we explained above, we need several steps. The first step is simulating FullPriv-AbOPK game without using key aggregation and multi-signature algorithms. In Game_1, we set an *aggregated secret key* ask^* and apk^* is just

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{MuSig},\mathcal{A}}(\lambda)$ Game_1

$b \leftarrow \{0,1\}, pp \leftarrow \mathsf{Pg}(1^\lambda), Q \leftarrow \emptyset, n \leftarrow \mathcal{A}(pp)$, **abort if** $n \not> 0$

$(SK, PK) := (\{sk_i\}_{i\in[n]}, \{pk_i\}_{i\in[n]}) \leftarrow (\mathsf{Kg}(pp))_{i\in[n]}$

$S^* \leftarrow \mathcal{A}(SK, PK)$, **abort if** $1 \notin S_j$ **for** $j \in \{0,1\}$

if $b = 0$ **then**

$r \leftarrow \mathsf{H}_1(pk_1, PK_{S^*})$

$ask^* \leftarrow sk_1 \cdot r + \sum_{j\in S^*\setminus\{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$apk^* \leftarrow \hat{g}^{ask^*}, pk^* \leftarrow apk^*$

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 $b = 1$ **then**

  return $\sigma \leftarrow \mathsf{BLSSign}(sk, m)$

$\sigma \leftarrow \mathsf{SchnorrSign}(ask^*, m)$

return σ

Fig. 30: Changes between Game_0 and Game_1 of FullPriv-AbOPK proof of MuSig. Gray highlight is used to indicate the changed code between games.

computed as a function of ask^*, exponentiation. Using ask^*, $\mathcal{O}^{\mathtt{Chl}}$ queries can be answered using SchnorrSign algorithm instead of MulSign() and Combine() algorithms even in the case that $b = 0$. The change in the original experiment is invisible to the adversary as we just perform an additional computation that does not impact the output. The change in the original experiment is invisible to the adversary as we just perform an additional computation that does not impact the output. Furthermore, we can observe that the change in $\mathcal{O}^{\mathtt{Chl}}$ is also perfectly indistinguishable from Game_1 as follows. A Schnorr signature σ can be parsed as (s, R). An s value which is a part of a multi-signature is,

$$\begin{aligned} s &= \sum_{pk_i\in PK} s_i \ \mathbf{mod}\ p \\ &= \sum_{pk_i\in PK} (r_i + sk_i \cdot c \cdot \mathsf{H}_1(pk_i, PK)) \ \mathbf{mod}\ p \\ &= (\sum_{i=1}^{n} r_i \ \mathbf{mod}\ p + c \cdot \sum_{pk_i\in PK} sk_i \cdot \mathsf{H}_1(pk_i, PK)) \ \mathbf{mod}\ p \\ &= (r + c \cdot ask) \ \mathbf{mod}\ p \end{aligned}$$

We also know that $r \leftarrow \sum_{i=1}^{n} r_i$ is uniformly random in $\mathbb{Z}_p$ just as in SchnorrSign. Thus, we conclude that $\Pr[W_1] = \Pr[W_0]$.

In Game_1, we have a game in which apk^* is computed from a secret key ask^* and $\mathcal{O}^{\mathtt{Chl}}$ queries are answered using the same secret key and SchnorrSign. In the following steps, we aim to show that the secret key ask^* is indistinguishable from a freshly sampled secret key.

Game_2: To show that ask^* is indistinguishable from a freshly sampled secret key, we need to show that we do not reveal information about pk_1 to the

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{MuSig},\mathcal{A}}(\lambda)\mathsf{Game}_1, \mathsf{Game}_2, \boxed{\mathsf{Game}_3}$

// No change in the initialization...

$S^* \leftarrow \mathcal{A}(SK \setminus \{sk_1\}, PK \setminus \{pk_1\}), \textbf{abort if } 1 \notin S_j \textbf{ for } j \in \{0,1\}$

if $b = 0$ **then**

if $\exists (PK')(\mathsf{HT}_1(pk_1, PK') \neq \perp)$: **abort**

$r \leftarrow \mathsf{H}_1(pk_1, PK_{S^*})$

$ask^* \leftarrow sk_1 \cdot r + \sum_{j \in S^* \setminus \{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$apk^* \leftarrow g^{ask^*}, pk^* \leftarrow apk^* \boxed{, \mathsf{Chlinit} \leftarrow 1}$

if $b = 1 : (sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}(\cdot)}(pk^*), \textbf{return } 1 \textbf{ if } b = b^*$

$\mathcal{O}^{\mathsf{H}_1}(pk', PK')$

$\boxed{\textbf{if } \mathsf{Chlinit} = 1 : \textbf{if } pk' = pk_1 : \textbf{abort}}$

if $\mathsf{HT}_1(pk', PK') = \perp$:

$\mathsf{HT}_1(pk', PK') \leftarrow \mathbb{Z}_p$

return $\mathsf{HT}_1(pk', PK')$

Fig. 31: Games Game_1, Game_2, and Game_3 of FullPriv-AbOPK proof of MuSig.

$\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{MuSig},\mathcal{A}}(\lambda)\mathsf{Game}_4, \boxed{\mathsf{Game}_5}, \boxed{\mathsf{Game}_6}$

// No change in the initialization...

$S^* \leftarrow \mathcal{A}(SK \setminus \{sk_1\}, PK \setminus \{pk_1\}), \textbf{abort if } 1 \notin S_j \textbf{ for } j \in \{0,1\}$

if $b = 0$ **then**

if $\exists (PK')(\mathsf{HT}_1(pk_1, PK') \neq \perp)$: **abort**

$r^* \leftarrow \mathbb{Z}_p, ask^* \leftarrow r^* + \sum_{j \in S^* \setminus \{1\}} sk_j \cdot \mathsf{H}_1(pk_j, PK_{S^*})$

$\boxed{ask^* \leftarrow \mathbb{Z}_p,} apk^* \leftarrow \hat{g}^{ask^*} \boxed{, (ask^*, apk^*) \leftarrow \mathsf{Kg}(pp)}, pk^* \leftarrow apk^*, \mathsf{Chlinit} \leftarrow 1$

if $b = 1 : (sk, pk) \leftarrow \mathsf{Kg}(pp), pk^* \leftarrow pk$

$b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Chl}}(\cdot)}(pk^*), \textbf{return } 1 \textbf{ if } b = b^*$

Fig. 32: Games Game_4, Game_5, and Game_6 of FullPriv-KPK proof of MuSig.

adversary and it will take two steps. We start by adding an abort condition in this game. Namely, if there was a H_1 query before setting the challenge set S^* which contains pk_1, then Game_2 aborts. Otherwise, it plays Game_1 identically. We show that this change is indistinguishable to the adversary as it only occurs with a negligible probability. pk_1 is a freshly sampled random value, so the probability that there is a previous query for pk_1 is q/p. Thus, we conclude that $|\Pr[W_2] - \Pr[W_1]| \leq q/p$.

Game_3: In this game, we show that the adversary can learn the hash values related to pk_1 only with a negligible probability. In Game_2, we already achieve a game hop that the adversary did not learn any hash values related to pk_1 until the challenge phase. To show that such queries are unlikely to occur even after the challenge phase, we add another abort condition in $\mathcal{O}^{\mathsf{H}_1}$. According to this condition, we abort if there is any hash queries related to pk_1 after the challenge

phase. To check whether a query is before or after the challenge phase, we add the flag $\mathsf{Chlinit}$ which is set to 0 by default and it is set to 1 in the challenge phase. We note that the game itself makes hash queries related to pk_1 to compute the aggregated secret key ask^*. However, the abort condition is only valid after the flag $\mathsf{Chlinit}$ is set. Thus, these queries do not cause a trivial abort.

We argue that the abort condition that we add holds only with a negligible probability as follows. By Game_2, we already know that there is no such random oracle query until the challenge set has been set. After that, the only value that we use which is dependent on pk_1 is ask^*. As we know that the adversary did not call the random oracle for pk_1 before we set ask^*, $r = \mathsf{H}_1(pk_1, PK_{S^*})$ is uniformly random to the adversary. It follows that $sk_1 \cdot r$ is uniformly random and finally the ask^* value is uniformly random. Thus, ask^* does not reveal any information about π^*. Now, we know that the oracles of Game_3 except $\mathcal{O}^{\mathtt{H}_1}$ do not reveal any information about pk_1. Hence, the adversary can make an $\mathcal{O}^{\mathtt{H}_1}$ query only with the negligible probability, $|\Pr[W_3] - \Pr[W_2]| \leq q/p$.

Game_4: Finally, we can start changing the way we compute ask^* to show its indistinguishability from a freshly sampled key. We change how we compute ask^*. We change $sk_1 \cdot r$ to r^* for a uniformly random r^*. As r is uniformly random in $\mathbb{Z}_p$, $sk_1 \cdot r$ is perfectly indistinguishable from r^*, so $\Pr[W_4] = \Pr[W_3]$.

Game_5: In this game, we sample ask^* uniformly from $\mathbb{Z}_p$. As r^* is uniformly random in $\mathbb{Z}_p$, $r^* + \sum_{j \in S^* \setminus \{i\}} sk_j \cdot \mathsf{H}_1(pk_j, PK)$ is perfectly indistinguishable from $ask^* \leftarrow \mathbb{Z}_p$, so $\Pr[W_5] = \Pr[W_4]$.

Game_6: Finally, we initialize key pair (ask^*, apk^*) as fresh individual key pair. While Game_5 samples ask^* from $\mathbb{Z}_p$, $\mathsf{Kg}(pp)$ samples it from $\mathbb{Z}_p^*$, which is a negligible difference. In particular, the advantage of the adversary against Game_5 is $|\Pr[W_6] - \Pr[5]| \leq 1/p$.

In the final game Game_6, challenge public key pk^* and corresponding signatures are computed as individual Schnorr signatures independently of the challenge bit b and challenge set S^*. Thus, the adversary has no more advantage than a random guess, so $\Pr[W_6] = 1/2$. Hence, using the sequences of games, we conclude that

$$\left|\mathsf{Exp}^{\mathsf{FullPriv\text{-}AbOPK}}_{\mathsf{MuSig},\mathcal{A}}(\lambda) - 1/2\right| \leq (2 \cdot q + 1)/p$$

□