Universally Composable Interactive and Ordered Multi-Signatures

Carsten Baum[1], Bernardo David[2], Elena Pagnin[3], and Akira Takahashi[4]

[1] Technical University of Denmark, Copenhagen, Denmark, cabau@dtu.dk
[2] IT University of Copenhagen, Copenhagen, Denmark, bernardo@bmdavid.com
[3] Chalmers University of Technology & University of Gothenburg, Gothenburg, Sweden, elenap@chalmers.se
[4] J.P.Morgan AI Research & AlgoCRYPT CoE, New York, USA, takahashi.akira.58s@gmail.com

May 9, 2025

Abstract. Multi-signatures allow a given set of parties to cooperate in order to create a digital signature whose size is independent of the number of signers. At the same time, no other set of parties can create such a signature. While non-interactive multi-signatures are known (e.g. BLS from pairings), many popular multi-signature schemes such as MuSig2 (which are constructed from pairing-free discrete logarithm-style assumptions) require interaction. Such interactive multi-signatures have recently found practical applications e.g. in the cryptocurrency space.
Motivated by classical and emerging use cases of such interactive multi-signatures, we introduce the first systematic treatment of interactive multi-signatures in the universal composability (UC) framework. Along the way, we revisit existing game-based security notions and prove that constructions secure in the game-based setting can easily be made UC secure and vice versa.
In addition, we consider interactive multi-signatures where the signers must interact in a fixed pattern (so-called ordered multi-signatures). Here, we provide the first construction of ordered multi-signatures based on the one-more discrete logarithm assumption, whereas the only other previously known construction required pairings. Our scheme achieves a stronger notion of unforgeability, guaranteeing that the adversary cannot obtain a signature altering the relative order of honest signers. We also present the first formalization of ordered multi-signatures in the UC framework and again show that our stronger game-based definitions are equivalent to UC security.

1 Introduction

Digital signatures play a crucial role in safeguarding online communication, as they make the authenticity and integrity of messages publicly verifiable. This fundamental property has widespread practical applications in e.g. digital certificates, electronic payment systems, and identification schemes. Although the primary objective of digital signatures was initially to enable public verification of message origin and integrity, a large body of work has explored additional guarantees and capabilities which signatures can provide.

One important feature for more complex signature schemes is the distribution of the signing process among a set of multiple parties. This allows to secret-share the signing key, which means that there is no single point where it resides and can be stolen from. There are multiple approaches exploring this concept, such as threshold signatures or multi signatures (MS) [IN83]. In addition, an MS can also guarantee that the message has been signed "in a specific order" by the signers, which makes it an ordered multi-signature (OMS) scheme [BGOY07]. Most importantly, in all of these cases, is that the size of the signature itself should be independent of the number of signers.

Multi-Signatures can be constructed from e.g. BLS [BLS04] in a non-interactive fashion where each signer just sends one message. But other realizations of MS from assumptions such as versions of the Discrete Logarithm (DL) problem or Short Integer Solution (SIS) problem require interaction. Those protocols are usually called *interactive* multi-signatures.

Interactive Multi-Signatures (IMS) have been deployed in the cryptocurrency space to realize a number of wallets. Ordered Multi-Signatures have been proposed to enhance the security of internet packet routing [BGOY07], Industrial Internet of Things mesh networks [AFY23], as well as to construct a verifiable delay function where delays are not dictated by the computational running time of devices [BDPT24].

Given the current and potential future deployment of IMS and OMS, it is surprising that the question of composition has so far not been addressed in a sufficient manner. In particular, it should be asked if IMS and OMS constructions can be proven secure in the universal composability (UC) framework, given their logical use together with other cryptographic primitives in e.g. blockchains.

1.1 Our Contributions

In this work, we investigate IMS and OMS security from the angle of composability. We give thorough UC security definitions and show how game-based constructions can directly imply UC security. Moreover, we also give the first construction of an OMS from a Discrete Logarithm-type assumption.

Revisiting Game-based Security of MS, IMS and OMS: We revisit the game-based definitions of MS, IMS and OMS and put them on equal footing in terms of notation and security games. For this, we revisit and strengthen the existing notions. For example, we put forth a general framework that handles complex corruption and forgery patterns for OMS.

Systematic UC Treatment of IMS and OMS: We present the first formalization of IMS and OMS in the UC model. These differ quite strongly from existing models of UC-secure signatures [Can04a, BMT18, CDG$^+$18] or threshold signatures [CGG$^+$20], so we devise a new framework to model dynamic key registration, preprocessing as well as order during signing accurately. Furthermore, we show that game-based security notions for MS and OMS imply our UC notion for these primitives, which implies strong composable security guarantees for existing construction with game-based security analyses.

The first OMS based on (A)OMDL: We introduce the first OMS scheme based on the hardness of the (Algebraic) One-More Discrete Logarithm assumption and the random oracle model, and show that it is secure with respect to our game-based definitions. Assuming an interactive preprocessing phase which can be carried out independently of the message to be signed, our OMS achieves an attractive set of features including: constant-size signatures (both as final proof of order signing and during partial signature aggregation) and computation and communication complexity (between every pair of consecutive signers) that is independent during the online signing phase of the total number of participants, and signers'-order unforgeability even in the presence of multiple dishonest signers in the set of co-signers.

Our OMS construction can be seen as a round-efficient interactive Schnorr-based multisignature where order is enforced by having each signer check the so-far aggregated signature (created by the preceding co-signers). It is obtained by carefully adjusting MuSig2 [NRS21] to comply with the ordered-signing setting. This yields compact signatures while keeping the computational complexity of each signer in the online phase (i.e. once they receive a message to be signed) independent of the total number of parties. We inherit the offline-online setup of MuSig2, and signers can preprocess the first round of communication before receiving the message. This motivates us to revisit existing (game-based) security models for OMS and introduce a generalized syntax that captures interactive OMS and explicitly supports this setting.

1.2 Related Work

There is a large body of work on interactive and non-interactive multi signatures. For the focus of this paper, the most relevant results are the ones connected to group-based constructions such as the BLS MS scheme [Bol03, RY07, BDN18], MuSig2 [NRS21], and works inspired by these, including lattice-based MS [BTT22]. A similar concept of "stake-based multisignatures" with extra properties has been defined and formalized in the UC framework in [CK21], but in this work we focus on standard MS and OMS.

Regarding ordered multi-signatures, the initial works considered a non-interactive setting often referred to as Sequential Aggregate Signatures (SAS). SAS constructions achieving compact signatures rely either on bilinear pairings [LMRS04, LOS$^+$06, BGOY07, BNN07, FLS12] or trapdoor permutations [LMRS04, Nev08, BGR12, GOR18]. The existing Schnorr-based SAS [CZ22] has a linear growth of the signature size. We take a different approach by lifting the bandwidth-efficient Schnorr-based interactive multi-signature of [NRS21] to OMS. Moreover, we work in the interactive OMS setting, a notion introduced by Boldyreva et al. [BGOY07] with the aim of enhancing network reliability and security by authenticity of data packets traveling through specific routes in networks. Sequent work removed the need for random oracles from pairing-based constructions [Hou10, YMO15], but no efficient pairing-free construction is known to date.

Unlike MS, the UC security of threshold signatures (TS) has been extensively studied in the literature in different forms [AF04, ADN06, CGG$^+$20, Lin17, DKLs18, Lin24, KOR23]. Although some MS and TS *constructions* happen to have similar structures, such as FROST [KG20] and MuSig2 [NRS21], the functional goals of MS and TS are substantially different. The goal of TS is to produce a signature that looks like a non-threshold signature generated under a single signing key. As such, TS fixes a single public key and a set of co-signers once and for all after a distributed key generation phase. To capture these, the majority of recent threshold Schnorr [Lin24, KOR23] and ECDSA [Lin17, DKLs18] protocols are UC secure in the sense that they realize scheme-specific functionalities, such as $\mathcal{F}_{\mathsf{Schnorr}}$ and $\mathcal{F}_{\mathsf{ECDSA}}$.

In contrast, as the purpose of MS is to combine individually generated signatures under different keys, the verification of MS is not interchangeable with that of an ordinary signature scheme. Importantly, the key generation phase of MS requires no coordination between signers as each generates their own key pair individually. Unlike TS, MS also allows for dynamically changing sets of potential co-signers associated with different public keys. As a result, our functionalities are incomparable to existing functionalities for abstract TS proposed in [AF04, ADN06, CGG$^+$20].

Concurrent work The concurrent work by Cohen et al. [CDL$^+$24] pointed out subtle issues of Canetti's original signature functionality $\mathcal{F}_{\mathsf{Sig}}$ when used as a subroutine of the Dolev-Strong broadcast protocol, and proposed an improved variant of $\mathcal{F}_{\mathsf{Sig}}$. While our functionalities can be used to construct protocols such as the sequential communication delay [BDPT24], similar to Canetti's $\mathcal{F}_{\mathsf{Sig}}$, they may not be used by protocols that suffer from the adversary's ability to take control back amid signature generation. We leave the task of upgrading our IMS and OMS functionalities to support broader applications to future work. As an additional contribution, the authors of [CDL$^+$24] patched Canetti's equivalence result for ordinary signature functionalities by carefully redesigning the correctness game. Although our work is concurrent and independent—originating from an earlier ePrint version of [BDPT24]—an earlier version of this manuscript contained an issue in the correctness definitions of IMS and OMS, which prevented us from proving the equivalence results. To address this, we took inspiration from [CDL$^+$24] and proposed new correctness notions that allow an adversary to adaptively query the oracles with chosen inputs (see Definition 5 and 10). This may be viewed as a generalization of the approach taken by [CDL$^+$24] to multi-signatures.

1.3 Paper Organization:

In Section 2, we recall UC basics, standard ideal functionalities, forking lemma, and the one-more discrete logarithm assumption. In Section 3, we formalize game-based security notions, ideal functionality for IMS, and UC-security of IMS protocols. Section 4 presents analogous results on OMS. Finally, Section 5 presents our OMS construction that is provably secure under the OMDL assumption in the random oracle model.

2 Preliminaries

Notation. We denote security parameter by λ, the concatenation of two strings a and b by $a|b$, and compact multiple concatenations by $(a_i)_{i=1}^n = a_1|a_2|\dots|a_n$.

2.1 Universal Composability and Ideal Functionalities

UC Primer. We use the Universal Composability framework [Can01] for analyzing security and refer interested readers to the original works for more details. In UC, protocols are run by interactive Turing Machines (iTMs) called *parties*. A protocol π will have a set of parties which we denote as P. The *adversary* $\mathcal{A}$, which is also an iTM, can corrupt a subset $C \subset \mathcal{P}$ as defined by the security model and gains control over these parties. In this work, we focus on static corruption, i.e., the adversary commits to a corruption set C at the beginning of a protocol execution. The parties can exchange messages via resources, called *ideal functionalities* (which themselves are iTMs) and which are denoted by $\mathcal{F}$.

Security is defined with respect to an iTM $\mathcal{Z}$ called *environment*. The environment provides inputs to and receives outputs from the parties $\mathcal{P}$. To define security, let $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance π using resources $\mathcal{F}_1, \ldots$. Furthermore, let $\mathcal{S}$ denote an *ideal world adversary* and $\mathcal{F} \circ \mathcal{S}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}$ instead of π and where $\mathcal{S}$ takes care of adversarial behavior.

Definition 1. *We say that π UC-realizes $\mathcal{F}$ in the $(\mathcal{F}_1, \ldots)$-hybrid model if for every (efficient) iTM $\mathcal{A}$ there exists an (efficient) iTM $\mathcal{S}$ such that no (efficient and balanced) environment $\mathcal{Z}$ can distinguish $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ from $\mathcal{F} \circ \mathcal{S}$ with non-negligible probability.*

In the security experiment $\mathcal{Z}$ may arbitrarily activate parties or $\mathcal{A}$, though *only one iTM (including $\mathcal{Z}$) is active at each point of time.*

In the plain UC framework, an ideal functionality $\mathcal{F}$ is assumed to be local to a single protocol session. To model a more realistic situation in which multiple protocols share the same "setup" resources, such as standardized hash functions (modeled as random oracles), public key infrastructure, or public ledgers, it is often useful to treat some functionalities as a *global subroutine* $\mathcal{G}$ in such a way that the same instance of $\mathcal{G}$ is globally available across many protocol sessions. Although the treatment of global subroutines previously required a different compositional framework [CR03, CDPW07], Badertscher et al. [BCH$^+$20] simplified the situation by providing the UC theorem in the presence of global subroutines in the (2020 version of) plain UC framework of Canetti [Can00]. Thus, the definition of UC-realization above can be naturally extended by considering both $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ and $\mathcal{F} \circ \mathcal{S}$ with access to a global subroutine $\mathcal{G}$ (using the so-called "management protocol" to formally treat the combination with $\mathcal{G}$ as a single protocol [BCH$^+$20, 3.1]). If for any $\mathcal{A}$, there exists $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish these two

Functionality 1: $\mathcal{G}_{\mathsf{RO}}$ for the Global Random Oracle

$\mathcal{G}_{\mathsf{RO}}$ is parameterized by an output size function ℓ and a security parameter τ, and keeps initially empty lists $\mathsf{List}_{\mathcal{H}}$.

Query: On input $(\textsc{Hash-Query}, m)$ from party $(\mathcal{P}, \mathsf{sid})$, proceed as follows:

1. Look up h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, sample $h \xleftarrow{\$} \{0,1\}^{\ell(\tau)}$ and set $\mathsf{List}_{\mathcal{H}} = \mathsf{List}_{\mathcal{H}} \cup \{(m, h)\}$.
2. Send $(\textsc{Hash-Confirm}, h)$ to the caller.

protocols, we say that π UC-realizes $\mathcal{F}$ in the $(\mathcal{F}_1, \ldots)$-hybrid model *in the presence of* $\mathcal{G}$.

The Global Random Oracle $\mathcal{G}_{\mathsf{RO}}$*.* In Functionality 1 we present the "strict" version of global random oracle ideal functionality from [CDG+18] without programmability and query observability. It follows the standard notion of a random oracle, when defined in the UC framework.

Digital Signatures Ideal Functionality $\mathcal{F}_{\mathsf{Sig}}$*.* The standard digital signature functionality $\mathcal{F}_{\mathsf{Sig}}$ from [Can04b] captures a randomized signature scheme where the signer may influence the generation of a signature by choosing the randomness used by the signing algorithm. This particularity is captured by allowing the ideal adversary $\mathcal{S}$ to choose a new string σ to represent a signature on a message m every time the signer $\mathcal{P}_s$ (a special party who has the right to generate signatures, *i.e.*, who holds the signature key) makes a new request for a signature on m. This process allows for multiple valid signatures to be produced for the same message. In the UC formalization of signature schemes, an instance of the functionality $\mathcal{F}_{\mathsf{Sig}}$ itself represents each different signing key by allowing only a special party $\mathcal{P}_s$ (*i.e.* the holder of a signing key) to produce signatures.

It is shown in [Can04b] that any EUF-CMA signature scheme UC realizes the standard signature functionality where multiple valid signatures may be produced for the same message under the same signing key (*i.e.* the same instance of $\mathcal{F}_{\mathsf{Sig}}$ may generate multiple signatures for the same message, as long as they have not been flagged as invalid signatures by a previous unsuccessful verification procedure). In [CDG+18], the authors extend this result to signature schemes that are EUF-CMA secure in the random oracle model. Importantly, even if the proof of EUF-CMA security requires observability and programmability of the RO, one can still guarantee that the corresponding signature protocol UC-realizes $\mathcal{F}_{\mathsf{Sig}}$

Functionality 2: $\mathcal{F}_{\mathsf{Sig}}$ for Digital Signatures [Can04b]

Given an ideal adversary $\mathcal{S}$, a signer $\mathcal{P}_s$, and an arbitrary verifier $\mathcal{V}$, $\mathcal{F}_{\mathsf{Sig}}$ performs:

Key Generation: Upon receiving $(\textsc{KeyGen}, \mathsf{sid})$ from $\mathcal{P}_s$, verify that $\mathsf{sid} = (\mathcal{P}_s, \mathsf{sid}')$ for some sid'. If not, ignore the request. Else, hand $(\textsc{KeyGen}, \mathsf{sid})$ to the adversary $\mathcal{S}$. Upon receiving $(\textsc{Verification Key}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{S}$, output $(\textsc{Verification Key}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{P}_s$, and record the pair $(\mathcal{P}_s, \mathsf{pk})$.

Signature Generation: Upon receiving a message $(\textsc{Sign}, \mathsf{sid}, m)$ from $\mathcal{P}_s$, verify that $\mathsf{sid} = (\mathcal{P}_s, \mathsf{sid}')$ for some sid' . If not, then ignore the request. Else, send $(\textsc{Sign}, \mathsf{sid}, m)$ to $\mathcal{S}$. Upon receiving $(\textsc{Signature}, \mathsf{sid}, m, \sigma)$ from $\mathcal{S}$, verify that no entry $(m, \sigma, \mathsf{pk}, 0)$ is recorded. If it is, then output an error message to $\mathcal{P}_s$ and halt. Else, output $(\textsc{Signature}, \mathsf{sid}, m, \sigma)$ to $\mathcal{P}_s$, and record the entry $(m, \sigma, \mathsf{pk}, 1)$.

Signature Verification: Upon receiving a message $(\textsc{Verify}, \mathsf{sid}, m, \sigma, \mathsf{pk}')$ from some party $\mathcal{V}$, hand $(\textsc{Verify}, \mathsf{sid}, m, \sigma, \mathsf{pk}')$ to $\mathcal{S}$. Upon receiving $(\textsc{Verified}, \mathsf{sid}, m, \phi)$ from $\mathcal{S}$ do:

1. If $\mathsf{pk}' = \mathsf{pk}$ and the entry $(m, \sigma, \mathsf{pk}, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key pk' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
2. Else, if $\mathsf{pk}' = \mathsf{pk}$, the signer $\mathcal{P}_s$ is not corrupted, and no entry $(m, \sigma', \mathsf{pk}, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, \mathsf{pk}, 0)$. (This condition guarantees unforgeability: If pk' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry $(m, \sigma, \mathsf{pk}', f')$ recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $f = \phi$ and record the entry $(m, \sigma, \mathsf{pk}', \phi)$.

Output $(\textsc{Verified}, \mathsf{sid}, m, f)$ to $\mathcal{V}$.

in the presence of *strict* global random oracle $\mathcal{G}_{\mathsf{RO}}$ without observability and programmability.

2.2 Forking Lemma

We restate a variant of the general forking lemma from [NRS21]. It differs from the widely used version of [BN06] in that an algorithm rewound by the forking algorithm Fork takes additional side inputs v_j's.

Lemma 2 (General Forking Lemma with side inputs [NRS21]). *Fix integers Q and m and sets C and V of size greater than 2. Let* IGen *be a randomized algorithm that we call the input generator. Let $\mathcal{B}$ be a randomized algorithm that on input* inp*, $c_1, \ldots, c_Q \in C$ and $v_1, \ldots, v_m \in V$ returns indices $i \in [0, Q]$, $j \in [0, m]$ and a side output* out*. Let* Fork *be a forking algorithm that works as in Fig. 3 given* inp *as input and given*

```
Fork(inp, v_1, \hat{v}_1, ..., v_m, \hat{v}_m)
1: $\rho \leftarrow \{0,1\}^*$
2: $c_1, \ldots, c_Q \leftarrow C$
3: $(i, j, \mathsf{out}) \leftarrow \mathcal{B}(\mathsf{inp}, (c_1, \ldots, c_Q), (v_1, \ldots, v_m); \rho)$
4: if $i = 0$ then return $(0, \perp, \perp)$
5: $\hat{c}_i, \ldots, \hat{c}_Q \leftarrow C$
6: $(\hat{i}, \hat{j}, \hat{\mathsf{out}}) \leftarrow \mathcal{B}(\mathsf{inp}, (c_1, \ldots, c_{i-1}, \hat{c}_i, \ldots, \hat{c}_Q), (v_1, \ldots, v_j, \hat{v}_{j+1}, \ldots, \hat{v}_m); \rho)$
7: if $i = \hat{i} \wedge c_i \neq \hat{c}_i$ then return $(1, \mathsf{out}, \hat{\mathsf{out}})$
8: else return $(0, \perp, \perp)$
```

black-box access to $\mathcal{B}$. Suppose the following probabilities.

$$\mathsf{acc} := \Pr\left[i \geq 1 \ : \ \begin{array}{c} \mathsf{inp} \leftarrow \mathsf{IGen}(1^\lambda); c_1, \ldots, c_Q \xleftarrow{\$} C; v_1, \ldots, v_m \xleftarrow{\$} V; \\ (i, j, \mathsf{out}) \leftarrow \mathcal{B}(\mathsf{inp}, (c_1, \ldots, c_Q), (v_1, \ldots, v_m)) \end{array} \right]$$

$$\mathsf{frk} := \Pr\left[b = 1 \ : \ \begin{array}{c} \mathsf{inp} \leftarrow \mathsf{IGen}(1^\lambda); v_1, \hat{v}_1 \ldots, v_m, \hat{v}_m \xleftarrow{\$} V; \\ (b, \mathsf{out}, \hat{\mathsf{out}}) \leftarrow \mathsf{Fork}(\mathsf{inp}, v_1, \hat{v}_1, \ldots, v_m, \hat{v}_m) \end{array} \right]$$

Then

$$\mathsf{frk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}}{Q} - \frac{1}{|C|} \right).$$

Alternatively,

$$\mathsf{acc} \leq \frac{Q}{|C|} + \sqrt{Q \cdot \mathsf{frk}}.$$

2.3 One-More Discrete Log Assumption

We state the one-more discrete log assumption which our OMS construction inherits from MuSig2 Schnorr-based (unordered) multi-signature [NRS21].

Definition 3 (OMDL Assumption). *Let $(\mathbb{G}, p, g)$ be group parameters generated by a group generator algorithm $\mathsf{GGen}(1^\lambda)$. Let $\mathcal{O}_{ch}$ be an oracle that returns a uniformly random element of $\mathbb{G}$, and $\mathcal{O}_{dl}$ be an oracle that on input $X \in \mathbb{G}$ returns its discrete logarithm $x \in \mathbb{Z}_p$ in base g, i.e., $g^x = X$. The one more discrete log assumption (OMDL) is true if for any PPT adversary $\mathcal{A}$, the probability $\mathbf{Adv}^{\mathsf{OMDL}}_{\mathsf{GGen}}(\mathcal{A})$ that, $\mathcal{A}$ outputs solutions to $q+1$ DLog instances produced by $\mathcal{O}_{ch}$ while having made at most q queries to $\mathcal{O}_{dl}$, is negligible in λ.*

3 Interactive Multi-Signatures

In this section, we provide the syntax (Section 3.1) for two-round interactive multi-signatures (IMS), a game-based security notion for IMS (Section 3.2), and an ideal functionality $\mathcal{F}_{\mathsf{IMS}}$ for IMS (Section 3.3). We then prove that any scheme secure according to the game-based IMS security notion also UC-realizes $\mathcal{F}_{\mathsf{IMS}}$ (Section 3.4).

3.1 Syntax

We define the syntax for IMS below. The syntax supports *two-round signing protocols with a preprocessing phase*, to model recent Fiat-Shamir-based schemes[5] such as MuSig2 [NRS21], DWMS [AB21], MuSig-L [BTT22], etc. Note that the following syntax can also model non-interactive multi-signatures as a special case by assuming $\mathsf{SignOff}$ outputs empty strings.

Definition 4 (Interactive Multi-Signature Scheme (IMS)). *We define interactive multi-signature* IMS *as a tuple of algorithms*[6]

$$\mathsf{IMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SignOff}, \mathsf{SignOn}, \mathsf{Agg}, \mathsf{Vrfy})$$

with the following input-output behavior. To formally handle two-stage interactive signing, $\mathsf{SignOff}$ *and* SignOn *share an internal state* $\mathtt{st}$.

$\mathsf{Setup}(1^\lambda)$**:** *on input the security parameter λ, this algorithm outputs a handle of public parameters* pp. *Throughout, we assume that* pp *is given as implicit input to all other algorithms.*

$\mathsf{KeyGen}()$**:** *on input the public parameters, the key generation algorithm outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{SignOff}(\mathsf{sk})$**:** *on input a secret key* sk, *the offline signing algorithm outputs an offline token* off *and an internal state* $\mathtt{st}$. *We note that this algorithm is agnostic of the message m to be signed, and runs independently of m.*

[5] To turn Fiat-Shamir-type signature schemes into compact multi-signatures, one must introduce at least two rounds of interaction in the signing phase because signers first must agree on joint Σ-protocol challenge derived from a common hash input.

[6] We remark that IMSs as defined in [NRS21] have an additional signing function that post-processes the aggregated online tokens to generate a signature. We assume this function to be trivial, meaning that it merely outputs the input aggregated online tokens as a signature (as MuSig2 does) and therefore can be omitted. Another minor modification from [NRS21] is that Vrfy explicitly takes a public key list L instead of a single aggregated key. This does not impact the security claim: since in the EUF-CMA game of [NRS21] the verifier is guaranteed to receive an output of key aggregation anyway, we can assume this operation happens inside Vrfy.

$\mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs})$**:** *on input an internal state* $\mathtt{st}$ *output by* $\mathsf{SignOff}$*, a secret key* sk*, a set of public keys* $L = \{\mathsf{pk}_j\}_{j\in[n]}$*, message* m*, offline tokens* $\mathsf{offs} = \{\mathsf{off}_j\}_{j\in[n]}$*, the online signing algorithm outputs an online signing token* σ*.*

$\mathsf{Agg}(L, m, \mathsf{offs}, \mathsf{ons})$**:** *on input a set of public keys* $L = \{\mathsf{pk}_j\}_{j\in[n]}$*, a message* m*, offline tokens* $\mathsf{offs} = \{\mathsf{off}_j\}_{j\in[n]}$*, and online tokens* $\mathsf{ons} = \{\sigma_j\}_{j\in[n]}$*, the aggregation algorithm outputs a signature* σ*. This algorithm is deterministic.*

$\mathsf{Vrfy}(L, m, \sigma)$**:** *on input a set of public keys* L*, a message* m*, and a signature* σ*, the verification algorithm outputs* 1 *(accept) or* 0 *(reject).*

3.2 Game-based Security

We recall the game-based correctness and unforgeability notions tailored to two-round offline-online IMS. The unforgeability game mostly follows [NRS21] except that we assume that aggregation of offline tokens is locally performed by each party (which is in fact the setting captured by the more widely used security notion of [BN06]). Inspired by [CDL$^+$24], we carefully define correctness to enable equivalence with our UC functionality presented in Section 3.3. Instead of fixing the message and key pairs in advance, we allow an adversary to adaptively determine the message to be signed after observing honest public keys. Unlike [CDL$^+$24], however, our definition is more involved to handle multiple honest signers and the adversary's ability to concurrently query interactive signing oracles.

Definition 5 (IMS Correctness). *An interactive multi signature* IMS *is said to be correct if for any probabilistic polynomial time adversary* $\mathcal{A}$*, the probability* $\Pr[\mathsf{IMS\text{-}COR}(\mathcal{A}, \lambda) = 1]$ *is negligible in* λ*, where the* IMS-COR *experiment is defined in Figure 4.*

Definition 6 (IMS Unforgeability). *An interactive multi signature* IMS *is said to be secure if for any probabilistic polynomial time adversary* $\mathcal{A}$*, the following probability is negligible in* λ*:*

$$\mathbf{Adv}^{\mathsf{IMS\text{-}UF\text{-}CMA}}_{\mathsf{IMS}}(\mathcal{A}, \lambda) := \Pr[\mathsf{IMS\text{-}UF\text{-}CMA}(\mathcal{A}, \lambda) = 1]$$

where IMS-UF-CMA *is the security-game for unforgeability under chosen-message attack of interactive multi-signatures defined in Figure 5.*

Figure 4: Game-based correctness for IMS

```
GAME IMS-COR($\lambda$)
1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2: $\mathcal{K} := \emptyset$; $\mathcal{T} := \emptyset$; $\mathcal{Q} := \emptyset$
3: $\mathcal{O} := \{\mathsf{OKeyReg}, \mathsf{OSignOff}, \mathsf{OSignOn}\}$
   ▷ RO in $\mathcal{O}$ if needed
4: $(L^*, m^*, \mathsf{offs}^*, \mathsf{ons}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
5: $\sigma^* \leftarrow \mathsf{Agg}(L^*, m^*, \mathsf{offs}^*, \mathsf{ons}^*)$
6: $\{\mathsf{pk}_1^*, \dots, \mathsf{pk}_n^*\} := L^*$
7: $\{\mathsf{off}_1^*, \dots, \mathsf{off}_n^*\} := \mathsf{offs}^*$
8: $\{\sigma_1^*, \dots, \sigma_n^*\} := \mathsf{ons}^*$
9: If $\forall\, i \in [n]$ :
       $\mathcal{K}[\mathsf{pk}_i^*] \neq \bot$
     $\wedge \exists \mathsf{ssid}\ (\mathcal{T}[\mathsf{ssid}, \mathsf{pk}_i^*] = (\cdot, \mathsf{off}_i^*)$
     $\wedge \mathcal{Q}[\mathsf{ssid}, \mathsf{pk}_i^*] = (L^*, m^*, \mathsf{offs}^*, \sigma_i^*))$:
10:   return $(\mathsf{Vrfy}(L^*, m^*, \sigma^*) = 0)$
11: return 0
```

```
ORACLE OKeyReg()
1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
2: $\mathcal{K}[\mathsf{pk}] := \mathsf{sk}$
3: return $\mathsf{pk}$
```

```
ORACLE OSignOff(ssid, pk)
1: if $\mathcal{T}[\mathsf{ssid}, \mathsf{pk}] \neq \bot$ then
2:   return $\bot$            ▷ Signing session exists
3: $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$
4: $\mathcal{T}[\mathsf{ssid}, \mathsf{pk}] := (\mathtt{st}, \mathsf{off})$
5: return $\mathsf{off}$
```

```
ORACLE OSignOn(ssid, pk, $L$, $m$, offs)
1: if $\mathsf{pk} \notin L \vee \mathcal{K}[\mathsf{pk}] = \bot$ then return $\bot$
2: $\mathsf{sk} := \mathcal{K}[\mathsf{pk}]$
3: $\{\mathsf{pk}_1, \dots, \mathsf{pk}_n\} := L$; let $i$ such that $\mathsf{pk}_i = \mathsf{pk}$.
4: $\{\mathsf{off}_1, \dots, \mathsf{off}_n\} := \mathsf{offs}$
5: if $(\mathcal{Q}[\mathsf{ssid}, \mathsf{pk}] \neq \bot) \vee (\mathcal{T}[\mathsf{ssid}, \mathsf{pk}] \neq (\cdot, \mathsf{off}_i))$
   then
6:   return $\bot$            ▷ Invalid signing session
7: $(\mathtt{st}, \mathsf{off}_i) := \mathcal{T}[\mathsf{ssid}, \mathsf{pk}]$
8: $\sigma \leftarrow \mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs})$
9: $\mathcal{Q}[\mathsf{ssid}, \mathsf{pk}] := (L, m, \mathsf{offs}, \sigma)$
10: return $\sigma$
```

3.3 Ideal Functionality for Multi-Signatures

We define an ideal functionality for IMS $\mathcal{F}_{\mathsf{IMS}}$ (Functionality 6) to capture two-round interactive multi-signatures with preprocessing. An ideal functionality for stake-based threshold multi-signatures exists in the literature [CK21]. Our functionality is much simpler because it aims to capture existing schemes in the standard plain-public key model. It can be seen as a generalization of $\mathcal{F}_{\mathsf{Sig}}$ from [Can03] with following differences:

- $\mathcal{F}_{\mathsf{Sig}}$ is defined for a single designated signer $\mathcal{P}_s$ and only accepts a key generation query with sid encoding a signer identity, i.e., $\mathsf{sid} = (\mathcal{P}_s, \mathsf{sid}')$ for some sid', whereas $\mathcal{F}_{\mathsf{IMS}}$ is defined w.r.t. a set of signers P and records an individual key for every signer $\mathcal{P} \in P$.
- $\mathcal{F}_{\mathsf{IMS}}$ additionally has the SIGNOFF command, which allows any party $\mathcal{P} \in P$ to produce a preprocessed offline "presignature" off independently of the message to be signed. The functionality keeps track of presignatures for every honest party and makes sure to generate one presignature for each signing session. The functionality then explicitly outputs the preprocessing shares that each party generates, and they are chosen by the simulator (which is a standard approach in UC).
- $\mathcal{F}_{\mathsf{IMS}}$'s SIGNON command takes a key set L, a message m to be signed, and presignatures offs from the offline phase, as input. To capture se-

Figure 5: Game-based unforgeability for IMS

```
GAME IMS-UF-CMA($\mathcal{A}, \lambda$)
1: $\mathcal{T} := \emptyset$; $\mathcal{Q} := \emptyset$
2: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
4: $\mathcal{O} := \{\mathsf{OSignOff}, \mathsf{OSignOn}\}$    ▷ If pp contain hash functions, RO is in $\mathcal{O}$
5: $(L^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk})$
6: if $\mathsf{pk} \notin L^*$ then
7:   return 0
8: if $\exists \mathsf{ssid}^* : \mathcal{Q}[\mathsf{ssid}^*] = (L^*, m^*)$ then
9:   return 0
10: return $\mathsf{Vrfy}(L^*, m^*, \sigma^*)$
```

```
ORACLE OSignOff(ssid)
1: if $\mathcal{T}[\mathsf{ssid}] \neq \perp$ then
2:   return $\perp$    ▷ Signing session exists
3: $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$
4: $\mathcal{T}[\mathsf{ssid}] := (\mathtt{st}, \mathsf{off})$
5: return off
```

```
ORACLE OSignOn($\mathsf{ssid}, L, m, \mathsf{offs}$)
1: if $\mathsf{pk} \notin L$ then return $\perp$
2: $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\} := L$; let $i$ such that $\mathsf{pk}_i = \mathsf{pk}$.
3: $\{\mathsf{off}_1, \ldots, \mathsf{off}_n\} := \mathsf{offs}$
4: if $(\mathcal{Q}[\mathsf{ssid}] \neq \perp) \vee (\mathcal{T}[\mathsf{ssid}] \neq (\cdot, \mathsf{off}_i))$ then
5:   return $\perp$    ▷ Invalid signing session
6: $(\mathtt{st}, \mathsf{off}_i) := \mathcal{T}[\mathsf{ssid}]$
7: $\sigma \leftarrow \mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs})$
8: $\mathcal{Q}[\mathsf{ssid}] := (L, m)$
9: return $\sigma$
```

curity in the plain public key model, the functionality does not validate keys of co-signers. This allows the environment to insert maliciously created self-chosen keys into L. Moreover, to model concurrent signing sessions with different subsets of co-signers, $\mathcal{F}_{\mathsf{IMS}}$ keeps track of each signing attempt using ssid and retrieves the corresponding offline presignature associated with the same ssid. Then $\mathcal{F}_{\mathsf{IMS}}$ stores an online signature output for party $\mathcal{P}$ if offline and online phases are executed in right order with valid inputs. Note that the signing session ID ssid is local to each party $\mathcal{P}$, and that the functionality does not assume that parties in P agree on the same ssid to produce a valid signature. Looking ahead, this allows us to map an execution of the UC experiment to the unforgeability game (Fig. 5) where there is no assumption on communication channels such as broadcast.

– $\mathcal{F}_{\mathsf{IMS}}$ additionally has the AGGREGATE command, which allows any party to aggregate offline tokens offs and (online) partial signatures ons into a single combined signature σ. Note that this command does not perform a validity check of individual partial signatures. To guarantee correctness, AGGREGATE stores a finalized multi-signature σ as long as it has been produced from honestly generated offs and ons. The abort event is analogous to the one in $\mathcal{F}_{\mathsf{Sig}}$, making sure that previously recorded invalid signature (through verification) does not turn valid. Note that we do not guarantee correctness if the key set L involves an unregistered key or a key owned by a corrupt signer. This

is necessary for modeling typical IMSs which do not have robustness by default, including MuSig2.
- $\mathcal{F}_{\mathsf{IMS}}$ verifies a multi-signature analogously to $\mathcal{F}_{\mathsf{Sig}}$, but its procedures are more involved due to multiple signers. It guarantees completeness by checking that the AGGREGATE command was invoked on some ons which have been explicitly created via the SIGNON command on input L and m. To model unforgeability, if there exists some honest party $\mathcal{P}$ that has never agreed to sign m with L for any signing session ssid, then it rejects a signature. Otherwise, $\mathcal{F}_{\mathsf{IMS}}$ asks an ideal adversary to determine the result of verification.
- Similar to the game-based syntax for IMS, $\mathcal{F}_{\mathsf{IMS}}$ is general enough that it captures non-interactive MS, such as the BLS-based MS scheme of [BDN18]. In that case, offline presignatures are assumed to be empty strings.

3.4 UC-security of Interactive Multi-Signatures

In Fig. 7 we define π_{IMS} as a direct instantiation of IMS in the UC model. Note that, as observed in the analysis of standard signature schemes in the UC framework [CDG^{+}18], the proof indeed goes through in the presence of *strict* global random oracle $\mathcal{G}_{\mathsf{RO}}$, since the simulator merely relays random oracle queries made by an algorithm of IMS to the global functionality.

Theorem 7. *Let* IMS *be an interactive multi-signature scheme that is correct (Definition 5) and* $\mathsf{IMS\text{-}UF\text{-}CMA}$ *secure in the random oracle model (Definition 6). Let* $|P| \in \mathsf{poly}(\lambda)$*. Then the protocol* π_{IMS} *(Figure 7) UC-realizes* $\mathcal{F}_{\mathsf{IMS}}$ *in the presence of* $\mathcal{G}_{\mathsf{RO}}$*.*

Proof. In Figure 8 we describe a PPT simulator $\mathcal{S}_{\mathsf{IMS}}$ that runs a copy of $\mathcal{A}$, and simulates $\mathcal{F}_{\mathsf{IMS}}$ in a way that is indistinguishable to any efficient environment. The IMS scheme used in the proof follows the syntax given in Def. 4.

The proof is two-fold. First, we prove that $\mathcal{Z}$ never observes discrepancies between real and ideal executions by causing **Signature Verification** to output 0 in the real world for honestly generated signatures. This event needs to be ruled out because in the ideal execution $\mathcal{F}_{\mathsf{IMS}}$ always accepts the signature in that case. This follows from the interactive correctness definition. The second step closely follows that of [CDG^{+}18] for a single-user signature scheme. Essentially, assuming correctness of

Functionality 6: $\mathcal{F}_{\mathsf{IMS}}$ for interactive multi-signatures in the plain public-key model

The functionality interacts with a set of signers P, an arbitrary verifier $\mathcal{V}$, and a simulator $\mathcal{S}$. The adversary can corrupt a subset of signers, denoted by the set $C \subsetneq P$. We denote a set of honest parties by $H = P \setminus C$.
We assume the functionality only accepts a key set L without duplicates. We assume sid encodes identities of all possible signers P. We distinguish different offline phases by using unique ssids.

Key generation Upon receiving input $(\textsc{KeyGen}, \mathsf{sid})$ from $\mathcal{P} \in P$:

1. If there exists a record $(\textsc{keyrec}, \mathcal{P}, *)$, then output $(\textsc{fail}, \mathsf{sid})$. ▷ Prevent overwriting
2. Send $(\textsc{KeyGen}, \mathsf{sid}, \mathcal{P})$ to $\mathcal{S}$ and wait for $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{S}$.
3. If there exists a record $(\textsc{msig}, L, *, *, *)$ such that $\mathsf{pk} \in L$, then abort. ▷ Enforce distinct public keys
4. Create record $(\textsc{keyrec}, \mathcal{P}, \mathsf{pk})$ and output $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{P}$.

Signature Setup (Offline Phase) Upon receiving $(\textsc{SignOff}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P} \in P$:

1. If $\mathcal{P}$ previously called $\textsc{SignOff}$ with the same ssid, then output $(\textsc{fail}, \mathsf{sid})$.
2. Retrieve $(\textsc{keyrec}, \mathcal{P}, \mathsf{pk})$ or output $(\textsc{fail}, \mathsf{sid})$ if no such record exists.
3. Send $(\textsc{Presig}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}, \mathsf{pk})$ to $\mathcal{S}$, wait for response $(\textsc{Presig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off})$. Then create a record $(\textsc{presig}, \mathsf{ssid}, \mathcal{P}, \mathsf{off})$.
4. Output $(\textsc{SignedOff}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off})$ to $\mathcal{P}$.

Signature Generation (Online Phase) Upon receiving $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L = \{\mathsf{pk}_j\}_{j\in[n]}, m, \mathsf{offs} = \{\mathsf{off}_j\}_{j\in[n]})$ from $\mathcal{P} \in P$.

1. If $\mathcal{P}$ previously called $\textsc{SignOn}$ with the same ssid, then output $(\textsc{fail}, \mathsf{sid})$.
2. Retrieve $(\textsc{keyrec}, \mathcal{P}, \mathsf{pk})$ or output $(\textsc{fail}, \mathsf{sid})$ if no such record exists.
3. If $\mathsf{pk} \notin L$ then output $(\textsc{fail}, \mathsf{sid})$.
4. Else, let $i \in [n]$ be such that $\mathsf{pk}_i = \mathsf{pk}$.
5. If there exists **no** record $(\textsc{presig}, \mathsf{ssid}, \mathcal{P}, \mathsf{off}_i)$, return $(\textsc{fail}, \mathsf{sid})$. ▷ Honest parties don't resume if the offline phase is not completed for ssid and has not produced off_i.
6. Send $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}, L, m, \mathsf{offs})$ to $\mathcal{S}$, and wait for $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_{\mathcal{P}})$.
7. Create record $(\textsc{sig}, \mathsf{ssid}, \mathcal{P}, L, m, \mathsf{offs}, \sigma_{\mathcal{P}})$.
8. Output $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_{\mathcal{P}})$.

Signature Aggregation Upon receiving input $(\textsc{Aggregate}, \mathsf{sid}, L = \{\mathsf{pk}_j\}_{j\in[n]}, m, \mathsf{offs} = \{\mathsf{off}_j\}_{j\in[n]}, \mathsf{ons} = \{\sigma_j\}_{j\in[n]})$ from $\mathcal{P} \in P$:

1. Send $(\textsc{Aggregate}, \mathsf{sid}, L, m, \mathsf{offs}, \mathsf{ons})$ to $\mathcal{S}$, wait for $(\textsc{Aggregated}, \mathsf{sid}, \sigma)$.
2. If for all $i \in [n]$, there exist $\mathcal{P} \in H$ and ssid such that (a) $(\textsc{keyrec}, \mathcal{P}, \mathsf{pk}_i)$ exists, (b) $(\textsc{presig}, \mathsf{ssid}, \mathcal{P}, \mathsf{off}_i)$ exists, and (c) $(\textsc{sig}, \mathsf{ssid}, \mathcal{P}, L, m, \mathsf{offs}, \sigma_i)$ exists:
   - If there exists a record $(\textsc{msig}, L, m, \sigma, 0)$, then abort. ▷ Abort if σ is known to be invalid.
   - Else, create a record $(\textsc{msig}, L, m, \sigma, 1)$ (unless it already exists). ▷ Correctness
3. Output $(\textsc{Aggregated}, \mathsf{sid}, \sigma)$.

Signature Verification Upon receiving input $(\textsc{Verify}, \mathsf{sid}, L = \{\mathsf{pk}_j\}_{j\in[n]}, m, \sigma)$ from $\mathcal{V}$:

1. If there exists a record $(\textsc{msig}, L, m, \sigma, b)$, set $f := b$. ▷ Consistency
2. Else, if there exist $i \in [n]$ and $\mathcal{P} \in H$ such that (a) $(\textsc{keyrec}, \mathcal{P}, \mathsf{pk}_i)$ exists and (b) there exists **no** record $(\textsc{sig}, *, \mathcal{P}, L, m, *, *)$, then set $f := 0$. ▷ Unforgeability
3. Else, send $(\textsc{Valid?}, \mathsf{sid}, L, m, \sigma)$ to $\mathcal{S}$, wait for $(\textsc{Valid?}, \mathsf{sid}, b')$ from $\mathcal{S}$ and set $f := b'$.
4. Create a record $(\textsc{msig}, L, m, \sigma, f)$ and output $(\textsc{Verified}, \mathsf{sid}, f)$.

IMS is not broken, we must bound the probability that $\mathcal{Z}$ causes Verify to accept a signature on (L, m) that has never been explicitly signed by some honest party. This is because in the ideal execution $\mathcal{F}_{\mathsf{IMS}}$ always rejects the signature in that case. This probability can be bounded by the advantage of breaking unforgeability of IMS, but the reduction must pinpoint one of the public keys that an environment uses for creating a forgery, incurring $1/|P|$ reduction loss.

Let us go over each interface.

- KeyGen: The only difference is when $\mathcal{F}_{\mathsf{IMS}}$ aborts after receiving pk from $\mathcal{S}_{\mathsf{IMS}}$. This event occurs with negligible probability assuming the unforgeability of IMS.
- SignOff, SignOn: Since $\mathcal{S}_{\mathsf{IMS}}$ runs $\mathsf{SignOff}, \mathsf{SignOn}$ as in π_{IMS}, there is no difference.
- Aggregate: $\mathcal{S}_{\mathsf{IMS}}$ runs Agg. The only difference is when $\mathcal{F}_{\mathsf{IMS}}$ aborts after receiving σ from $\mathcal{S}_{\mathsf{IMS}}$. This event occurs with negligible probability assuming the correctness and unforgeability of IMS. That is, if every condition of Step 2 is met, then, by the correctness, σ output by Agg must pass the verification with respect to L and m. Thus, if the environment has made $\mathcal{F}_{\mathsf{IMS}}$ record $(\text{MSIG}, L, m, \sigma, 0)$ through the verification interface, such an environment is indeed a successful forger against the IMS scheme.
- Verify: We consider the following cases:
  - If every $\mathsf{pk} \in L$ belongs to an honest party, each honest party has completed SignOn with the same input (L, m, offs), and Aggregate was called on $(L, m, \mathsf{offs}, \mathsf{ons})$: In this case, there is a potential discrepancy in the view of $\mathcal{Z}$ since $\mathcal{F}_{\mathsf{IMS}}$ always outputs a decision bit 1 while π_{IMS} returns the output of $\mathsf{Vrfy}(L, m, \sigma)$. However, if $\mathcal{Z}$ outputs (L, m, σ) causing Vrfy to return 0, this implies $\mathcal{Z}$ breaks the correctness of IMS after adaptively querying the oracles for key generation, offline signing, and online signing. This event thus happens with negligible probability.
  - If for some $\mathsf{pk} \in L$ there exists no record $(\text{KEYREC}, *, \mathsf{pk})$, or for input $(*, L, m, *)$ all honest parties $\mathcal{P} \in H$ have completed SignOn: In this case, $\mathcal{F}_{\mathsf{IMS}}$ asks $\mathcal{S}_{\mathsf{IMS}}$ to simulate a decision bit. Since $\mathcal{S}_{\mathsf{IMS}}$ also runs Vrfy as in π_{IMS}, there is no difference in the view of $\mathcal{Z}$.
  - Else, (i.e., if for input (L, m) some honest party $\mathcal{P} \in H$ has generated $\mathsf{pk} \in L$ through the invocation of $(\text{KeyGen}, \mathsf{sid})$ **and** has **not** completed SignOn for any ssid): In this case, there is a potential discrepancy in the view of $\mathcal{Z}$ since $\mathcal{F}_{\mathsf{IMS}}$ always outputs a decision bit 0 while π_{IMS} returns the output of $\mathsf{Vrfy}(L, m, \sigma)$. However, if $\mathcal{Z}$

manages to come up with (L, m, σ) causing Vrfy to return 1, this implies $\mathcal{Z}$ created a valid forgery in the $\mathsf{IMS\text{-}UF\text{-}CMA}$ game. That is, using such $\mathcal{Z}$ as a subroutine one can construct a reduction $\mathcal{B}$ breaking $\mathsf{IMS\text{-}UF\text{-}CMA}$. On receiving a challenge public key pk, $\mathcal{B}$ with access to $\mathsf{OSignOff}, \mathsf{OSignOn}$ and the random oracle H plays the role of $\mathcal{F}_{\mathsf{IMS}}$, $\mathcal{S}_{\mathsf{IMS}}$, and $\mathcal{G}_{\mathsf{RO}}$ and proceeds as follows.

* Uniformly choose uncorrupted party $\mathcal{P}^* \in P$ and use pk as a public key for $\mathcal{P}^*$. Generate key pairs for all the other honest parties as $\mathcal{S}_{\mathsf{IMS}}$ would.
* Whenever $\mathcal{Z}$ makes a query to $\mathcal{G}_{\mathsf{RO}}$, relay queries and responses to and from H.
* Whenever $\mathcal{S}_{\mathsf{IMS}}$ is asked to run SignOff command for party $\mathcal{P}^*$, query $\mathsf{OSignOff}$ with ssid to receive off and forwards it to $\mathcal{F}_{\mathsf{IMS}}$. Otherwise, SignOff command is handled as in $\mathcal{S}_{\mathsf{IMS}}$.
* Whenever $\mathcal{S}_{\mathsf{IMS}}$ is asked to run SignOn command for party $\mathcal{P}^*$, query $\mathsf{OSignOn}$ with input $(\mathsf{ssid}, L, m, \mathsf{offs})$ to receive σ. Otherwise, SignOn command is handled as in $\mathcal{S}_{\mathsf{IMS}}$.
* If $\mathcal{Z}$ outputs (L, m, σ) causing $\mathsf{Vrfy}(L, m, \sigma) = 1$ while $\mathsf{pk} \in L$ and (L, m) has never been queried to $\mathsf{OSignOn}$, forward this tuple to the $\mathsf{IMS\text{-}UF\text{-}CMA}$ game.

In this way, the reduction $\mathcal{B}$ succeeds in winning the $\mathsf{IMS\text{-}UF\text{-}CMA}$ game as long as the honest party $\mathcal{P}^*$ is correctly guessed. Hence, with a multiplicative factor of loss $1/|H| \geq 1/|P|$, the advantage of $\mathcal{B}$ is non-negligible if $\mathcal{Z}$ finds inconsistency of verification in real and ideal executions with non-negligible probability.

□

3.5 UC-secure IMS implies game-based IMS

We now prove the opposite direction to Section 3.4 and show that a UC-secure IMS implies a construction that is secure according to the game-based definition. Towards this, we overload the following algorithms with oracle access to an UC experiment which either interacts with π_{IMS} or $\mathcal{F}_{\mathsf{IMS}}$:

$\mathsf{KeyGen}()$**:** The key generation algorithm sends $(\text{KeyGen}, \mathsf{sid})$ to $\pi_{\mathsf{IMS}}/\mathcal{F}_{\mathsf{IMS}}$ and obtains $(\text{KeyConf}, \mathsf{sid}, \mathsf{pk})$. It outputs $(\mathsf{pk}, \perp)$ as the public key-signing key pair.

$\mathsf{OSignOff}(\mathsf{ssid})$**:** on input signing session ID ssid, it follows Fig. 5 except that off is retrieved by querying $\pi_{\mathsf{IMS}}/\mathcal{F}_{\mathsf{IMS}}$ with $(\text{SignOff}, \mathsf{sid}, \mathsf{ssid})$.

Protocol 7: π_{IMS} for interactive multi-signature protocol

The protocol is parameterized by $\mathsf{IMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SignOff}, \mathsf{SignOn}, \mathsf{Agg}, \mathsf{Vrfy})$ and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and executed by a set of signers P and an arbitrary $\mathcal{V}$. Whenever an algorithm of IMS queries to a random oracle, π_{IMS} relays queries to and responses from $\mathcal{G}_{\mathsf{RO}}$.

Key generation Upon $(\textsc{KeyGen}, \mathsf{sid})$, $\mathcal{P} \in P$ proceeds as follows:

1. If there exists a record $(\textsc{keyrec}, *, *)$, then output $(\textsc{fail}, \mathsf{sid})$.
2. Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
3. Create the record $(\textsc{keyrec}, \mathsf{sk}, \mathsf{pk})$ and output $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk})$.

Signature Setup (Offline Phase) Upon $(\textsc{SignOff}, \mathsf{sid}, \mathsf{ssid})$, $\mathcal{P} \in P$ proceeds as follows:

1. If $\mathcal{P}$ previously called $\textsc{SignOff}$ with the same ssid, then output $(\textsc{Fail}, \mathsf{sid})$.
2. If there exists **no** record $(\textsc{keyrec}, *, *)$, then output $(\textsc{Fail}, \mathsf{sid})$.
3. Retrieve $(\textsc{keyrec}, \mathsf{sk}, \mathsf{pk})$, run $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$ and create a record $(\textsc{presig}, \mathsf{ssid}, \mathtt{st}, \mathsf{off})$.
4. Output $(\textsc{SignedOff}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off})$.

Signature Generation (Online Phase) Upon receiving $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L = \{\mathsf{pk}_j\}_{j\in[n]}, m, \mathsf{offs} = \{\mathsf{off}_j\}_{j\in[n]})$, $\mathcal{P} \in \mathcal{P}$ proceeds as follows:

1. If $\mathcal{P}$ previously called $\textsc{SignOn}$ with the same ssid, then output $(\textsc{Fail}, \mathsf{sid})$.
2. Retrieve $(\textsc{keyrec}, \mathsf{sk}, \mathsf{pk})$ or output $(\textsc{Fail}, \mathsf{sid})$ if no such record exists.
3. If $\mathsf{pk} \notin L$ then output $(\textsc{Fail}, \mathsf{sid})$.
4. Else, let $i \in [n]$ be such that $\mathsf{pk}_i = \mathsf{pk}$.
5. If there exists **no** record $(\textsc{presig}, \mathsf{ssid}, *, \mathsf{off}_i)$, return $(\textsc{fail}, \mathsf{sid})$.
6. Retrieve $(\textsc{presig}, \mathsf{ssid}, \mathtt{st}, \mathsf{off}_i)$ and run $\sigma_{\mathcal{P}} \leftarrow \mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs})$.
7. Output $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_{\mathcal{P}})$.

Signature Aggregation Upon receiving input $(\textsc{Aggregate}, \mathsf{sid}, L, m, \mathsf{offs}, \mathsf{ons})$, $\mathcal{P} \in P$ runs $\sigma \leftarrow \mathsf{Agg}(L, m, \mathsf{offs}, \mathsf{ons})$ and outputs $(\textsc{Aggregated}, \mathsf{sid}, \sigma)$.

Signature Verification Upon receiving input $(\textsc{Verify}, \mathsf{sid}, L, m, \sigma)$, $\mathcal{V}$ runs $b \leftarrow \mathsf{Vrfy}(L, m, \sigma)$ and outputs $(\textsc{Verified}, \mathsf{sid}, b)$.

$\mathsf{OSignOn}(\mathsf{ssid}, L, m, \mathsf{offs})$**:** on input signing session ID ssid, a set of public keys L, message m, offline tokens offs, it follows Fig. 5 except that σ is retrieved by querying $\pi_{\mathsf{IMS}}/\mathcal{F}_{\mathsf{IMS}}$ with $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L, m, \mathsf{offs})$.

Noticably, the key generation algorithm only outputs an empty string $\perp$ as secret signing key, since such a key is not defined in $\mathcal{F}_{\mathsf{IMS}}$.

Theorem 8. *Let $\mathcal{F}_{\mathsf{IMS}}$ be an ideal functionality, π_{IMS} be defined as Fig. 7 and $n \in \mathsf{poly}(\lambda)$. If the protocol π_{IMS} defined by an interactive multi-signature scheme* IMS *UC-realizes $\mathcal{F}_{\mathsf{IMS}}$ in the presence of $\mathcal{G}_{\mathsf{RO}}$, then* IMS *is correct (Definition 5) and* $\mathsf{IMS\text{-}UF\text{-}CMA}$ *secure (Definition 6) in the random oracle model.*

Simulator 8: $\mathcal{S}_{\mathsf{IMS}}$ interacting with $\mathcal{F}_{\mathsf{IMS}}$

$\mathcal{S}_{\mathsf{IMS}}$ interacts with an internal copy of $\mathcal{A}$, towards which it simulates the honest parties in P and relays all the random oracle calls requested by SignOff, SignOn, and Vrfy externally to the global random oracles functionality $\mathcal{G}_{\mathsf{RO}}$.

KeyGen: On receiving $(\textsc{KeyGen}, \mathsf{sid}, \mathcal{P})$ from $\mathcal{F}_{\mathsf{IMS}}$, run KeyGen to obtain the key pair $(\mathsf{sk}, \mathsf{pk})$. Create a record $(\mathcal{P}, \mathsf{sk}, \mathsf{pk})$. Return $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk})$ to $\mathcal{F}_{\mathsf{IMS}}$.

Sign Setup: On receiving $(\textsc{SignOff}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}, \mathsf{pk})$ from $\mathcal{F}_{\mathsf{IMS}}$:
1. Retrieve a record $(\mathcal{P}, \mathsf{sk}, \mathsf{pk})$
2. Run $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$ and create a record $(\mathsf{ssid}, \mathcal{P}, \mathtt{st}, \mathsf{off})$.
3. Output $(\textsc{Presig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off})$ to $\mathcal{F}_{\mathsf{IMS}}$

Sign Gen: On receiving $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}, L, m, \mathsf{offs})$ from $\mathcal{F}_{\mathsf{IMS}}$, retrieve the corresponding stored $\mathtt{st}$, then run $\sigma \leftarrow \mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs})$, then return $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ to $\mathcal{F}_{\mathsf{IMS}}$.

Sign Agg: On receiving $(\textsc{aggregate}, \mathsf{sid}, L, m, \mathsf{offs}, \mathsf{ons})$ from $\mathcal{F}_{\mathsf{IMS}}$, run $\sigma = \mathsf{Agg}(L, m, \mathsf{offs}, \mathsf{ons})$, return $(\textsc{Aggregated}, \mathsf{sid}, \sigma)$ to $\mathcal{F}_{\mathsf{IMS}}$.

Verify: On receiving $(\textsc{Valid?}, \mathsf{sid}, L, m, \sigma)$, run $b \leftarrow \mathsf{Vrfy}(L, m, \sigma)$ and return $(\textsc{Valid?}, \mathsf{sid}, b)$.

Proof. Correctness can be ensured by inspection of the algorithm and the functionality. That is, if IMS is not correct, there exists an adversary $\mathcal{A}$ that causes Agg to output an invalid signature with non-negligible probability even if its inputs are produced by honest parties (Definition 5). Using such $\mathcal{A}$, we can construct an environment that distinguishes whether it's interacting with $\mathcal{F}_{\mathsf{IMS}}$ or π_{IMS} as follows: the environment calls key generation upon $\mathcal{A}$ querying OKeyReg, signature setup upon $\mathcal{A}$ querying OSignOff, and signature generation upon $\mathcal{A}$ querying OSignOn for all honest parties, generates a signature σ by calling the aggregation interface once for an arbitrary party, and then verifies σ with respect to the message and the public key set used earlier. If in the real world, the environment observes the return value of verification being 0 with non-negligible probability; if in the ideal world, the the return value of verification is always 1. Thus, this is a valid distinguisher with non-negligible advantage.

Towards proving unforgeability consider that the oracles are already fully defined. Assume that there exists an attacker $\mathcal{A}$ which can win Game IMS-UF-CMA with non-negligible probability. The game is won if it outputs 1, and it simulates signatures under a challenge public key pk (belonging to one of the honest parties) whose corresponding secret key is never given to $\mathcal{A}$. We then show the existence of an environment $\mathcal{Z}$ such that for any ideal adversary $\mathcal{S}$, $\mathcal{Z}$ can tell whether it is interacting with π_{IMS} or $\mathcal{F}_{\mathsf{IMS}}$. To this end, we construct $\mathcal{Z}$ that runs a copy of game-based adversary $\mathcal{A}$ and simulates the view of $\mathcal{A}$ in $\mathsf{IMS\text{-}UF\text{-}CMA}$ game. This is done by mapping return values of $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{OSignOff}, \mathsf{OSignOn}$ to responses from the corresponding interfaces in an UC experiment as above. Moreover, $\mathcal{Z}$ relays any random oracle queries made by $\mathcal{A}$ to $\mathcal{G}_{\mathsf{RO}}$ and returns $\mathcal{G}_{\mathsf{RO}}$'s response to $\mathcal{A}$.

Upon receiving a forgery tuple (L^*, m^*, σ^*) from $\mathcal{A}$, the game simulated by $\mathcal{Z}$ would output 1 if and only if $\mathsf{Vrfy}(L^*, m^*, \sigma^*) = 1$. This requires that $\mathsf{pk} \in L^*$. Moreover, **Signature Generation (Online Phase)** of $\pi_{\mathsf{IMS}}/\mathcal{F}_{\mathsf{IMS}}$ has never been called for L^*, m^* by the oracle $\mathsf{OSignOn}$ as the attacker would otherwise have lost the game. $\mathcal{Z}$ finally queries $\pi_{\mathsf{IMS}}/\mathcal{F}_{\mathsf{IMS}}$ with input $(\textsc{Verify}, \mathsf{sid}, L^*, m^*, \sigma^*)$ to obtain its result and outputs what it outputs. Here, if $\mathcal{Z}$ is interacting with the ideal process, $\mathcal{F}_{\mathsf{IMS}}$ never outputs $(\textsc{Verified}, \mathsf{sid}, 1)$ due to Step 2. However, if $\mathcal{Z}$ is interacting with the real process, π_{IMS} outputs $(\textsc{Verified}, \mathsf{sid}, 1)$ as long as $\mathcal{A}$ wins the $\mathsf{IMS\text{-}UF\text{-}CMA}$ game. Therefore, $\mathcal{Z}$ has non-negligible advantage in the UC experiment.

□

4 Ordered Multi-Signatures

In this section, we provide the syntax (Section 4.1) for ordered multi-signatures (OMS), a game-based security notion for OMS (Section 4.2), and an ideal functionality $\mathcal{F}_{\mathsf{OMS}}$ for OMS (Section 4.3). We then prove that any scheme secure according to the game-based OMS security notion also UC-realizes $\mathcal{F}_{\mathsf{OMS}}$ (Section 4.4).

4.1 Syntax

For the syntax, our starting point are the (unordered) multi-signatures MuSig2 [NRS21], where we shed algorithms related to key aggregation but keep the two-phase signing (through the algorithms $\mathsf{SignOff}$ and SignOn). This allows for offloading most of the overhead to the offline phase ($\mathsf{SignOff}$), which can be preprocessed before the messages are known. Unlike the usual multi-signatures, SignOn now takes an *aggregate so-far* as input and updates it with contribution by the current signer.

Definition 9 (Ordered Multi-Signature Scheme (OMS)). *An ordered multi-signature* OMS *consists of a tuple of algorithms* $\mathsf{OMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SignOff}, \mathsf{SignOn}, \mathsf{Vrfy})$ *with the following input-output behavior.*

To formally handle the interactive signing, $\mathsf{SignOff}$ *and* SignOn *share a common state* $\mathtt{st}$*.*

$\mathsf{Setup}(1^\lambda)$**:** *on input the security parameter* λ*, this algorithm outputs a handle of public parameters* pp*, which in particular includes an initial aggregate so far* σ_0 *as a public constant. Throughout, we assume that* pp *is given as implicit input to all other algorithms.*

$\mathsf{KeyGen}()$**:** *on input the public parameters, the key generation algorithm outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{SignOff}(\mathsf{sk})$**:** *on input a secret key* sk*, the offline signing algorithm outputs an offline token* off *and an internal state* $\mathtt{st}$*. We note that this algorithm is agnostic of the message* m *to be signed, and runs independently of* m*.*

$\mathsf{SignOn}(\mathtt{st}, \mathsf{sk}, L, m, \mathsf{offs}, \sigma)$**:** *on input a secret key* sk*, an ordered list of public keys* L*, a message* m*, some offline-token information* offs*, and a so-far-aggregated online signature* σ*, the online signing algorithm outputs a tuple* (σ', b')*:* σ' *is a new so-far-aggregated signature* σ'*, which may contain a special symbol:* $\sigma' = \varepsilon$ *if the inputs are not well-formed, and* b' *is a bit indicating validity of the input so-far-aggregate signature* σ' *w.r.t.* L *and* m*.*

$\mathsf{Vrfy}(L, m, \sigma)$**:** *on input an ordered list of public keys* L*, a message* m*, and a signature* σ*, the verification algorithm outputs* 1 *(accept) or* 0 *(reject).*

While $\mathsf{SignOff}$ can run without a specified signer order, honest executions of OMS demand SignOn be run in a sequential manner. That is to say, SignOn takes in input offline tokens offs (usually one for each co-signer in L) and the online output σ_{i-1} of co-signer $i-1$, and passes their output σ_i on to the co-signer in position $i+1$. The signer in position $i = 1$ receives σ_0 as initial input, where σ_0 is some constant specified in pp, and the signer in position $i = n$ outputs $\sigma_n = \sigma$ as the final OMS.

4.2 Game-based Security of OMS

The original OMS syntax and security model defined by [BGOY07] are tailored to non-interactive schemes. We adapt the security game of [BGOY07] so as to capture *interactive* and *offline-online* signing as required by our syntax. Notably, Ordered Mult-Signatures not only need to be unforgeable in the standard sense but must also enforce the ordering on the signers. In [BGOY07], this last requirement is considered for the worst case where all but one co-signer are corrupted. Our security model is more complex and takes into account forgeries that involve *at least one, but potentially* *more* honest co-signers, a more realistic and general setting than the one in [BGOY07]. We first define correctness. Following the correctness game for IMS (Definition 5), our OMS correctness game allows an adversary to adaptively query the key registration and signing oracles. The adversary then wins if for any $i \in [n]$, an aggregate so-far σ_i fails to pass verification while there exists a sequence of preceding signatures $(\sigma_1, \ldots, \sigma_{i-1})$ that have been generated honestly.

Definition 10 (OMS Correctness). *An interactive multi signature* OMS *is said to be correct if for any probabilistic polynomial time adversary* $\mathcal{A}$*, the probability* $\Pr[\mathsf{OMS\text{-}COR}(\mathcal{A}, \lambda) = 1]$ *is negligible in* λ*, where the* $\mathsf{OMS\text{-}COR}$ *experiment is defined in Figure 9.*

Unforgeability Our security model for OMS aims at capturing the two following conditions. The signature shall be *unforgeable*, that is, as long as at least one signer in L is honest (i.e., the adversary has no access to the signer's secret key) it should be computationally infeasible to generate a signature σ^* that verifies for a *new* message (not queried during the game) or against a *new* list of co-signers L^*. Moreover *order matters*, so an adversary $\mathcal{A}$ that has access to a subset C of the keys identified by an ordered signer list L should be unable to generate a signature σ^* that verifies for a list $L^* \neq L$ where the public keys differ, or the position of some honest signers is changed. That is to say, we expect to detect all order changes except for shuffling of dishonest parties. This latter property separates OMS from MS: in the OMS security game, $\mathcal{A}$ may win by outputting a valid signature on $(m, L^* = (\mathsf{pk}, \mathsf{pk}'))$ after querying $(m, L = (\mathsf{pk}', \mathsf{pk}))$ to the signing oracle; whereas in the usual MS security game L^* is not considered new, and thus the adversary cannot win with such an output.

Following the original formulation due to [BGOY07], we require the key pairs to be *registered*, that is, $\mathcal{A}$ must prove the knowledge of its secret keys in advance, moreover any query or forgery attempt that involves at least one unregistered key in L gets rejected. This requirement essentially models the situation where users are asked to prove knowledge of their secret keys and corresponding randomness during public-key registration with a CA. Such registered key (RK) model [RY07] is weaker than the typical plain public key (PPK) model, where $\mathcal{A}$ may use arbitrary public keys for which it may not know the corresponding secret keys.

Definition 11 (OMS Security). *An ordered multi signature* OMS *is said to be secure if for any probabilistic polynomial time adversary* $\mathcal{A}$*, the*

Figure 9: Game-based correctness for OMS

```
Game OMS-COR($\mathcal{A}, \lambda$)
  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
  $\mathcal{K} := \emptyset$; $\mathcal{T} := \emptyset$; $\mathcal{Q} := \emptyset$
  $\mathsf{win} := 0$
  $\mathcal{O} := \{\mathsf{OKeyReg}, \mathsf{OSignOff}, \mathsf{OSignOn}\}$ ▷ RO in $\mathcal{O}$ if needed
  $(L, m, \mathsf{offs}, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
  $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) := L$
  $(\mathsf{off}_1, \ldots, \mathsf{off}_n) := \mathsf{offs}$
  $\sigma_0 := \mathsf{pp}.\sigma_0$; $\sigma_n := \sigma$
  If $\forall j \in [n]\ \exists \mathsf{uid}_j : \mathcal{K}[\mathsf{uid}_j] = (1, \mathsf{pk}_j, \cdot)$
      $\wedge \forall j \in [n]\ \exists \mathsf{uid}_j, \mathsf{ssid}_j :$
          $\mathcal{T}[\mathsf{ssid}_j, \mathsf{uid}_j] = (\cdot, \mathsf{off}_j)$
      $\wedge \exists (\sigma_1, \ldots, \sigma_{n-1})\ \forall j \in [n] :$
          $(L, m, \mathsf{offs}, \sigma_{j-1}, \sigma_j, 1, j) \in \mathcal{Q}$
      $\wedge \mathsf{Vrfy}(L, m, \sigma_n) = 0$ then
    $\mathsf{win} := 1$
  return $\mathsf{win}$

Oracle OKeyReg($\mathsf{uid}, \mathsf{aux}$)
  if $\mathcal{K}[\mathsf{uid}] \neq \perp$ then
    return $\perp$
  if $\mathsf{aux} = \epsilon$ then ▷ Register an honest key
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
    $\mathcal{K}[\mathsf{uid}] := (1, \mathsf{pk}, \mathsf{sk})$
  else ▷ Register a corrupt key
    parse $\mathsf{aux} = (\mathsf{pk}, \mathsf{sk}, \rho)$
    if $(\mathsf{pk}, \mathsf{sk}) \neq \mathsf{KeyGen}(1^\lambda; \rho)$ then
      return $\perp$
    $\mathcal{K}[\mathsf{uid}] := (0, \mathsf{pk}, \mathsf{sk})$
  return $\mathsf{pk}$

Oracle OSignOff($\mathsf{ssid}, \mathsf{uid}$)
  $(b, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}[\mathsf{uid}]$
  if $b = 0 \vee \mathsf{sk} = \epsilon$ then return $\perp$
  if $\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] \neq \perp$ then return $\perp$
  $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$
  $\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] := (\mathtt{st}, \mathsf{off})$
  return $\mathsf{off}$

Oracle OSignOn($\mathsf{ssid}, \mathsf{uid}, L, m, \mathsf{offs}, \sigma_{i-1}, i$)
  $(b, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}[\mathsf{uid}]$;
  $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) := L$; $(\mathsf{off}_1, \ldots, \mathsf{off}_n) := \mathsf{offs}$
  if $b = 0 \vee \mathsf{sk} = \epsilon$ then return $\perp$
  if $(\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] \neq (\cdot, \mathsf{off}_i)) \vee (\mathcal{Q}[\mathsf{ssid}, \mathsf{uid}] \neq \perp)$
  then return $\perp$ ▷ Invalid session
  if $\mathsf{pk} \neq \mathsf{pk}_i$ then return $\perp$ ▷ Wrong position
  if $i = 1 \wedge \sigma_{i-1} \neq \mathsf{pp}.\sigma_0$ then return $\perp$ ▷ Invalid initial input
  if $\exists j \in [n] : (*, \mathsf{pk}_j, *) \notin \mathcal{K}$ then
    return $\perp$ ▷ Unregistered key
  $(\mathtt{st}_i, \mathsf{off}_i) := \mathcal{T}[\mathsf{ssid}, \mathsf{uid}]$
  $(\sigma_i, b_{i-1}) \leftarrow \mathsf{SignOn}(\mathtt{st}_i, \mathsf{sk}, L, m, \mathsf{offs}, \sigma_{i-1})$
  $\mathcal{Q}[\mathsf{ssid}, \mathsf{uid}] := (L, m, \mathsf{offs}, \sigma_{i-1}, \sigma_i, b_{i-1}, i)$
  If $\forall j \in [n]\ \exists \mathsf{uid}_j : \mathcal{K}[\mathsf{uid}_j] = (1, \mathsf{pk}_j, \cdot)$
      $\wedge \forall j \in [n]\ \exists \mathsf{uid}_j, \mathsf{ssid}_j :$
          $\mathcal{T}[\mathsf{ssid}_j, \mathsf{uid}_j] = (\cdot, \mathsf{off}_j)$
      $\wedge \exists (\sigma_1, \ldots, \sigma_{i-1})\ \forall j \in [i-1] :$
          $(L, m, \mathsf{offs}, \sigma_{j-1}, \sigma_j, 1, j) \in \mathcal{Q}$
      $\wedge b_{i-1} = 0$ then
    $\mathsf{win} := 1$
  return $(\sigma_i, b_{i-1})$
```

following probability is negligible in λ:

$$\mathbf{Adv}^{\mathsf{OMS\text{-}UF\text{-}CMA}}_{\mathsf{OMS}}(\mathcal{A}, \lambda) := \Pr[\mathsf{OMS\text{-}UF\text{-}CMA}(\mathcal{A}, \lambda) = 1]$$

where $\mathsf{OMS\text{-}UF\text{-}CMA}$ *is the security-game for unforgeability under chosen-message attack of ordered multi-signatures defined in Figure 10.*

Remark 12. While we take inspiration from the OMS security model of [BGOY07], we also identify a weakness that motivates our new model. In [BGOY07] security holds for one single honest signer in L. While this setting may seem stronger than ours, it does not capture more complex scenarios where the adversary tries to swap the signing order of honest parties, which is the goal of this work. In particular, in [BGOY07] a signature output by an honest signer authenticates the message m and the signer's position i in the chain, but not the order of honest signers that

Figure 10: Game-based unforgeability for OMS

```
GAME OMS-UF-CMA($\mathcal{A}, \lambda$)
1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2: $\mathcal{K} := \emptyset$; $\mathcal{T} := \emptyset$; $\mathcal{Q} := \emptyset$
3: $\mathsf{win} := 0$
4: $\mathcal{O} := \{\mathsf{OKeyReg}, \mathsf{OSignOff}, \mathsf{OSignOn}\}$ ▷ RO in $\mathcal{O}$ if needed
5: $(L^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
6: if $\mathsf{win} = 1$ then return $1$
7: $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) := L^*$
8: if $\exists \mathsf{pk} \in L^*$ s.t. $(\cdot, \mathsf{pk}, \cdot) \notin \mathcal{K}$ then return $0$
9: $b^* \leftarrow \mathsf{Vrfy}(L^*, m^*, \sigma^*)$
10: if $b^* = 1 \wedge \exists j \in [n] : ((1, \mathsf{pk}_j, \cdot) \in \mathcal{K} \wedge (m^*, L^*, j) \notin \mathcal{Q})$ then return $1$
11: else return $0$

ORACLE OKeyReg($\mathsf{uid}, \mathsf{aux}$)
1: if $\mathcal{K}[\mathsf{uid}] \neq \bot$ then
2:   return $\bot$
3: if $\mathsf{aux} = \epsilon$ then ▷ Register an honest key
4:   $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
5:   $\mathcal{K}[\mathsf{uid}] := (1, \mathsf{pk}, \mathsf{sk})$
6: else ▷ Register a corrupt key
7:   parse $\mathsf{aux} = (\mathsf{pk}, \mathsf{sk}, \rho)$
8:   if $(\mathsf{pk}, \mathsf{sk}) \neq \mathsf{KeyGen}(1^\lambda; \rho)$ then
9:     return $\bot$
10:  $\mathcal{K}[\mathsf{uid}] := (0, \mathsf{pk}, \mathsf{sk})$
11: return $\mathsf{pk}$

ORACLE OSignOff($\mathsf{ssid}, \mathsf{uid}$)
1: $(b, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}[\mathsf{uid}]$
2: if $b = 0 \vee \mathsf{sk} = \epsilon$ then return $\bot$
3: if $\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] \neq \bot$ then return $\bot$
4: $(\mathsf{off}, \mathtt{st}) \leftarrow \mathsf{SignOff}(\mathsf{sk})$
5: $\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] := (\mathtt{st}, \mathsf{off})$
6: return $\mathsf{off}$

ORACLE OSignOn($\mathsf{ssid}, \mathsf{uid}, L, m, \mathsf{offs}, \sigma_{i-1}, i$)
1: $(b, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}[\mathsf{uid}]$;
2: $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) := L$; $(\mathsf{off}_1, \ldots, \mathsf{off}_n) := \mathsf{offs}$
3: if $b = 0 \vee \mathsf{sk} = \epsilon$ then return $\bot$
4: if $(\mathcal{T}[\mathsf{ssid}, \mathsf{uid}] \neq (\cdot, \mathsf{off}_i)) \vee (\mathcal{Q}[\mathsf{ssid}, \mathsf{uid}] \neq \bot)$ then return $\bot$ ▷ Invalid session
5: if $\mathsf{pk} \neq \mathsf{pk}_i$ then return $\bot$ ▷ Wrong position
6: if $i = 1 \wedge \sigma_{i-1} \neq \mathsf{pp}.\sigma_0$ then return $\bot$ ▷ Invalid initial input
7: if $\exists j \in [n] : (*, \mathsf{pk}_j, *) \notin \mathcal{K}$ then
8:   return $\bot$ ▷ Unregistered key
9: $(\mathtt{st}_i, \mathsf{off}_i) := \mathcal{T}[\mathsf{ssid}, \mathsf{uid}]$
10: $(\sigma_i, b_{i-1}) \leftarrow \mathsf{SignOn}(\mathtt{st}_i, \mathsf{sk}, L, m, \mathsf{offs}, \sigma_{i-1})$
11: $\mathcal{Q}[\mathsf{ssid}, \mathsf{uid}] := (m, L, i)$
12: if $(b_{i-1} = 1) \wedge (\exists j \in [i-1] : ((1, \mathsf{pk}_j, \cdot) \in \mathcal{K} \wedge (m, L, j) \notin \mathcal{Q}))$ then
13:   $\mathsf{win} := 1$ ▷ order forgery: party $i$ verifies but honest party $j < i$ has not signed yet
14: return $(\sigma_i, b_{i-1})$
```

contribute before and after position i. In more detail, [BGOY07] considers a forgery to be successful if (among other trivial checks) $\mathcal{A}$ never queried the signing oracle with (m^*, σ', L') for the same m^* as in the forgery, $|L'| = i^* - 1$ and i^* is the position of the honest signer in L^*, and $\sigma' \in \{0,1\}^*$ is arbitrary. This constraint is needed as in [BGOY07] extensions of an ordered multi signature to a longer set of signers $L' = L|\mathsf{pk}'$ are trivial, which is not the case in our construction.

4.3 Ideal Functionality for Ordered Multi-Signatures

Finally, we present $\mathcal{F}_{\mathsf{OMS}}$ (Figure 11) to model *ordered* interactive multi-signatures. The functionality essentially works like $\mathcal{F}_{\mathsf{IMS}}$, with the difference being that 1) key registration phase requires corrupted parties to supply random coins ρ for key generation, and 2) the signature generation (or online phase) of $\mathcal{P}_i$ depends on σ_{i-1} as well as the offline shares offs. σ_n will then be the signature, as $\mathcal{F}_{\mathsf{OMS}}$ has no aggregation interface since OMS assumes every party in a sequence to perform partial aggregation as it contributes to signing. To model the signing order guarantee, Step 6 of the online phase always aborts if it detects some prior (honest) signer in the list that has never agreed to generate a partial signature for the same (L, m).

4.4 UC-security of Ordered Multi-Signatures in the Key Registration Model

In Fig. 13 we define π_{OMS} as a direct instantiation of OMS. To realize a protocol in the key registration model, we assume parties have access to the verified key registration functionality $\mathcal{F}_{\mathsf{vReg}}$ (Functionality 12). This ideal functionality captures a public key infrastructure, allowing parties to register their public keys in such a way that other parties can retrieve public keys with the guarantee that they belong to the party who originally registered them. $\mathcal{F}_{\mathsf{vReg}}$ is inspired by the functionalities from [BCNP04, CH06], providing the ability to register a single public key per party from [CH06] (as in a public key infrastructure) while supporting the knowledge of secret key (KOSK) requirement [Bol03] or the key verification [BCJ08, DEF$^+$19] via a mechanism similar to that of [BCNP04]. That is, any party registering their public key is required to prove possession of the corresponding secret key, which is captured by either generating a valid key pair using fresh randomness (in case of an honest request) or requiring the adversary to provide randomness for the key generation algorithm (in case of a corrupted party request). Such

Functionality 11: $\mathcal{F}_{\mathsf{OMS}}$ for ordered multi-signatures in the key registration model

The functionality interacts with n signers $P = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, an arbitrary verifier $\mathcal{V}$, and a simulator $\mathcal{S}$. The adversary can corrupt $c < n$ signers, denoted by the set $C \subsetneq \mathcal{P}$. The functionality is parameterized by $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and KeyGen. We denote a set of honest parties by $H = P \setminus C$. We assume sid encodes identities of all possible signers P. We distinguish different offline phases by using unique ssids. In what follows, L denotes the list of signers' public keys, ordered according to the index of the corresponding signer, i.e., let pk_i denote the public key of signer $\mathcal{P}_i$, then $L = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. Signature generation, and verification only accept input containing L such that for all $i \in [n]$, the public key pk_i is registered for $\mathcal{P}_i$ in the functionality.

Adversarial Key Registration On receiving $(\textsc{Register}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pk}, \mathsf{sk}, \rho)$ from $\mathcal{S}$:
1. If $\mathcal{P}_i \in C$, $(\mathsf{pk}, \mathsf{sk}) = \mathsf{KeyGen}(1^\lambda; \rho)$ and there exists **no** record $(\textsc{keyrec}, \mathcal{P}_i, *)$, then create a record $(\textsc{keyrec}, \mathcal{P}_i, \mathsf{pk})$
2. Else, ignore the request.

Key Registration Upon receiving input $(\textsc{Register}, \mathsf{sid})$ from a signer $\mathcal{P}_i \in P \setminus C$:
1. If there exists a record $(\textsc{keyrec}, \mathcal{P}_i, *)$, then output $(\textsc{fail}, \mathsf{sid})$. $\triangleright$ Prevent overwriting
2. Send $(\textsc{Register}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{S}$ and wait for $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk}_i)$ from $\mathcal{S}$.
3. If there exists a record $(\textsc{msig}, L, *, *, *)$ such that $\mathsf{pk}_i \in L$, then abort. $\triangleright$ Enforce distinct public keys
4. Create record $(\textsc{keyrec}, \mathcal{P}_i, \mathsf{pk}_i)$ and output $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk}_i)$ to $\mathcal{P}_i$.

Signature Setup (Offline Phase) Upon receiving $(\textsc{SignOff}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}_i \in P$:
1. Retrieve $(\textsc{keyrec}, \mathcal{P}_i, \mathsf{pk}_i)$ or output $(\textsc{fail}, \mathsf{sid})$ if no such record exists.
2. If $\mathcal{P}_i \in H$ and $\mathcal{P}_i$ previously made a request with the same ssid, then output $(\textsc{fail}, \mathsf{sid})$. $\triangleright$ Honest parties will only use ssid once.
3. Else, send $(\textsc{Presig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}_i)$ to $\mathcal{S}$, wait for response $(\textsc{Presig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off}_i)$. Then create a record $(\textsc{presig}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{off}_i)$.
4. Output $(\textsc{SignedOff}, \mathsf{sid}, \mathsf{ssid}, \mathsf{off}_i)$ to $\mathcal{P}_i$.

Signature Generation (Online Phase) Upon receiving input $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), m, \mathsf{offs} = (\mathsf{off}_1, \ldots, \mathsf{off}_n), \sigma_{i-1})$ from any $\mathcal{P}_i \in P$:
1. Check that there exists a record $(\textsc{keyrec}, \mathcal{P}_j, \mathsf{pk}_j)$ for all $j \in [n]$. If not, return $(\textsc{fail}, \mathsf{sid})$.
2. If $\mathcal{P}_i \in H$ and there exists **no** record $(\textsc{presig}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{off}_i)$, return $(\textsc{fail}, \mathsf{sid})$. $\triangleright$ Honest parties don't resume if the offline phase is not completed for ssid.
3. If $\mathcal{P}_i \in H$ and $\mathcal{P}_i$ previously made a request with the same ssid, return $(\textsc{fail}, \mathsf{sid})$. $\triangleright$ Honest parties only issue one partial signature per ssid.
4. If $i = 1$ and $\sigma_{i-1} \neq \mathsf{pp}.\sigma_0$, return $(\textsc{fail}, \mathsf{sid})$.
5. Send $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, L, m, \mathsf{offs}, \sigma_{i-1})$ to $\mathcal{S}$, and wait for $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_i, b_{i-1})$.
6. Set f_{i-1} as follows:
   - If there exists a record $(\textsc{sig}, \cdot, \mathcal{P}_{i-1}, L, m, \mathsf{offs}, \cdot, \sigma_{i-1}, b'_{i-1})$, set $f_{i-1} := b'_{i-1}$
   - Else, if for some $1 \leq j < i \leq n$ with $\mathcal{P}_j \in H$, there exists **no** record $(\textsc{sig}, \cdot, \mathcal{P}_j, L, m, \mathsf{offs}, \cdot, \cdot, \cdot)$, set $f_{i-1} := 0$. $\triangleright$ If some honest signer prior to i has not signed yet, it must be detected
   - Else, set $f_{i-1} := b_{i-1}$.
7. If (a) $\forall j \in [n]\ \mathcal{P}_j \in H$, (b) $\forall j \in [n]\ \exists(\textsc{presig}, \cdot, \mathcal{P}_j, \mathsf{off}_j)$, (c) $\exists(\sigma_1, \ldots, \sigma_{i-2})\ \forall j \in [i-1]$: $\exists(\textsc{sig}, \mathsf{ssid}, \mathcal{P}_j, L, m, \mathsf{offs}, \sigma_{j-1}, \sigma_j, 1)$ $\triangleright$ Correctness
   - If there exists a record $(\textsc{sig}, \cdot, \mathcal{P}_i, L, m, \mathsf{offs}, \sigma_{i-1}, \sigma_i, 0)$, then abort
   - Else, create a record $(\textsc{sig}, \mathsf{ssid}, \mathcal{P}_i, L, m, \mathsf{offs}, \sigma_{i-1}, \sigma_i, 1)$
8. If $i = n$ and conditions (a)-(c) above are satisfied: $\triangleright$ Correctness
   - If there exists a record $(\textsc{msig}, L, m, \sigma_n, 0)$, then abort
   - Else, create a record $(\textsc{msig}, L, m, \sigma_n, 1)$
9. Output $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_i, f_{i-1})$

Signature Verification Upon receiving input $(\textsc{Verify}, \mathsf{sid}, L = (\mathsf{pk}_i)_{i\in[n]}, m, \sigma)$ from some $\mathcal{V}$:
1. Check that there exists a record $(\textsc{keyrec}, \mathcal{P}_i, \mathsf{pk}_i)$ for all $i \in [n]$. If not, return $(\textsc{fail}, \mathsf{sid})$.
2. If there exists a record $(\textsc{msig}, L, m, \sigma, b)$, set $f := b$. $\triangleright$ Consistency
3. Else, if for some $\mathcal{P}_i \in H$, there exists **no** record $(\textsc{sig}, *, \mathcal{P}_i, L, m, *, *, *, *)$, then set $f := 0$. $\triangleright$ Unforgeability
4. Else, send $(\textsc{Valid?}, \mathsf{sid}, L, m, \sigma)$ to $\mathcal{S}$, wait for $(\textsc{Valid?}, \mathsf{sid}, L, m, \sigma, b')$ from $\mathcal{S}$ and set $f := b'$.
5. Create a record $(\textsc{msig}, L, m, \sigma, f)$ and output $(\textsc{Verified}, \mathsf{sid}, f)$.

Functionality 12: $\mathcal{F}_{\mathsf{vReg}}$ for verified key registration

$\mathcal{F}_{\mathsf{vReg}}$ is parametrized by some key generation algorithm KeyGen (e.g., signature key pair generation) and interacts with a set of parties P and an ideal adversary $\mathcal{S}$.

Key Registration: Upon receiving a message $(\textsc{Register}, \mathsf{sid})$ from an uncorrupted party $\mathcal{P}_i \in P$:

1. Sample ρ uniformly at random
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda; \rho)$
3. Send $(\textsc{Registering}, \mathsf{sid}, \mathsf{pk}, \mathcal{P}_i)$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, ok, \mathcal{P}_i)$ from $\mathcal{S}$, and if this is the first message from $\mathcal{P}_i$, then record the tuple $(\mathcal{P}_i, \mathsf{pk})$ and return $(\textsc{Registered}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ to $\mathcal{P}_i$; else, return $(\textsc{Registered}, \mathsf{sid}, \perp)$ to $\mathcal{P}_i$.

Adversarial Key Registration: Upon receiving a message $(\textsc{Register}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, \rho)$ from a corrupted party $\mathcal{P}_i \in P$:

1. If $\mathsf{KeyGen}(1^\lambda; \rho) \neq (\mathsf{pk}, \mathsf{sk})$, then return $(\textsc{Registered}, \mathsf{sid}, \perp)$.
2. If there exists **no** record $(\mathcal{P}_i, *)$, then record the tuple $(\mathcal{P}_i, \mathsf{pk})$

Key Retrieval: Upon receiving a message $(\textsc{Retrieve}, \mathsf{sid}, \mathcal{P}_j)$ from a party $\mathcal{P}_i \in P$, send message $(\textsc{Retrieve}, \mathsf{sid}, \mathcal{P}_j)$ to $\mathcal{S}$ and wait for it to return a message $(\textsc{Retrieve}, \mathsf{sid}, ok)$. Then, if there is a recorded tuple $(\mathcal{P}_j, \mathsf{pk})$ output $(\textsc{Retrieve}, \mathsf{sid}, \mathcal{P}_j, \mathsf{pk})$ to $\mathcal{P}_i$. Otherwise, if there is no recorded tuple, return $(\textsc{Retrieve}, \mathsf{sid}, \mathcal{P}_j, \perp)$.

functionality can be realized using efficient non-interactive proof of knowledge [RY07]. We introduce this requirement to model the key regstration oracle present in the OMS security model originated in [BGOY07].

The following theorems are OMS analogues of Theorems 7 and 8 from previous section. As the proofs are similar, we defer them to Appendix A.

Theorem 13. *Let* OMS *be an ordered multi-signature scheme (Definition 9) that is correct and* $\mathsf{OMS\text{-}UF\text{-}CMA}$ *secure in the random oracle model (Definition 11). Then the protocol* π_{OMS} *(Figure 13) UC-realizes* $\mathcal{F}_{\mathsf{OMS}}$ *(Figure 11) in the* $\mathcal{F}_{\mathsf{vReg}}$*-hybrid model in the presence of* $\mathcal{G}_{\mathsf{RO}}$ *against a static active adversary corrupting a majority of parties in* P*.*

Theorem 14. *Let* $\mathcal{F}_{\mathsf{OMS}}$ *be an ideal functionality,* π_{OMS} *be defined as Fig. 7. If the protocol* π_{OMS} *in the* $\mathcal{F}_{\mathsf{vReg}}$*-hybrid model UC-realizes* $\mathcal{F}_{\mathsf{OMS}}$ *in the presence of* $\mathcal{G}_{\mathsf{RO}}$*, then* OMS *is correct (Definition 10) and* $\mathsf{OMS\text{-}UF\text{-}CMA}$ *secure (Definition 11) in the random oracle model.*

Protocol 13: π_{OMS} for interactive ordered multi-signature protocol

The protocol is parameterized by $\mathsf{OMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SignOff}, \mathsf{SignOn}, \mathsf{Vrfy})$ and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ executed by n signers $P = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ and an arbitrary $\mathcal{V}$. Whenever an algorithm of OMS queries to a random oracle, π_{OMS} relays queries to and responses from $\mathcal{G}_{\mathsf{RO}}$. We assume that pp includes a description of σ_0 and initializates the internal states $\mathtt{st}_i$ to empty strings. The Signature Generation and Verification interfaces validate the input $L = (\mathsf{pk}_1, \dots, \mathsf{pk}_n)$ by retrieving a key pk'_i for $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{vReg}}$ for each $i \in [n]$ and checking $\mathsf{pk}_i = \mathsf{pk}'_i$. If validation fails, the protocol returns $(\textsc{Fail}, \mathsf{sid})$.

Key Registration Upon $(\textsc{Register}, \mathsf{sid})$ a signer $\mathcal{P}_i \in P$ proceeds as follows:

1. If there exists a record $(\textsc{keyrec}, *, *)$, then $(\textsc{fail}, \mathsf{sid})$.
2. Send $(\textsc{Register}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{vReg}}$.
3. Upon receiving $(\textsc{Registered}, \mathsf{sid}, \mathsf{pk}_i, \mathsf{sk}_i)$, create record $(\textsc{keyrec}, \mathsf{sk}_i, \mathsf{pk}_i)$ and output $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk}_i)$.

Signature Setup (Offline Phase) Same as π_{IMS}.

Signature Generation (Online Phase) Upon receiving $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L = (\mathsf{pk}_j)_{j \in I}, m, \mathsf{offs} = (\mathsf{off}_j)_{j \in [n]}, \sigma_{i-1})$, a signer $\mathcal{P}_i \in P$ proceeds as follows:

1. If there exists **no** record $(\textsc{presig}, \mathsf{ssid}, *, \mathsf{off}_i)$, return $(\textsc{fail}, \mathsf{sid})$.
2. If there already exists record $(\textsc{sig}, \mathsf{ssid}, *, *, *)$, return $(\textsc{fail}, \mathsf{sid})$.
3. Retrieve $(\textsc{presig}, \mathsf{ssid}, \mathtt{st}_i, \mathsf{off}_i)$ and run $(\sigma_i, b_{i-1}) \leftarrow \mathsf{SignOn}(\mathtt{st}_i, \mathsf{sk}_i, L, m, \mathsf{offs}, \sigma_{i-1})$.
4. Output $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_i, b_{i-1})$.

Signature Verification Upon receiving input $(\textsc{Verify}, \mathsf{sid}, L, m, \sigma)$, $\mathcal{V}$ runs $b \leftarrow \mathsf{Vrfy}(L, m, \sigma)$ and outputs $(\textsc{Verified}, \mathsf{sid}, b)$.

5 OMuSig2: Two-Round OMS based on Schnorr

In this section, we present a modified version of MuSig2 that realizes an Ordered Multi-Signature. The protocol is depicted in Figure 14.

We assume that the list of co-signers is known in advance and therefore construct an algorithm in the offline-online model. Following the route given by L in a round-Robin fashion, the signer in position i will only send messages to the signer in position $i+1$. The OMS setting makes the algorithms simpler and more efficient than MuSig2. In particular, each signer can compute its *so-far accumulated public key* $\widetilde{\mathsf{pk}} = \prod_{k=1}^{i-1} \mathsf{pk}_k$, and reuse it for later OMS signing instances, making runtime in the online phase independent of n, the total number of co-signers, and i, the signer's position in L.

Following the security model defined in Section 4, we assume that an adversary only outputs a forgery with a key list L^* consisting of pre-registered public keys. That is, the reduction knows secret keys associated

Construction 14: Ordered Multi-Signature from the (Algebraic) One-More Discrete Log Assumption

```
$\mathsf{Setup}(1^\lambda)$ :
1: $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
2: $(n, \ell) \leftarrow \mathsf{poly}(\lambda)$
3: $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{ch}} : \{0,1\}^* \to \mathbb{Z}_p^*$
4: $\sigma_0 := (1_{\mathbb{G}}, 0) \in \mathbb{G} \times \mathbb{Z}_p$
5: return $\mathsf{pp} := (\mathbb{G}, g, p, n, \ell, \mathsf{H}, \sigma_0)$

$\mathsf{KeyGen}()$ :
1: $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$; $\mathsf{pk} := g^{\mathsf{sk}}$
2: return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Vrfy}(L, m, \sigma)$ :
1: Parse $L = (\mathsf{pk}_1, \dots, \mathsf{pk}_n) \in \mathbb{G}^n$
2: for $i, j \in [n]$ do
3:   if $\mathsf{pk}_i = \mathsf{pk}_j \wedge i \neq j$ then return 0
4: Parse $\sigma = (R, z) \in \mathbb{G} \times \mathbb{Z}_p$
5: $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, L)$
6: if $g^z = R \cdot (\prod_{i=1}^n \mathsf{pk}_i)^c$ then return 1
7: return 0
```

Two-phase interactive signing run in a loop determined by $L = (\mathsf{pk}_1, \dots, \mathsf{pk}_n)$. Note that the first part of SignOn (lines 1-7) can be preprocessed given all offline tokens from the co-signers and independently of the message.

```
For $i \in [n]$ do
$\mathsf{SignOff}(\mathsf{sk}_i)$ :
1: for $j \in [\ell]$ do
2:   $r_{i,j} \xleftarrow{\$} \mathbb{Z}_p$
3:   $R_{i,j} := g^{r_{i,j}}$
4:   $\mathtt{st}_i := \mathtt{st}_i | r_{i,j} | R_{i,j}$
5: $\mathsf{off}_i := (R_{i,1}, \dots, R_{i,\ell})$
6: return $(\mathsf{off}_i, \mathtt{st}_i)$

$\mathsf{offs} := (\mathsf{off}_1, \dots, \mathsf{off}_n)$

For $i \in [n]$ do
$\mathsf{SignOn}(\mathtt{st}_i, \mathsf{sk}_i, m, L, \mathsf{offs}, \sigma_{i-1})$ :
            ▷ Lines 1-7: processing independent of $m, \sigma_{i-1}$
1: Parse $\mathtt{st}_i = (r_{i,j} | R'_{i,j})_{j=1}^{\ell} \in (\mathbb{Z}_p \times \mathbb{G})^{\ell}$
2: Parse $\mathsf{offs} = ((R_{1,j})_{j=1}^{\ell}, \dots, (R_{n,j})_{j=1}^{\ell}) \in \mathbb{G}^{\ell \times n}$
3: $\widetilde{\mathsf{pk}} := \prod_{k=1}^{i-1} \mathsf{pk}_k$
4: for $j \in [\ell]$ do
5:   $R_j := \prod_{k=1}^{n} R_{k,j}$
6:   $\tilde{R}_j := \prod_{k=1}^{i-1} R_{k,j}$          ▷ So-far aggregation
7:   if $R_j = 1_{\mathbb{G}}$ or $\tilde{R}_j = 1_{\mathbb{G}}$ then return $(\varepsilon, 0)$
            ▷ Lines 8-19: processing that depends on $m, \sigma_{i-1}$
8: Parse $\sigma_{i-1} = (R, \tilde{z}) \in \mathbb{G} \times \mathbb{Z}_p$
9: $v \leftarrow \mathsf{H}_{\mathsf{non}}(m, \mathsf{offs}, L)$
10: if $i = 1$ then
11:   $R := \prod_{j=1}^{\ell} R_j^{v^{j-1}}$
12:   $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, L)$
13: else                                  ▷ Check so-far aggregation
14:   if $R \neq \prod_{j=1}^{\ell} R_j^{v^{j-1}}$ then return $(\varepsilon, 0)$
15:   $\tilde{R} := \prod_{j=1}^{\ell} \tilde{R}_j^{v^{j-1}}$
16:   $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, L)$
17:   if $g^{\tilde{z}} \neq \tilde{R} \cdot \widetilde{\mathsf{pk}}^c$ then return $(\varepsilon, 0)$
18: $z_i := c \cdot \mathsf{sk}_i + \sum_{j=1}^{\ell} v^{j-1} \cdot r_{i,j}$
19: $z := \tilde{z} + z_i$
20: $\sigma_i = (R, z)$
21: return $(\sigma_i, 1)$
```

with all public keys in L^* except for the challenge key pk^*. In the proof we will make use of the General Forking Lemma (Lemma 2) and the One-More Discrete Log (OMDL) assumption (Definition 3). While this proof strategy closely follows that of MuSig2, guaranteeing correct order of honest signers turns out to be non-trivial. In fact, it is easy to see that the original MuSig2 doesn't qualify as a secure OMS, since an adversary can easily swap the order of honest parties' contribution by querying signing oracles in different order than what's specified in L. To overcome this hurdle, our proof makes use of the so-far aggregation check (L.17). In Appendix C, we also sketch how to strengthen this basic construction to meet security under the PPK model using existing tricks of MuSig2.

Theorem 15. *The* OMS *scheme of Construction 14 with* $\ell = 2$ *is* $\mathsf{OMS\text{-}UF\text{-}CMA}$ *secure (Fig.10) under the OMDL assumption and in the programmable random oracle model. Concretely, for any adversary* $\mathcal{A}$ *against* $\mathsf{OMS\text{-}UF\text{-}CMA}$ *security that receives at most* Q_k *honest public keys from the key generation oracle, makes at most* Q_h *queries to the random oracles, and starts at most* Q_s *sessions in total with* $\mathsf{OSignOff}, \mathsf{OSignOn}$ *oracles, there exists an adversary* $\mathcal{C}$ *against the* OMDL *assumption such that*

$$\mathbf{Adv}^{\mathsf{OMS\text{-}UF\text{-}CMA}}_{\mathsf{OMS}}(\mathcal{A}) \leq \frac{Q(Q+Q_k^2)}{p} + Q_k^2 \cdot \sqrt{Q \cdot \left(\mathbf{Adv}^{\mathsf{OMDL}}_{\mathsf{GGen}}(\mathcal{C}) + \frac{Q^2}{p}\right)}$$

where $Q = Q_s + 2Q_h + 1$.

Here we first provide a high level overview. The complete proof is deferred to Appendix B. The reduction has the same structure (except additional guessing arguments explained below) as the one used in the proof of MuSig2 [NRS21], with three wrappers around $\mathcal{A}$. The innermost wrapper, $\mathcal{B}$, simulates the OMS security game to $\mathcal{A}$ by embedding a Dlog challenge in one of the honest keys registered by $\mathcal{A}$ and by answering signing queries using queries to the Dlog solver oracle through the outer wrappers. Similar to the proof for MuSig2, $\mathcal{B}$ only queries the Dlog solver with a known linear combination of Dlog challenges. Therefore, one may in fact adjust the proof to prove security under the *algebraic* OMDL assumption, a weaker and falsifiable variant of OMDL [NRS21]. Next there is an algorithm Fork of the forking lemma [BN06, NRS21], that passes all Dlog challenges to $\mathcal{B}$ as well as inputs to be used to answer some of $\mathcal{A}$'s oracle queries. The main task of Fork is to rewind $\mathcal{B}$ (and thus $\mathcal{A}$, see Fig. 3) and carefully match inputs to $\mathcal{B}$ with the series of OMDL values obtained by Fork via the final wrapper $\mathcal{C}$, which is playing the OMDL game (see Def. 3).

Our proof differs from MuSig2 essentially in two ways due to a different security model. Since the OMS scheme additionally guarantees a correct sequence of honest signers unlike usual multi-signatures, we consider a more complex situation where multiple signing oracles co-exist instead of one. The reduction embeds a DLog challenge in one of the honest signing oracles and answers queries to the remaining oracles using self-generated keys. In this way, the reduction still succeeds using standard guessing arguments. While MuSig2 incurs a quartic security loss due to double-forking, our proof does not suffer from it since the security notion of OMS we inherit from [BGOY07] is not in the plain-public key model. Depending on the application context, one may want to guarantee the security without the key registration requirement, while withstanding the so-called *rogue-key attacks* [MOR01]. To retain security in the plain-public key model, the present construction should be modified by having each party use distinct Fiat-Shamir challenges instead of a single $c$as in Fig. 16.

Acknowledgment

We thank Jan Bobolz for helpful comments on an earlier version of this manuscript and discussions on the correctness definitions, which helped us fix technical issues in the UC treatments of interactive multi-signatures. We thank Jack Doerner for kindly clarifying the details of [CDL$^+$24]. The work described in this paper has received funding from: the Protocol Labs Research Grant Program PL-RGP1-2021-064, the Protocol Labs-CryptoSat SpaceVDF program, the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B, the research grant VIL53029 from Villum Fonden, and Vetenskapsrådet (VR) starting grant 2022-04684. This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

AB21. Handan Kilinç Alper and Jeffrey Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, August 2021. Springer, Cham.

ADN06. Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Berlin, Heidelberg, May / June 2006.

AF04. Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Berlin, Heidelberg, August 2004.

AFY23. E. Aleksandrova, A. Fedichev, and A. Yarmak. Lattice-based ordered multisignature for industrial iot. In Andrey A. Radionov and Vadim R. Gasiyarov, editors, *Advances in Automation IV*, pages 363–373, Cham, 2023. Springer International Publishing.

BCH$^+$20. Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 1–30. Springer, Cham, November 2020.

BCJ08. Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008.

BCNP04. Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195. IEEE Computer Society Press, October 2004.

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Cham, December 2018.

BDPT24. Carsten Baum, Bernardo Machado David, Elena Pagnin, and Akira Takahashi. CaSCaDE: (time-based) cryptography from space communications DElay. In Clemente Galdi and Duong Hieu Phan, editors, *SCN 24, Part I*, volume 14973 of *LNCS*, pages 252–274. Springer, Cham, September 2024.

BGOY07. Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 276–285. ACM Press, October 2007.

BGR12. Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 644–662. Springer, Berlin, Heidelberg, December 2012.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

BMT18. Christian Badertscher, Ueli Maurer, and Björn Tackmann. On composable security for digital signatures. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 494–523. Springer, Cham, March 2018.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

BNN07. Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422. Springer, Berlin, Heidelberg, July 2007.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003.

BTT22. Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 276–305. Springer, Cham, August 2022.

Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Paper 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

Can03. Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003.

Can04a. Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.

Can04b. Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.

CDG$^+$18. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Cham, April / May 2018.

CDL$^+$24. Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. Cryptology ePrint Archive, Paper 2024/1807, 2024.

CDPW07. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Berlin, Heidelberg, February 2007.

CGG$^+$20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni

Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

CH06. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 380–403. Springer, Berlin, Heidelberg, March 2006.

CK21. Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. Cryptology ePrint Archive, Report 2021/916, 2021.

CR03. Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Berlin, Heidelberg, August 2003.

CZ22. Yanbo Chen and Yunlei Zhao. Half-aggregation of schnorr signatures with tight reductions. Cryptology ePrint Archive, Report 2022/222, 2022.

DEF$^+$19. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1084–1101. IEEE, 2019.

DKLs18. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

FLS12. Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 113–130. Springer, Berlin, Heidelberg, September 2012.

GOR18. Craig Gentry, Adam O'Neill, and Leonid Reyzin. A unified framework for trapdoor-permutation-based sequential aggregate signatures. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 34–57. Springer, Cham, March 2018.

Hou10. Wei-hua Hou. An ordered multisignature without random oracles. In *2010 International Conference on Communications and Mobile Computing*, volume 1, pages 21–25. IEEE, 2010.

IN83. K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. NEC Res. Development 71, 1983.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.

KOR23. Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 646–677. Springer, Cham, August 2023.

Lin17. Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Cham, August 2017.

Lin24. Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *CiC*, 1(1):25, 2024.

LMRS04. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Berlin, Heidelberg, May 2004.

LOS$^+$06. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Berlin, Heidelberg, May / June 2006.

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001.

Nev08. Gregory Neven. Efficient sequential aggregate signed data. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 52–69. Springer, Berlin, Heidelberg, April 2008.

NRS21. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Cham.

RY07. Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Berlin, Heidelberg, May 2007.

YMO15. Naoto Yanai, Masahiro Mambo, and Eiji Okamoto. An ordered multisignature scheme under the cdh assumption without random oracles. In *Information Security: 16th International Conference, ISC 2013, Dallas, Texas, November 13-15, 2013, Proceedings*, pages 367–377. Springer, 2015.

A UC Treatment of OMS

A.1 Proof of Theorem 13

Proof. In Figure 15 we describe a PPT simulator $\mathcal{S}_{\mathsf{OMS}}$ that has black-box access to $\mathcal{A}$, and simulates π_{OMS} in a way that is indistinguishable to any efficient environment $\mathcal{Z}$. The OMS scheme used in the proof follows the syntax given in Definition 9. The proof closely follows that of Theorem 7 for an unordered interactive multi-signature scheme. Let us go over each interface.

- KeyGen: The only difference is when $\mathcal{F}_{\mathsf{OMS}}$ aborts after receiving pk_i from $\mathcal{S}_{\mathsf{OMS}}$. This event occurs with negligible probability assuming sufficiently high min-entropy of honestly generated pk.
- SignOff: Since $\mathcal{S}_{\mathsf{OMS}}$ runs $\mathsf{SignOff}$ as in π_{OMS}, there is no difference.
- SignOn: $\mathcal{S}_{\mathsf{OMS}}$ runs SignOn. There are three events that introduce a potential discrepancy in the view of $\mathcal{Z}$: 1) all signers are honest while $\mathcal{F}_{\mathsf{OMS}}$ aborts on $i = n$ after detecting an existing record

Simulator 15: $\mathcal{S}_{\mathsf{OMS}}$

$\mathcal{S}_{\mathsf{OMS}}$ interacts with an internal copy of $\mathcal{A}$, towards which it simulates the honest parties in $P = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and relays all the random oracle calls requests from $\mathsf{SignOff}$, SignOn, and Vrfy to the external global random oracle functionality $\mathcal{G}_{\mathsf{RO}}$. Whenever $\mathcal{A}$ queries $\mathcal{F}_{\mathsf{vReg}}$ with input $(\mathsf{pk}, \mathsf{sk}, \rho)$ on behalf of a corrupt party $\mathcal{P}_i \in C$, $\mathcal{S}_{\mathsf{OMS}}$ executes the code of $\mathcal{F}_{\mathsf{vReg}}$ for **Adversarial Key Registration** to simulate its response and then queries $\mathcal{F}_{\mathsf{OMS}}$ with input $(\textsc{Register}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pk}, \mathsf{sk}, \rho)$, so that pk gets registered for $\mathcal{P}_i$.

Key Registration: On receiving $(\textsc{Register}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{F}_{\mathsf{OMS}}$, run KeyGen to obtain the key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. Return $(\textsc{KeyConf}, \mathsf{sid}, \mathsf{pk}_i)$ to $\mathcal{F}_{\mathsf{OMS}}$.
Sign Setup: Same as $\mathcal{S}_{\mathsf{IMS}}$.
Sign Gen: On receiving $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}_i, L, m, \sigma_{i-1})$ from $\mathcal{F}_{\mathsf{OMS}}$, retrieve the corresponding stored $\mathtt{st}_i$ and offs, then run $(\sigma_i, b_{i-1}) \leftarrow \mathsf{SignOn}(\mathtt{st}_i, \mathsf{sk}_i, L, m, \mathsf{offs}, \sigma_{i-1})$, then return $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \sigma_i)$ to $\mathcal{F}_{\mathsf{OMS}}$.
Verify: Same as $\mathcal{S}_{\mathsf{IMS}}$.

$(\textsc{msig}, L, m, \sigma_n)$. 2) $\mathcal{F}_{\mathsf{OMS}}$ outputs $f_{i-1} = 1$ after the honest $i-1$ signers generated signatures $\sigma_1, \ldots, \sigma_{i-1}$ on the same input (L, m, offs) in legitimate order, even if $b_{i-1} = 0$ (i.e., the aggregate-so-far σ_{i-1} is invalid w.r.t. L, m, offs). 3) $\mathcal{F}_{\mathsf{OMS}}$ outputs $f_{i-1} = 0$ after detecting **no** record $(\textsc{sig}, *, \mathcal{P}_j, L, m, *, *, *)$ for some $1 \leq j < i \leq n$ with $\mathcal{P}_j \in H$, even if $b_{i-1} = 1$ (i.e., the aggregate-so-far σ_{i-1} is valid w.r.t. L, m, offs). Case 1) occurs with negligible probability assuming correctness of honestly generated $\sigma_1, \ldots, \sigma_n$ and unforgeability. Case 2) introduces a discrepancy since π_{OMS} may end up outputting $b_{i-1} = 0$. However, if $\mathcal{Z}$ causes an honest execution of SignOn to output $b_{i-1} = 0$ on honestly generated inputs, this implies $\mathcal{Z}$ sets the flag win to 1 in the OMS correctness game. That is, using such $\mathcal{Z}$ as a subroutine, one can construct a reduction $\mathcal{B}$ breaking correctness. Case 3) introduces a discrepancy since π_{OMS} may end up outputting $b_{i-1} = 0$. However, if $\mathcal{Z}$ manages to come up with an input $(L, m, \mathsf{offs}, \sigma_{i-1})$ that causes the honest execution of SignOn to output $b_{i-1} = 1$, this implies $\mathcal{Z}$ created a forged so-far-aggregated signature that sets the flag win to 1 in the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game (i.e. detection of bad order). That is, using such $\mathcal{Z}$ as a subroutine, one can construct a reduction $\mathcal{B}$ breaking $\mathsf{OMS\text{-}UF\text{-}CMA}$. This is done in a manner analogous to the last case of $\textsc{Verify}$.

– $\textsc{Verify}$: We consider the following cases:

- If there is **no** corruption **and** for the same ssid and offs, all $\mathcal{P}_i \in P$ sequentially completed SignOn with input (L, m) with partial signatures $(\sigma_1, \ldots, \sigma_n)$: In this case, there is a potential discrepancy in the view of $\mathcal{Z}$ since $\mathcal{F}_{\mathsf{OMS}}$ always outputs a decision bit 1 while π_{OMS} returns the output of $\mathsf{Vrfy}(L, m, \sigma)$. However, if $\mathcal{Z}$ outputs (L, m, σ) causing Vrfy to return 0, this implies $\mathcal{Z}$ breaks the correctness of OMS after adaptively querying the oracles for key registration, offline signing, and online signing. This event thus happens with negligible probability.

- If for input (L, m) all honest parties $\mathcal{P}_i \notin C$ complete $\textsc{SignOn}$ with partial signature σ_i: In this case, $\mathcal{F}_{\mathsf{OMS}}$ asks $\mathcal{S}_{\mathsf{OMS}}$ to simulate a decision bit. Since $\mathcal{S}_{\mathsf{OMS}}$ also runs Vrfy as in π_{OMS}, there is no difference in the view of $\mathcal{Z}$.

- If for input (L, m) some honest party $\mathcal{P}_i \notin C$ has **not** completed $\textsc{SignOn}$: In this case, there is a potential discrepancy in the view of $\mathcal{Z}$ since $\mathcal{F}_{\mathsf{OMS}}$ always outputs a decision bit 0 while π_{OMS} returns the output of $\mathsf{Vrfy}(L, m, \sigma)$. However, if $\mathcal{Z}$ manages to come up with (L, m, σ) causing Vrfy to return 1, this implies $\mathcal{Z}$ created a valid forgery in the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game. That is, using such $\mathcal{Z}$ as a subroutine one can construct a reduction $\mathcal{B}$ breaking $\mathsf{OMS\text{-}UF\text{-}CMA}$. On receiving public parameters pp, $\mathcal{B}$ with access to $\mathsf{OKeyReg}, \mathsf{OSignOff}, \mathsf{OSignOn}$ and the random oracle H, plays the role of $\mathcal{F}_{\mathsf{OMS}}$, $\mathcal{S}_{\mathsf{OMS}}$, and $\mathcal{G}_{\mathsf{RO}}$ and proceeds as follows.

  * Upon receiving a key generation request for $\mathcal{P}_i$: If $\mathcal{P}_i$ is honest, then query $\mathsf{OKeyReg}$ with $\mathsf{uid} = i$ and $\mathsf{aux} = \varepsilon$ to retrieve a key pk_i. If $\mathcal{P}_i$ is corrupted and queries $\mathcal{F}_{\mathsf{vReg}}$ with ($\textsc{register}$, $\mathsf{sid}, \mathsf{pk}_i, \mathsf{sk}_i, \rho_i$), then query $\mathsf{OKeyReg}$ with $\mathsf{uid} = i$ and $\mathsf{aux} = (\mathsf{pk}_i, \mathsf{sk}_i, \rho_i)$ to register a corrupted key.
  * Whenever $\mathcal{Z}$ makes a query to $\mathcal{G}_{\mathsf{RO}}$, relay queries and responses to and responses from H.
  * Whenever $\mathcal{S}_{\mathsf{OMS}}$ is prompt with a $\textsc{SignOff}$ command for party $\mathcal{P}_i \notin C$, query $\mathsf{OSignOff}$ with ssid and $\mathsf{uid} = i$ to receive off_i and forwards it to $\mathcal{F}_{\mathsf{OMS}}$. Otherwise, $\textsc{SignOff}$ command is handled as in $\mathcal{S}_{\mathsf{OMS}}$.
  * Whenever $\mathcal{S}_{\mathsf{OMS}}$ is asked to run $\textsc{SignOn}$ command for party $\mathcal{P}_i \notin C$, query $\mathsf{OSignOn}$ with input $(\mathsf{ssid}, \mathsf{uid} = 1, L, m, \mathsf{offs}, \sigma_{i-1})$ to receive (σ_i, b_{i-1}). Otherwise, $\textsc{SignOn}$ command is handled as in $\mathcal{S}_{\mathsf{OMS}}$.

* If $\mathcal{Z}$ outputs (L, m, σ) causing $\mathsf{Vrfy}(L, m, \sigma) = 1$ while $\exists \mathsf{pk}_{i^*} \in L$ such that $\mathcal{P}_{i^*} \notin C$ and (L, m) has never been queried to $\mathsf{OSignOn}$, forward this tuple to the $\mathsf{IMS\text{-}UF\text{-}CMA}$ game.

In this way, the reduction $\mathcal{B}$ succeeds in winning the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game. Hence, the advantage of $\mathcal{B}$ is non-negligible if $\mathcal{Z}$ finds inconsistency of verification in real and ideal executions with non-negligible probability.

□

A.2 Proof of Theorem 14

We now prove the opposite direction to Section 4.4 and show that a UC-secure OMS implies a construction that is secure according to the game-based definition. Towards this, we overload the following algorithms with oracle access to a UC experiment which either interacts with π_{OMS} or $\mathcal{F}_{\mathsf{OMS}}$:

$\mathsf{OKeyReg}(\mathsf{uid}, \mathsf{aux})$**:** it follows Game 10 except the following differences. If $\mathsf{aux} = \epsilon$ then it retrieves pk by querying $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ with $(\textsc{Register}, \mathsf{sid})$; else, it parses $\mathsf{aux} = (\mathsf{pk}, \mathsf{sk}, \rho)$ and makes a query to $\mathcal{F}_{\mathsf{vReg}}$ with $(\textsc{Register}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, \rho)$ on behalf of arbitrary corrupt parties whose key is not registered yet.

$\mathsf{OSignOff}(\mathsf{ssid}, \mathsf{uid})$**:** it follows Game 10 except that it retrieves off by querying $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ with input $(\textsc{SignOff}, \mathsf{sid}, \mathsf{ssid})$ on behalf of a party associated with uid.

$\mathsf{OSignOn}(\mathsf{ssid}, \mathsf{uid}, L, m, \mathsf{offs}, \sigma, i)$**:** it follows Game 10 except that (σ_i, b_{i-1}) is obtained by querying $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ with input $(\textsc{SignOn}, \mathsf{sid}, \mathsf{ssid}, L, m, \mathsf{offs}, \sigma_{i-1})$ on behalf of $\mathcal{P}_i$.

As in Section 3.5, the key generation algorithm only outputs an empty string $\perp$ as secret signing key, since such a key is not defined in $\mathcal{F}_{\mathsf{OMS}}$.

Proof. Correctness can be ensured by inspection of the algorithm and the functionality. That is, if OMS is not correct, there exists an adversary $\mathcal{A}$ that causes either SignOn or Vrfy to output a decision bit 0 with non-negligible probability, even if its inputs are produced by honest parties (Definition 10). Using such $\mathcal{A}$, we can construct an environment that distinguishes whether it's interacting with $\mathcal{F}_{\mathsf{OMS}}$ or π_{OMS} as follows: the environment calls key registration upon $\mathcal{A}$ querying $\mathsf{OKeyReg}$, signature setup upon $\mathcal{A}$ querying $\mathsf{OSignOff}$, and signature generation upon $\mathcal{A}$ querying $\mathsf{OSignOn}$ for all honest parties, and then verifies the final σ with respect to the message and the public key set used earlier. If in the real world, the environment observes either b_{i-1} for some $i \in [n]$ or the return value of verification being 0 with non-negligible probability; if in the ideal world, those decision bits are always 1. Thus, this is a valid distinguisher with non-negligible advantage.

Toward proving unforgeability, consider that the oracles are already fully defined. Assume that there exists an attacker $\mathcal{A}$ which can win Game $\mathsf{OMS\text{-}UF\text{-}CMA}$ with non-negligible probability. This happens either in two cases: either the oracle $\mathsf{OSignOn}$ set $\mathtt{win} = 1$, or line 10 of the game $\mathsf{OMS\text{-}UF\text{-}CMA}$ returns 1. We then show the existence of an environment $\mathcal{Z}$ such that for any ideal adversary $\mathcal{S}$, $\mathcal{Z}$ can tell whether it is interacting with π_{OMS} or π_{OMS}. To this end, we construct $\mathcal{Z}$ that runs a copy of game-based adversary $\mathcal{A}$ and simulates the view of $\mathcal{A}$ in $\mathsf{OMS\text{-}UF\text{-}CMA}$ game. This is done by mapping return values of $\mathsf{OKeyReg}, \mathsf{OSignOff}, \mathsf{OSignOn}$ to responses from the corresponding interfaces in an UC experiment as above. Moreover, $\mathcal{Z}$ relays any random oracle queries made by $\mathcal{A}$ to $\mathcal{G}_{\mathsf{RO}}$ and returns $\mathcal{G}_{\mathsf{RO}}$'s response to $\mathcal{A}$.

First, consider the case where $\mathsf{OMS\text{-}UF\text{-}CMA}$ sets $\mathsf{win} = 1$ at Line 10. Upon receiving a forgery tuple (L^*, m^*, σ^*) from $\mathcal{A}$, the game simulated by $\mathcal{Z}$ would output 1 if $\mathsf{Vrfy}(L^*, m^*, \sigma^*) = 1$. This requires that $\mathsf{pk} \in L^*$. Moreover, **Signature Generation (Online Phase)** of $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ has never been called for L^*, m^* by the oracle $\mathsf{OSignOn}$ as the attacker would otherwise have lost the game. $\mathcal{Z}$ finally queries $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ with input $(\textsc{Verify}, \mathsf{sid}, L^*, m^*, \sigma^*)$ to obtain its result and outputs what it outputs. Here, if $\mathcal{Z}$ is interacting with the ideal process, $\mathcal{F}_{\mathsf{OMS}}$ never outputs $(\textsc{Verified}, \mathsf{sid}, 1)$ due to Step 3. However, if $\mathcal{Z}$ is interacting with the real process, π_{OMS} outputs $(\textsc{Verified}, \mathsf{sid}, 1)$ as long as $\mathcal{A}$ wins the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game.

Second, consider the case where $\mathsf{OMS\text{-}UF\text{-}CMA}$ sets $\mathsf{win} = 1$ at Line 13 of $\mathsf{OSignOn}$. Upon receiving a tuple $(\mathsf{uid}, L, m, \mathsf{offs}, \sigma_{i-1}, i)$ from $\mathcal{A}$, the game simulated by $\mathcal{Z}$ would output 1 if SignOn outputs $b_{i-1} = 1$ while any uncorrupt party with index $j < i$ has never signed (m, L). Moreover, **Signature Generation (Online Phase)** of $\pi_{\mathsf{OMS}}/\mathcal{F}_{\mathsf{OMS}}$ has never been called for L, m and for uncorrupt $\mathcal{P}_j$ by the oracle $\mathsf{OSignOn}$ as the attacker would otherwise have lost the game. Here, if $\mathcal{Z}$ is interacting with the ideal process, $\mathcal{F}_{\mathsf{OMS}}$ never outputs $b_{i-1} = 1$ due to Step 5 of **Signature Generation (Online Phase)**. However, if $\mathcal{Z}$ is interacting with the real process, π_{OMS} outputs $b_{i-1} = 0$ as long as $\mathcal{A}$ wins the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game. Therefore, $\mathcal{Z}$ has non-negligible advantage in the UC experiment.

□

B Proof of Theorem 15

Proof. We prove that any successful forger $\mathcal{A}$ can be used to break the OMDL assumption. We consider two types of forgeries: (1) $(m^*, \sigma^* = (R^*, z^*), L^*)$ output at the end of the game, and (2) $(m^*, L^*, \mathsf{offs}^*, \sigma^*_{i^*-1})$ sent to the $\mathsf{OSignOn}$ oracle with index i^* (and which triggers the $\mathsf{win} := 1$ flag).

Type-(1) Forgery If the adversary $\mathcal{A}$ causes a Type-(1) forgery, we construct the following reduction to OMDL. Let Q_k be the number of queries to $\mathsf{OKeyReg}$ with $\mathsf{aux} = \varepsilon$ (i.e., maximum number of honest keys). The reduction goes through multiple layers of wrappers below.

- $\mathcal{B}(\mathsf{pk}^*, U_1, \ldots, U_{2Q_s}, c_1, \ldots, c_Q, v_1, \ldots, v_Q)$: A wrapper algorithm that simulates the view of $\mathcal{A}$ without using a secret key belonging to one of the uncorrupted (honest) parties. $\mathcal{B}$ initially picks uniform $t \in [1, Q_k]$. Whenever $\mathcal{A}$ submits a new honest signer key request (uid, ϵ), if this is the t-th query, then it answers with the challenge public key pk^*. Otherwise, it generates a new key pair using KeyGen algorithm and returns the corresponding public key as $\mathsf{OKeyReg}$ would. In this way, $\mathcal{B}$ knows $Q_k - 1$ secret keys of honest parties. $\mathcal{B}$ answers queries to H_{ch}, $\mathsf{H}_{\mathsf{non}}$, $\mathsf{OSignOff}$ and $\mathsf{OSignOn}$ in the natural way (explained below). Upon receiving a valid forgery $(m^*, L^*, \sigma^* = (R^*, z^*))$ at the end of the game, it outputs $(\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, \mathsf{out})$, which are as described below.
- $\mathsf{Fork}(\mathsf{pk}^*, U_1, \ldots, U_{2Q_s}, v_1, \ldots, v_Q, \hat{v}_1, \ldots, \hat{v}_Q)$: A forker algorithm that rewinds $\mathcal{B}$ as in Figure 3.
- $\mathcal{C}$: A OMDL adversary again that runs Fork internally. It initially makes $2Q_s{+}1$ queries to $\mathcal{O}_{\mathrm{ch}}$ to obtain DLog instances $(\mathsf{pk}^*, U_1, \ldots, U_{2Q_s})$. After sampling $v_1, \hat{v}_1, \ldots, v_Q, \hat{v}_Q \in \mathbb{Z}_p$ uniformly, $\mathcal{C}$ runs $\mathsf{Fork}(\mathsf{inp}, v_1, \hat{v}_1, \ldots, v_Q, \hat{v}_Q)$ where $\mathsf{inp} = (\mathsf{pk}^*, U_1, \ldots, U_{2Q_s})$. Then on receiving $(1, \mathsf{out}, \hat{\mathsf{out}})$ from Fork, it outputs DLog solutions $(\mathsf{sk}^*, u_1, \ldots, u_{2Q_s})$ such that $\mathsf{pk}^* = g^{\mathsf{sk}^*}$ and $U_j = g^{u_j}$ for $j = 1, \ldots, 2Q_s$. Note that $\mathcal{C}$ is allowed to make at most $2Q_s$ queries to $\mathcal{O}_{\mathrm{dl}}$ in order to win the OMDL game.

Description of $\mathcal{B}$ $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game as follows, except at a few abort events highlighted.

- Initialization: $\mathcal{B}$ runs $\mathcal{A}$ on input pp. It initially samples uniform $t \in [1, Q_k]$ and sets $\mathsf{ctr\_u} = 0$, $\mathsf{ctr\_ch} = 0$ and $\mathsf{ctr\_non} = 0$. It also initializes empty key-value lookup tables $\mathrm{HT}_{\mathsf{ch}}$ and $\mathrm{HT}_{\mathsf{non}}$.

– $\mathsf{H}_{\mathsf{non}}$: On receiving a query to $\mathsf{H}_{\mathsf{non}}$ with input $X := (m, (R_{1,1}, R_{1,2}, \ldots, R_{n,1}, R_{n,2}), L)$, if $\mathrm{HT}_{\mathsf{non}}[X]$ is already defined $\mathcal{B}$ returns this value. Otherwise, $\mathcal{B}$ increments $\mathsf{ctr\_non}$, and computes $R = \prod_{i=1}^{n} R_{i,1} \cdot (\prod_{i=1}^{n} R_{i,2})^{v_{\mathsf{ctr\_non}}}$, using the v_i values in its input. *If* $\prod_{i=1}^{n} R_{i,2} \neq 1_{\mathbb{G}}$ *and* $\mathrm{HT}_{\mathsf{ch}}[m, R, L]$ *is already defined*[7]*, $\mathcal{B}$ aborts. Assuming $v_{\mathsf{ctr\_non}}$ is sampled uniformly from $\mathbb{Z}_p$, this happens with probability at most $1/p$ for each query and thus overall at most Q^2/p by the union bound.* Otherwise, it makes a query to H_{ch} with input (m, R, L), sets $\mathrm{HT}_{\mathsf{non}}[X] := v_{\mathsf{ctr\_non}}$, and returns $v_{\mathsf{ctr\_non}}$. Here we use the programmability of the random oracle.

– H_{ch}: On receiving a query to H_{ch} with input $X := (m, R, L)$, if $\mathrm{HT}_{\mathsf{ch}}[X]$ is defined $\mathcal{B}$ returns this value. Otherwise, $\mathcal{B}$ increments $\mathsf{ctr\_ch}$, sets $\mathrm{HT}_{\mathsf{ch}}[X] := c_{\mathsf{ctr\_ch}}$, and returns $c_{\mathsf{ctr\_ch}}$. Here we use the programmability of the random oracle.

– $\mathsf{OKeyReg}$: Whenever $\mathcal{A}$ queries $\mathsf{OKeyReg}$ with new uid with empty aux, if this is the t-th query with empty aux, $\mathcal{B}$ lets $\mathsf{uid}^* = \mathsf{uid}$, registers $\mathcal{K}[\mathsf{uid}^*] = (1, \mathsf{pk}^*, 0)$, and returns pk^*. Otherwise, it proceeds as the actual $\mathsf{OKeyReg}$ would.

– $\mathsf{OSignOff}$: Whenever $\mathcal{A}$ queries $\mathsf{OSignOff}$ with a new session identifier ssid and user ID uid corresponding to one of the honest keys, if $\mathsf{uid} = \mathsf{uid}^*$ then $\mathcal{B}$ lets $R_{i,j} := U_{\mathsf{ctr\_u}+j}$ for $j = 1, 2$, and then sets $\mathsf{ctr\_u} = \mathsf{ctr\_u} + 2$. Otherwise, it proceeds as the actual $\mathsf{OSignOff}$ would. $\mathcal{B}$ also updates $\mathcal{T}$ as $\mathsf{OSignOff}$ would and returns $\mathsf{off}_i = (R_{i,1}, R_{i,2})$.

– $\mathsf{OSignOn}$: Whenever $\mathcal{A}$ queries $\mathsf{OSignOn}$ with a valid ssid, uid, m, $L = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, $\mathsf{offs} = (R_{1,1}, R_{1,2}, \ldots, R_{n,1}, R_{n,2})$, $\sigma_{i-1} = (R, \tilde{z})$ with an index i corresponding to one of the honest keys, $\mathcal{B}$ runs all the sanity checks performed by $\mathsf{OSignOn}$, and executes the operations of SignOn, except at line 18 where $\mathcal{B}$'s behavior differs depending on the type of the key:
  - If $\mathsf{uid} = \mathsf{uid}^*$ and $\mathsf{pk}_i = \mathsf{pk}^*$, $\mathcal{B}$ makes a query to $\mathcal{O}_{\mathrm{dl}}$ with input $R_{i,1} \cdot R_{i,2}^{v} \cdot \mathsf{pk}_i^{c}$ to obtain its discrete logarithm: z_i.
  - Else, $\mathcal{B}$ computes z_i using the knowledge of sk_i and $r_{i,1}, r_{i,2}$ as the actual SignOn would.

  Whenever any of the checks performed by $\mathsf{OSignOn}$ fails, $\mathcal{B}$ also returns $\perp$ to $\mathcal{A}$ as $\mathsf{OSignOn}$ would.

– When $\mathcal{A}$ outputs a forgery tuple $(m^*, L^*, \sigma^* = (R^*, z^*))$, $\mathcal{B}$ returns $(0, \perp, \perp)$ if any of the validity checks in the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game fails.

[7] Note that this is exactly why SignOn of Fig. 14 aborts in Line 7, because otherwise this abort event trivially happens with non-negligible probability.

Moreover, if $\mathsf{pk}^* \notin L^*$ *or* (m^*, L^*) *has been signed by* pk^* *previously,* $\mathcal{B}$ *aborts. Since the challenge* pk^* *is embedded in one of* Q_k *honest keys uniformly at random, the probability that* $\mathcal{B}$ *does* ***not*** *abort here is at least* $1/Q_k$. Otherwise, let $\mathsf{ctr\_ch}^*$ and $\mathsf{ctr\_non}^*$ be the values of $\mathsf{ctr\_ch}$ and $\mathsf{ctr\_non}$ at the moment $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*]$ is defined, respectively. That is, $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*] = c_{\mathsf{ctr\_ch}^*}$ and $v_1, \ldots, v_{\mathsf{ctr\_non}^*}$ have been used by the time $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*]$ is set. Moreover let $j^* \in [n]$ be the index such that $\mathsf{pk}^* = \mathsf{pk}_{j^*} \in L$ and K^* be the list of secret keys corresponding to other public keys $L^* \setminus \{\mathsf{pk}_{j^*}\}$. Finally, $\mathcal{B}$ outputs $(\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, \mathsf{out} = (\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, j^*, L^*, K^*, R^*, c^*, z^*)$.

Description of $\mathcal{C}$

- $\mathcal{C}$ runs Fork on its input and uniformly sampled $v_1, \ldots, v_Q, \hat{v}_1, \ldots, \hat{v}_Q \in \mathbb{Z}_q$. Whenever $\mathcal{B}$ asks for a query to $\mathcal{O}_{\mathrm{dl}}$ through Fork, $\mathcal{C}$ forwards that query and the response accordingly, but in case the second run of $\mathcal{B}$ makes a query identical to one of the previous queries, $\mathcal{C}$ uses a cached response from the previous run instead of redundantly querying $\mathcal{O}_{\mathrm{dl}}$. After obtaining $(b, \mathsf{out}, \hat{\mathsf{out}})$ from Fork, if $b = 0$ $\mathcal{C}$ aborts. If $b = 1$, then $\mathcal{C}$ parses
$$\mathsf{out} = (\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, (j^*, L^*, K^*, R^*, c^*, z^*))$$
and
$$\hat{\mathsf{out}} = (\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, (j^*, L^*, K^*, R^*, \hat{c}^*, \hat{z}^*)),$$
from Fork (where $\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, j^*, L^*, K^*, R^*$ from two outputs are guaranteed to be identical thanks to the way $\mathcal{B}$ simulates the view and the successful run of Fork). Due to the verification condition, if $b = 1$ we have that
$$g^{z^*} = R^* \cdot \left(\prod_{i=1}^{n} \mathsf{pk}_i\right)^{c_{\mathsf{ctr\_ch}^*}}$$
$$g^{\hat{z}^*} = R^* \cdot \left(\prod_{i=1}^{n} \mathsf{pk}_i\right)^{\hat{c}_{\mathsf{ctr\_ch}^*}}.$$
- $\mathcal{C}$ extracts the secret key for the challenge public key pk^* as follows.
$$\mathsf{sk}^* = \left((z^* - \hat{z}^*) \cdot (c_{\mathsf{ctr\_ch}^*} - \hat{c}_{\mathsf{ctr\_ch}^*})^{-1} - \sum_{i=1, i \neq j^*}^{n} \mathsf{sk}_i\right)$$
where co-signers' secret keys sk_i for $i \neq j^*$ are indeed known thanks to the key registration requirement.

– Now that $\mathcal{C}$ knows DLog of pk, what's left is computing DLogs of $(U_1, \ldots, U_{2Q_s})$. For each $\mathsf{ctr\_u} \in [0, Q_s - 2]$, consider the response z_i computed inside $\mathsf{OSignOn}$ by consuming $U_{\mathsf{ctr\_u}+1}, U_{\mathsf{ctr\_u}+2}$ and querying $\mathcal{O}_{\mathrm{dl}}$ during the first run of $\mathcal{B}$. For some $\mathsf{ctr\_ch}$, $\mathsf{ctr\_non}$, z_i satisfies

$$z_i = r_{i,1} + v_{\mathsf{ctr\_non}} \cdot r_{i,2} + c_{\mathsf{ctr\_ch}} \cdot \mathsf{sk}^* \tag{1}$$

where $r_{i,1}$ and $r_{i,2}$ are DLogs of $R_{i,1} = U_{\mathsf{ctr\_u}+1}$ and $R_{i,2} = U_{\mathsf{ctr\_u}+2}$, and $v_{\mathsf{ctr\_non}} = \mathsf{H}_{\mathsf{non}}(m, \mathsf{offs}, L)$. If $\mathsf{ctr\_ch} < \mathsf{ctr\_ch}^*$ at that moment, then $\mathcal{C}$ hasn't made any extra query to $\mathcal{O}_{\mathrm{dl}}$ during the second run of $\mathcal{B}$. Thus, $\mathcal{C}$ simply queries $\mathcal{O}_{\mathrm{dl}}$ with $R_{i,1}$ to learn $r_{i,1}$ and uses this information to compute $r_{i,2} := dLog(R_{i,2}) = dLog(U_{\mathsf{ctr\_u}+2})$. If $\mathsf{ctr\_ch} > \mathsf{ctr\_ch}^*$ at that moment[8], the response $\hat{z}_i$ computed in the second run also satisfies

$$\hat{z}_i = r_{i,1} + \hat{v}_{\widehat{\mathsf{ctr\_non}}} \cdot r_{i,2} + \hat{c}_{\widehat{\mathsf{ctr\_ch}}} \cdot \mathsf{sk}^* \tag{2}$$

where $\widehat{\mathsf{ctr\_non}} > \mathsf{ctr\_non}^*$ and $\hat{v}_{\widehat{\mathsf{ctr\_non}}} = \mathsf{H}_{\mathsf{non}}(\hat{m}, \hat{\mathsf{offs}}, \hat{L})$. If $v_{\mathsf{ctr\_non}} = \hat{v}_{\widehat{\mathsf{ctr\_non}}}$ (which happens with probability at most $1/p$ for each combination of $\mathsf{ctr\_non}$ and $\widehat{\mathsf{ctr\_non}}$ and thus overall at most Q^2/p by the union bound), $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ can successfully solve the system of two linear equations with two unknowns $r_{i,1}$ and $r_{i,2}$ (recall that at this point $\mathsf{sk}^*, v_{\mathsf{ctr\_non}}, \hat{v}_{\widehat{\mathsf{ctr\_non}}}, c_{\mathsf{ctr\_ch}}$ and $\hat{c}_{\widehat{\mathsf{ctr\_ch}}}$ are all known to the reduction).

We now invoke the forking lemma (Lemma 2). By construction we have that

$$\mathsf{acc}(\mathcal{B}) = \left(\mathbf{Adv}^{\mathsf{OMS\text{-}UF\text{-}CMA}}_{\mathsf{OMS}}(\mathcal{A}) - \frac{Q^2}{p} \right) / Q_k$$

by accounting for the abort events happening when queries to $\mathsf{H}_{\mathsf{non}}$ are made and when the forgery is submitted by $\mathcal{A}$. Since $\mathcal{C}$ succeeds in breaking the OMDL assumption as long as Fork outputs two valid outputs *and* it doesn't abort, overall

$$\begin{aligned}\mathbf{Adv}^{\mathsf{OMDL}}(\mathcal{C}) &\geq \mathsf{frk} - Q^2/p \\ &\geq \mathsf{acc}(\mathcal{B}) \cdot \left(\frac{\mathsf{acc}(\mathcal{B})}{Q} - \frac{1}{p} \right) - \frac{Q^2}{p}.\end{aligned}$$

Finally, we remark that the above lower bound is > 0 and can easily be made noticeable since p's size is determined directly by the security parameter.

[8] It must be that $\mathsf{ctr\_ch} \neq \mathsf{ctr\_ch}^*$ if $\mathcal{A}$ creates a successful forgery. Otherwise, $\mathsf{ctr\_ch}^* = \mathsf{H}_{\mathsf{ch}}(m^*, R^*, L^*)$ is used while computing the response, contradicting the winning condition that (R^*, L^*) has never been signed by the owner of pk^*.

Type-(2) Forgery If the adversary $\mathcal{A}$ causes a Type-(2) forgery, we construct the following reduction to OMDL. The reduction goes through multiple layers of wrappers analogous to Type-(1) forgery. We will only explain the differences below.

Description of $\mathcal{B}$

- Initialization: $\mathcal{B}$ performs the same initialization operations as before. Additionally, it samples uniform $s \in [1, Q_k] \setminus \{t\}$, indicating one of the honest public keys whose owner is responsible for detecting Type-(2) forgery. We call it a *detector key*.
- $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{ch}}, \mathsf{OKeyReg}, \mathsf{OSignOff}$: Queries to these oracles are answered as before. Additionally, if $\mathcal{A}$ makes the s-th query with empty aux to $\mathsf{OKeyReg}$, $\mathcal{B}$ remembers an honestly generated key pk' as the detector key.
- $\mathsf{OSignOn}$: Whenever $\mathcal{A}$ queries $\mathsf{OSignOn}$ with an valid ssid, uid, m^*, $L^* = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, $\mathsf{offs}^* = (R^*_{1,1}, R^*_{1,2}, \ldots, R^*_{n,1}, R^*_{n,2})$, $\sigma_{i^*-1} = (R^*, \tilde{z}^*)$ with an index i^* corresponding to one of the honest keys, $\mathcal{B}$ performs the same operations as before, except that before sending a response to $\mathcal{A}$, it carries out the following checks. $\mathcal{B}$ verifies the so-far aggregation, i.e., check $g^{\tilde{z}^*} = \tilde{R}^* \cdot \prod_{i=1}^{i^*-1} \mathsf{pk}_i^{c^*}$, where $\tilde{R}^* = (\prod_{i=1}^{i^*-1} R^*_{i,1})(\prod_{i=1}^{i^*-1} R^*_{i,2})^{v^*}$, $v^* = \mathsf{H}_{\mathsf{non}}(m^*, \mathsf{offs}^*, L^*)$, and $c^* = \mathsf{H}_{\mathsf{ch}}(m^*, R^*, L^*)$. If the check passes and inputs are well-formed, $\mathcal{B}$ proceeds as follows.
  - If there exists some $j^* < i^*$ such that pk_{j^*} has never signed (m^*, L^*) while the check passes (i.e., the flag win would be set in the $\mathsf{OMS\text{-}UF\text{-}CMA}$ game), then $\mathcal{B}$ proceeds as follows. *First, if $\mathsf{pk}_{i^*} \neq \mathsf{pk}'$ then $\mathcal{B}$ aborts. Second, if for all $j < i^*$ $\mathsf{pk}_j \neq \mathsf{pk}^*$ or pk^* has already signed (m^*, L^*) then $\mathcal{B}$ aborts. Since the challenge pk^* is uniformly embedded in one of Q_k honest keys and the detector key pk' is uniformly embedded in one of the remaining $Q_k - 1$ honest keys, respectively, the probability that $\mathcal{B}$ does* ***not*** *abort here is at least $1/Q_k^2$.* If $\mathcal{B}$ does **not** abort (i.e., $\mathsf{pk}_{i^*} = \mathsf{pk}'$ **and** $\exists j^* < i^*$ such that $\mathsf{pk}^* = \mathsf{pk}_{j^*} \in L^*$ and pk_{j^*} has never signed (m^*, L^*)), $\mathcal{B}$ sets the following variables. Let $\mathsf{ctr\_ch}^*$ and $\mathsf{ctr\_non}^*$ be the values of $\mathsf{ctr\_ch}$ and $\mathsf{ctr\_non}$ at the moment $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*]$ is defined, respectively. That is, $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*] = c_{\mathsf{ctr\_ch}^*}$ and $v_1, \ldots, v_{\mathsf{ctr\_non}^*}$ have been used by the time $\mathrm{HT}_{\mathsf{ch}}[m^*, R^*, L^*]$ is set. Let K^* be the list of secret keys corresponding to the public keys in $L^* \setminus \{\mathsf{pk}_{j^*}\}$. Then $\mathcal{B}$ halts by outputting $(\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, \mathsf{out})$, where $\mathsf{out} = (i^*, j^*, L^*, K^*, \tilde{R}^*, c^*, \tilde{z}^*)$.

- Otherwise, $\mathcal{B}$ proceeds as in Type-(1) forgery.

Description of $\mathcal{C}$

- $\mathcal{C}$ runs Fork as before. After obtaining $(b, \mathsf{out}, \hat{\mathsf{out}})$ from Fork, if $b = 0$ $\mathcal{C}$ aborts. If $b = 1$, then $\mathcal{C}$ parses

$$\mathsf{out} = (\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, (i^*, j^*, L^*, K^*, \tilde{R}^*, c^*, \tilde{z}^*))$$

and

$$\hat{\mathsf{out}} = (\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, (i^*, j^*, L^*, K^*, \tilde{R}^*, \hat{c}^*, \hat{\tilde{z}}^*)),$$

from Fork.
We argue that $\mathsf{ctr\_ch}^*, \mathsf{ctr\_non}^*, i^*, j^*, L^*, K^*, \tilde{R}^*$ from two outputs are guaranteed to be identical whenever $b = 1$, $\mathsf{ctr\_ch}^*$ is by definition identical due to the success condition of Fork. In that case, $\mathsf{ctr\_non}^*$ must also be identical due to the way $\mathcal{B}$ defines it. Since $\mathsf{ctr\_ch}^*$ is identical and $\mathcal{B}$'s behavior is identical until the $\mathsf{ctr\_ch}^*$-th query is made to $\mathsf{H_{ch}}$, *it is guaranteed* that the corresponding input (m^*, R^*, L^*) to $\mathsf{H_{ch}}$ is the *same* in the two executions as in the analysis of Type-(1) forgeries. Since L^* is identical, K^* is also identical. Thanks to the abort condition of $\mathcal{B}$ when handling $\mathsf{OSignOn}$ queries, once L^* is fixed *it is guaranteed* that i^* and j^* are the same in the two executions.

The crucial difference with the analysis of Type-(1) is that the partial aggregation of $R_{i,j}$ for $i = 1, \ldots, i^* - 1$, denoted by $\tilde{R}^*$, is *not* included in the input of $\mathsf{H_{ch}}$. Still, we can make sure that $\tilde{R}^*$ is identical in the two executions. Thanks to the abort event happening within simulation of $\mathsf{H_{non}}$, *it is guaranteed* that the corresponding $(m^*, \mathsf{offs}^* = (R^*_{1,1}, R^*_{1,2}, \ldots, R^*_{n,1}, R^*_{n,2}), L^*)$ has been queried to $\mathsf{H_{non}}$ (which then determines the value of v^*) *before* $\mathsf{H_{ch}}$ is queried with (m^*, R^*, L^*). Thus, $\tilde{R}^* = (\prod_{i=1}^{i^*-1} R^*_{i,1}) \cdot (\prod_{i=1}^{i^*-1} R^*_{i,2})^{v^*}$ does not change after forking.

Now, due to the verification condition, if $b = 1$ we have that

$$g^{\tilde{z}^*} = \tilde{R}^* \cdot \left(\prod_{i=1}^{i^*-1} \mathsf{pk}_i \right)^{c_{\mathsf{ctr\_ch}^*}}$$

$$g^{\hat{\tilde{z}}^*} = \tilde{R}^* \cdot \left(\prod_{i=1}^{i^*-1} \mathsf{pk}_i \right)^{\hat{c}_{\mathsf{ctr\_ch}^*}} .$$

– $\mathcal{C}$ extracts the secret key for the challenge public key pk^* as follows.

$$\mathsf{sk}^* = \left((z^* - \hat{z}^*) \cdot (c_{\mathsf{ctr\_ch}^*} - \hat{c}_{\mathsf{ctr\_ch}^*})^{-1} - \sum_{i=1, i \neq j^*}^{i^*-1} \mathsf{sk}_i \right)$$

where co-signers' secret keys sk_i for $i \neq j^*$ are indeed known thanks to the key registration requirement.

– Now that $\mathcal{C}$ knows DLog of pk, what's left is computing DLogs of $(U_1, \ldots, U_{2Q_s})$. This can be done as in Type-(1) forgery with the same additive loss of Q^2/p.

By the forking lemma (Lemma 2) we have that

$$\mathsf{acc}(\mathcal{B}) = \left(\mathbf{Adv}_{\mathsf{OMS}}^{\mathsf{OMS\text{-}UF\text{-}CMA}}(\mathcal{A}) - \frac{Q^2}{p} \right) / Q_k^2$$

by accounting for the abort events happening when queries to $\mathsf{H}_{\mathsf{non}}$ are made and when a forged aggregate so-far is submitted to $\mathsf{OSignOn}$. The rest of the analysis is identical to Type-(1) forgery. Rearranging the terms, we obtain the advantage bound of the theorem statement □

C OMS in the Plain Public Key Model

In Construction 16 we describe a variant of OMS and show how to tweak the plain construction in Construction 14 to meet different features. In addition to the OMS syntax defined earlier, we introduce an key aggregation algorithm KAgg which takes a key list L and a position $k \leq n$ and outputs an aggregated key up to the k-th signer. The construction closely follows MuSig2, except that an aggregated key pk varies depending on the order of keys in the list L and SignOn additionally validates an aggregate so-far as in our basic OMS construction. Essentially, each signer needs to derive joint aggregate public keys by taking the random linear combination of all public keys, where coefficients are derived through the random oracle $\mathsf{H}_{\mathsf{agg}}$. To generalize the security proof of Theorem 15 to prove security of the present variant in the PPK model, one should set $\ell = 4$ as in [NRS21]. Accordingly, the reduction must invoke the forking lemma twice, first at an aggregation coefficient a_i for honest party's pk and second at challenge c. This will lead to a quartic reduction loss similar to [NRS21], which, howerver, could be circumvent by accepting stronger assumptions such as the algebraic group model [FKL18].

Construction 16: OMS construction supporting key aggregation and security in the PPK model

KeyGen() and SignOff are the same as in Construction 14. Vrfy(pk, m, σ) is identical to the usual Schnorr verification algorithm. Setup and SignOn are very similar to the ones in Construction 14, we highlight the lines that differ. KAgg is the additionalkey aggregation algorithm that combines an ordered list of keys into a single pk that looks like a usual Schnorr verification key.

```
Setup(1^λ) :
1: $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
2: $(n, \ell) \leftarrow \mathsf{poly}(\lambda)$
3: $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{ch}}, \mathsf{H}_{\mathsf{agg}} : \{0,1\}^* \to \mathbb{Z}_p^*$
4: $\sigma_0 := (1_{\mathbb{G}}, 0) \in \mathbb{G} \times \mathbb{Z}_p$
5: return $\mathsf{pp} := (\mathbb{G}, g, p, n, \ell, \mathsf{H}, \sigma_0)$

KAgg(L, k) :
1: Parse $L = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \in \mathbb{G}^n$
2: for $i \in [1, n]$ do
3:   $a_i = \mathsf{H}_{\mathsf{agg}}(L, \mathsf{pk}_i)$
4: $\mathsf{pk} = \prod_{i=1}^{k} \mathsf{pk}_i^{a_i}$
5: return pk

Vrfy(pk, m, σ) :
1: Parse $\sigma = (R, z) \in \mathbb{G} \times \mathbb{Z}_p$
2: $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, \mathsf{pk}) \in \mathbb{Z}_p$
3: if $g^z = R \cdot \mathsf{pk}^c$ then
4:   return 1
5: return 0
```

```
SignOn(st_i, sk_i, L, m, offs, σ_{i-1}) :
        ▷ Lines 1-9: processing independent of $m, \sigma_{i-1}$
1: Parse $\mathtt{st}_i = (r_{i,j} | R'_{i,j})_{j=1}^{\ell}$
2: Parse $\mathsf{offs} = ((R_{1,j})_{j=1}^{\ell}, \ldots, (R_{n,j})_{j=1}^{\ell})$
3: for $j \in [1, \ell]$ do
4:   $R_j := \prod_{k=1}^{\ell} R_{k,j}$
5:   $\tilde{R}_j := \prod_{k=1}^{i-1} R_{k,j}$
6:   if $R_j = 1_{\mathbb{G}}$ or $\tilde{R}_j = 1_{\mathbb{G}}$ then return $(\varepsilon, 0)$
7: $\mathsf{pk} := \mathsf{KAgg}(L, n)$
8: $\widetilde{\mathsf{pk}} := \mathsf{KAgg}(L, i-1)$
9: $a_i := \mathsf{H}_{\mathsf{agg}}(L, \mathsf{pk}_i)$
        ▷ Lines 10-22: processing that depends on $m, \sigma_{i-1}$
10: Parse $\sigma_{i-1} = (R, \tilde{z})$
11: $v \leftarrow \mathsf{H}_{\mathsf{non}}(m, \mathsf{offs}, \mathsf{pk})$
12: if $i = 1$ then
13:   $R := \prod_{j=1}^{\ell} R_j^{v^{j-1}}$
14:   $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, \mathsf{pk})$
15: else                ▷ Check so-far aggregation
16:   if $R \neq \prod_{j=1}^{\ell} R_j^{v^{j-1}}$ then return $(\varepsilon, 0)$
17:   $\tilde{R} := \prod_{j=1}^{\ell} \tilde{R}_j^{v^{j-1}}$
18:   $c \leftarrow \mathsf{H}_{\mathsf{ch}}(m, R, \mathsf{pk})$
19:   if $g^{\tilde{z}} \neq \tilde{R} \cdot \widetilde{\mathsf{pk}}^c$ then return $(\varepsilon, 0)$
20: $z_i := c \cdot a_i \cdot \mathsf{sk}_i + \sum_{j=1}^{\ell} v^{j-1} \cdot r_{i,j}$
21: $z := \tilde{z} + z_i$
22: $\sigma_i := (R, z)$
23: return $(\sigma_i, 1)$
```