Subset-optimized BLS Multi-signature with Key Aggregation

Foteini Baldimtsi[1,2], Konstantinos Kryptos Chalkias[1], François Garillot[3⋆], Jonas Lindstrøm[1], Ben Riva[1], Arnab Roy[1], Mahdi Sedaghat[4⋆], Alberto Sonnino[1,5], Pun Waiwitlikhit[1,6⋆], and Joy Wang[1]

[1] Mysten Labs, research@mystenlabs.com
[2] George Mason University
[3] Lurk Labs
[4] COSIC, KU Leuven, Leuven, Belgium
[5] University College London, London, UK
[6] Stanford University

Abstract. We propose a variant of the original Boneh, Drijvers, and Neven (Asiacrypt'18) BLS multi-signature aggregation scheme, which is best suited to applications where the full set of potential signers is fixed and known and any subset I of this group can create a multi-signature over a message m. This setup is very common in proof-of-stake blockchains where if you assume a total of $3f$ validators, a $2f + 1$ majority can sign transactions and/or blocks and is secure against *rogue-key* attacks without requiring a proof of key possession mechanism.

In our scheme, instead of randomizing the aggregated signatures, we have a one-time randomization phase of the public keys: each public key is replaced by a sticky randomized version (for which each participant can still compute the derived private key). The main benefit compared to the original Boneh et al. approach is that since our randomization process happens only once and not per signature we can have significant savings during aggregation and verification without requiring a proof of possession. Specifically, for a subset I of t signers, we save t exponentiations in $\mathbb{G}_2$ at aggregation and t exponentiations in $\mathbb{G}_1$ at verification or vice versa, depending on which BLS mode we prefer: *minPK* (public keys in $\mathbb{G}_1$) or *minSig* (signatures in $\mathbb{G}_1$).

Interestingly, our security proof requires a significant departure from the co-CDH based proof of Boneh et al. When n (size of the universal set of signers) is small, we prove our protocol secure in the Algebraic Group and Random Oracle models based on the hardness of the Discrete Log problem. For larger n, our proof also requires the Random Modular Subset Sum (RMSS) problem.

Keywords: BLS · multi-signatures · signature aggregation · blockchain.

⋆ Work done at Mysten Labs.

1 Introduction

A *multi-signature* scheme [26] allows a set of n signers to generate a short signature σ, on the *same* message m (where the size of the signature should be independent of the number of signers). To verify the multi-signature one needs all the signers' public keys, m and σ. A useful property of many multi-signature schemes, is that they additionally support *public-key aggregation*; thus the verifier only needs a short aggregated public key instead of an explicit list of all n public keys[1].

Multi-signatures have attracted considerable attention in recent years due to their applications in the blockchain setting. The initial application under consideration was the creation of multi-user wallets where multiple users share ownership of funds and can utilize multi-signatures to collectively sign transactions for spending these funds. Notably, multi-signatures with public key aggregation play a fundamental role in the scalability of blockchain systems since they allow the compression of posted signatures and verification keys up to a factor of n. Of particular interest are multi-signature schemes the verification algorithm of which, is fully compatible with algorithms supported by blockchain systems such as Schnorr [39] or BLS [9].

When designing multi-signature schemes, a significant challenge is to avoid the so-called *rogue-key* attacks: a forgery attack caused in schemes where the adversaries are allowed to choose their public keys arbitrarily. In a typical multi-signature rogue key attack, the adversary would attempt to create a public key that is a function of an honest user's key allowing possible forgeries. An easy defense against such attacks is to require the parties to present a proof-of-possession (PoP), i.e., a proof of knowledge of their secret keys. However this complicates implementations and is not compatible with existing infrastructures. In particular, in order to avoid long-range attacks [14] in proof of stake protocols, a commonly used approach is to enforce the validators to rotate their keys in each epoch. Key rotation creates the need to post a PoP proof for each validator in each epoch which in turn might end up in more costly protocols than those without PoP. The current widely adopted model for multi-signatures is known as the *plain public key model*, which was introduced by Bellare and Neven [2]. In this model, each signer independently generates their own key pair, and no proofs of possession are required. Our focus in this paper is specifically on the plain public key model and in particular on the BLS multi-signature scheme given its wide adoption in the blockchain space.

The BLS multi-signature [8]. Boneh–Lynn–Shacham (BLS) proposed an efficient signature scheme in [9] that uses pairing friendly elliptic curves. BLS signatures are important in various blockchain related projects including the Chia

[1] We note that *aggregate signatures* are a more general primitive, which as opposed to multi-signatures, allow the aggregation of n signatures of *different* messages in a short single signature.

network[2], the Plumo ultralight client [42], and Dfinity's random beacon [24]. A standardization attempt for BLS with IETF is ongoing since 2019.[3]

Technically, BLS supports multi-signing with signature/public-key aggregation [8] in a non-interactive, deterministic and non-malleable manner. More specifically, the scheme proposed in [8] has signature sizes of size $O(\lambda)$, where λ is the security parameter and its security is proven in the random oracle (RO) model under a generalization of the Computational Diffie-Hellman (CDH) assumption, called co-CDH [9]. Although ECDSA signatures can be verified much faster individually, BLS signatures on the same message can be verified much faster in the aggregated form, therefore making it more practical for multiple validators attesting the same block or transaction. In particular, n aggregated signatures on the same message can be performed with just 2 pairings instead of $n + 1$ [8]. Notably, the most recent Ethereum Consensus client deployed on the mainnet has adopted BLS signatures for validators to attest block proposals [19,17] addressing the verification bottleneck [15].

The BLS multi-signature [8] avoids the need for proofs of possession by following the paradigm of [2] which allows the public keys to be aggregated without the need to check their validity through a series of signature and public key randomizations. Given an efficiently computable bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ over groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of same prime order q (cf. Appendix B.1). Let g_1, g_2 be generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively, along with hash functions $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$, the BLS multi-signature of [8] works as follows:

- **Key Generation**: A secret/public key pair is denoted by $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathsf{pk} = g_2^{\mathsf{sk}} \in \mathbb{G}_2$. Let PK be the set of public keys of n signers.
- **Signing**: To compute a multi-signature over PK, each party computes $\sigma_i = \mathsf{H}_0(m)^{a_i \mathsf{sk}_i}$, where $a_i = \mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})$ and a designated combiner computes the final aggregated signature to be $\sigma = \prod_{i=1}^{n} \sigma_i$.
- **Verification**: The aggregated signature can be verified by checking $e(\sigma, g_2) = e(\mathsf{H}_0(m), \mathsf{apk}_{\mathsf{PK}})$, where the aggregated public key across the n signers is obtained via $\mathsf{apk}_{\mathsf{PK}} = \prod_{i=1}^{n} \mathsf{pk}_i^{\mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})}$.

Our Results. The BLS multi-signature of [8] requires a total of n exponentiations in $\mathbb{G}_1$ during the aggregation of signature shares and n exponentiations in $\mathbb{G}_2$ during the computation of apk for signature verification in order to randomize the signature and keys. While this cost is needed if the set of the n signers is dynamic and constantly updated, our results are inspired by the observation that it is not necessary in all settings. For example, in proof-of-stake settings it is common to have a static set of n signers per epoch during which multiple subsets of the n signers would be required to engage in multi-signing.

Our protocol takes advantage of that setting and moves the need of signature and apk randomization to a *one-time* public key randomization process which happens once the set of n signers is fixed. Thus, instead of randomizing every

[2] https://github.com/Chia-Network/bls-signatures
[3] https://www.ietf.org/id/draft-irtf-cfrg-bls-signature-05.html.

single multi-signature and the corresponding aggregated public key, we randomize each signer's public key *once* at the beginning of the protocol (or at the beginning of each epoch for the consensus setting[4]), and then for every signing subset I we simply aggregate signatures and the randomized public keys together via a cheap multiplication and without the need for any exponentiations. The advantage is that now the cost of a multi-signature is the same as a regular BLS signature. We first provide a new set of definitions that allow for *subset* multi-signing with key randomization in Section 3, and then present our construction in Section 4. However, a natural question arises:

How can we prove that a subset multi-signature scheme, which allows the adversary to adaptively sample any subset of a fixed universe set of signers, is secure against a rogue-key attack?

Security. The security analysis of our scheme is technically interesting as it has to significantly depart from the analysis of the standard BLS multi-signature [8]. In particular, the rewinding approach of [8] would fail in our case unless the adversary was forced to declare the signer subset for which it would output its forgery (to explain the technicality of our proof we give a security proof for this weaker adversary in Appendix C.1). To overcome this limitation, we leverage the algebraic group model (AGM) [21] and we prove our scheme secure under the discrete logarithm assumption in the combined AGM + ROM.

The Necessity of RMSS assumption. Our proof incurs a security loss of 2^n assuming Discrete Log (DL) hardness in AGM, thus failing to provide any guarantee when n is large. This is not just a proof artifact: we show a concrete attack if the adversary can solve Random Modular Subset Sum (RMSS) problems. The subset sum problem is a fundamental NP-complete problem. It involves a set $S = \{s_1, s_2, \ldots, s_n\}$ of integers and an integer target t. The task is to determine if there exists a subset $I \subseteq S$ that sums to the target t. The Modular Subset Sum (MSS) problem, is a variant that assumes elements and operations in the finite cyclic group $\mathbb{Z}_q$, instead of integers. The RMSS assumption conjectures that the MSS problem is hard on average, that is, when all the input elements are chosen randomly. There is a long line of work analyzing the complexity of RMSS problems [10,27,20,11,30] for different parameters. Impagliazzo and Naor [25] constructed pseudo-random generators and one-way hash functions based on the hardness of RMSS.

In Section 5, we formally prove the security of our proposed construction under both DL and RMSS assumptions. An important note for practical applications is that when the number of possible subsets is negligibly smaller than the size of the output space of the hash function H_1, then the probability of existence of a subset sum solution is negligible. In this case, no subset attacks are

[4] This is much preferable to the PoP approach as it avoids the need for zero-knowledge proofs (which includes prover/verifier costs as well as support for ZK from the underlying blockchain.

possible and just the Discrete Log assumption is enough. We provide a concrete analysis for specific cases in Section 5.

Implementation. Finally, in Section 6, we provide an implementation of our construction and a baseline comparison with [8]. Notably, for $n = 500$ signers our signature aggregation saves between 150 ms - 180 ms; and our signature verification saves between 40 ms - 75 ms (depending on whether signatures are in $\mathbb{G}_2$ and keys in $\mathbb{G}_1$ or vice versa). Our performance benefit increases linearly with the number of signers.

1.1 Related Work

Numerous proposals for multi-signatures have been developed, each relying on distinct assumptions and analyzed under various security models. In terms of assumptions, we have schemes based on RSA [26,35], Discrete Logarithm, DDH and/or Schnorr signatures [34,2,1,33,32,13,28,41,36], bilinear pairings [5,29,38,6,8,12] and more recently lattice based assumptions [18,22]. Pairing based schemes can lead to protocols that are non-interactive [5,38,8] while most of the DL based protocols require two or three rounds of interaction and often require non-interactive assumptions. Recent works focus on removing the need for interactive assumptions [41,36] and/or achieving tighter security reductions [36].

Accountable-Subgroup Multi-signatures. A relevant notion is that of *accountable-subgroup multi-signatures* (ASM) introduced by Micali et al. [31]. At a high level, ASM are defined to allow any subset I of the n signers in PK to jointly sign a message m, in a way that the subgroup I is accountable for signing (i.e. the verifier can derive the signing subset). The main difference from our notion of subgroup optimized signatures, is that ASM requires an *interactive* group setup phase in order for users to create their "membership key" for the group. More concretely, for the case of BLS, their AMS construction [8] requires an interactive group setup phase with n^2 communication costs, where all the n signers compute multi-signatures on the aggregate public key and the index of every signer (i.e. the i-th signer of the set PK obtains a "membership key" which is a multi-signature on $(\mathsf{apk}_{\mathsf{PK}}, i)$). Additionally, the AMS scheme of [8] requires an additional pairing during signature verification. An important note is that the interactive group setup of AMS is not compatible with our main application scenario of PoS committee signing: if the PoS committee is formed and all PKs are public (thus apk is fixed), if one committee member does not participate in the setup phase then the scheme fails. This is not a problem in our scheme, where members create their keys independently.

2 Preliminaries

Basic Notations. Throughout, we denote the security parameter by λ and its unary representation by 1^λ. A function $\mathsf{negl}(\lambda)$ is called *negligible* if for every positive polynomial $p(\lambda)$, there exists λ_0 s.t. for all $\lambda > \lambda_0 \colon \mathsf{negl}(\lambda) < 1/p(\lambda)$.

The set $\{1, \ldots, n\}$ is denoted as $[1, n]$ for a positive integer $n > 1$. To check if two elements are equal, we use the symbol "=". To assign a value to a variable we use ":=", however randomized assignment is denoted with $a \xleftarrow{\$} A$, where A is a randomized algorithm.

Computational Assumptions. For our security proofs of BLS subset multi-signatures, we recall the assumptions of Discrete Log (DL), a generalization of the Computational Diffie-Hellman (CDH) assumption, called co-CDH [9], and the Random Modular Subset Sum (RMSS) assumption [25,30].

Definition 1 (Discrete Logarithm Problem). *For a group $\mathbb{G} = \langle g \rangle$ of prime order q, we define the advantage $\mathsf{Adv}^{DL}_{\mathbb{G}}(\mathcal{A})$ of an adversary $\mathcal{A}$ as* $\Pr\left[z' = z \ : \ z \xleftarrow{\$} \mathbb{Z}_q,\ Z \leftarrow g^z,\ z' \leftarrow \mathcal{A}(g, Z)\right]$*, where the probability is taken over the random choices of the adversary $\mathcal{A}$ and the random selection of z. DL is (τ, ϵ)-hard if there is no adversary $\mathcal{A}$ that can break the DL problem in time τ and with advantage $\mathsf{Adv}^{DL}_{\mathbb{G}}(\mathcal{A}) > \epsilon$.*

Following [21], we show that our construction is secure in AGM assuming the hardness of Discrete Log problem.

Definition 2 (Computational co-Diffie-Hellman Problem). *For groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order q, we define the advantage $\mathsf{Adv}^{co\text{-}DH}_{\mathbb{G}_1,\mathbb{G}_2}(\mathcal{A})$ of an adversary $\mathcal{A}$ as* $\Pr\left[y = g_1^{\alpha\beta} \ : \ (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_q^2,\ y \leftarrow \mathcal{A}(g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})\right]$*, where the probability is taken over the random choices of the adversary $\mathcal{A}$ and the random selection of (α, β). co-CDH is (τ, ϵ)-hard if there is no adversary $\mathcal{A}$ that can break the co-CDH problem in time τ and with advantage $\mathsf{Adv}^{co\text{-}DH}_{\mathbb{G}_1,\mathbb{G}_2}(\mathcal{A}) > \epsilon$.*

For symmetric pairing groups, co-CDH reduces to standard CDH.

Definition 3 (Random Modular Subset Sum (RMSS) Problem). *For a prime number q, we define the advantage $\mathsf{Adv}^{RMSS}_{n,q}(\mathcal{A})$ of an adversary $\mathcal{A}$ as* $\Pr\left[\sum_{i \in I} s_i = t \ : \ S = \{s_i\}_{i=1}^{n} \xleftarrow{\$} \mathbb{Z}_q^n,\ t \leftarrow \mathbb{Z}_q,\ S \supseteq I \leftarrow \mathcal{A}(S, t)\right]$*, where the probability is taken over the random choices of the adversary $\mathcal{A}$ and the random selection of (S, t). RMSS is (τ, ϵ)-hard if there is no adversary that can break the RMSS problem in time τ and with advantage $\mathsf{Adv}^{RMSS}_{n,q}(\mathcal{A}) > \epsilon$.*

Impagliazzo and Naor [25] argued that the hardest instances of RMSS are characterized by $n = c\log(q)$, where c is a constant factor. Although RMSS is poly-time solvable [27,20] through a reduction to lattice SVP problems for $n = O(\sqrt{\log(q)})$, this is of lesser consequence to us, as the probability of the existence of a solution is low.

Algebraic Group Model. The algebraic group model (AGM) introduced in [21], is a model for security proofs that lies between the generic group model (GGM) and the standard model. In AGM the adversary is considered *algebraic*:

whenever it outputs a group element, it also outputs a representation of that group element relative to all of the other input group elements the algorithm has received up to that point.

Definition 4 (Algebraic Algorithm [21]). *Over a group $\mathbb{G}$, an algorithm $\mathcal{A}$ is called algebraic, if for all $\mathcal{A}$'s group elements outputs $\zeta \in \mathbb{G}$, $\mathcal{A}$ additionally provides a representation vector, $z = (z_0, \ldots, z_m)$ of integers such that $\zeta = \prod_i g_i^{z_i}$, where $(g_0, \ldots, g_m)$ is the list of group elements $\mathcal{A}$ has seen thus far (w.l.o.g. $g_0 = g$).*

The AGM model was used before to tighten the security reduction of the standard BLS signature scheme [21]. While previous reductions non-tightly reduced from the CDH problem with a tightness loss linear in the number of signing queries, [21] provided a tight reduction in the AGM+ROM under the hardness of discrete logarithm assumption.

3 SMSKR: Syntax and Security Properties

Informally, a multi-signature (MS) allows multiple signers with public keys $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ to sign the same message with a signature size independent to the number of signers. The set of signers' public keys over the public key set PK is aggregated into a single key $\mathsf{apk}_{\mathsf{PK}}$. The formal definition can be found in Appendix B.4.

Next, we formally define the Subset Multi-Signatures with Key Randomization (SMSKR) as an extension to the original multi-signatures. Compared to MS, the main difference in our scheme is that we assume a fixed set of signers with public keys $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ and we allow different subsets I of them to compute signatures. However, during signing phase, the signer does not have to be aware of who are the rest of the subset members as long as it knows PK. We separate the signing process from the signature aggregation and divide the signing algorithm into two: key randomization and signing algorithms. The separation of key randomization and signing, is the *key point* of our construction that allows for efficient implementations in the blockchain setting. Assuming a known set of eligible signers $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ at the beginning of a blockchain epoch, all entities can appropriately randomize their keys *once at the beginning of the epoch* and then participate in multiple BLS multi-signatures for any subset of $[1, n]$. The advantage is that now the cost of a multi-signature is the same as a regular BLS signature (plus the cost of a one-time key randomization), while the users do need to know who participates in the multi-signing amongst the eligible signers. This is different than the original BLS multi-signature [8], where the secret keys of each user were repeatedly randomized during each signing session for the specific set of signers that participated in each multi-signature.

Definition 5 (Subset-Optimised Multi-Signature with Key Randomization). *Over a message space $\mathcal{M}$, a subset-optimised multi-signature with Key Randomization (SMSKR) consists of the following PPT algorithms:*

- $\mathsf{pp} \leftarrow \textbf{\textit{SMSKR.Setup}}(1^\lambda)$*: Take the security parameter* λ *in its unary representation, and output the public parameters* pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \textbf{\textit{SMSKR.KeyGen}}(\mathsf{pp})$*: Take the public parameters* pp*, and output a pair of public/secret keys* $(\mathsf{pk}, \mathsf{sk})$.
- $(\mathsf{sk}_i^\star, \mathsf{pk}_i^\star) \leftarrow \textbf{\textit{SMSKR.KeyRand}}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{PK})$*: Take secret key* sk_i *and set of public keys* PK *as inputs, generate randomized secret/public*[5] *keys* $(\mathsf{sk}_i^\star, \mathsf{pk}_i^\star)$ *for user* i *and for the set of signers captured in* PK.
- $\sigma_i \leftarrow \textbf{\textit{SMSKR.Sign}}(\mathsf{pp}, \mathsf{sk}_i^\star, m)$*: Take rerandomized secret signing key,* $\mathsf{sk}_i^\star$*, and a message* $m \in \mathcal{M}$*, and output the signature share* σ_i.
- $\sigma_I \leftarrow \textbf{\textit{SMSKR.SigAggr}}(\mathsf{pp}, \{\sigma_i\}_{i\in I}, m)$*: Take* $|I|$ *signature shares on message* m*, i.e.* σ_i *for all* $i \in I$*, and output an aggregated signature* σ_I *for the subset* I*. (It could potentially also take* PK *as input.)*
- $\mathsf{apk}_I^* \leftarrow \textbf{\textit{SMSKR.SubsetKeyAggr}}(\mathsf{pp}, \{\mathsf{pk}_i^\star\}_{i\in I})$*: Take the set of rerandomized public keys,* $\mathsf{pk}_i^\star$ *for all* $i \in I$ *as input, and output a subset aggregated key* apk_I^*.
- $0/1 \leftarrow \textbf{\textit{SMSKR.Verify}}(\mathsf{pp}, \mathsf{apk}_I^*, \sigma_I, m)$*: Output* 1 *if the signature* σ_I *verifies for message* m *under the aggregated public key* apk_I^**, and 0 otherwise.*

Similar to digital signatures, discussed in Appendix B.3, an SMSKR scheme has two main security properties: correctness and unforgeability.

Definition 6 (Correctness). *An SMSKR scheme satisfies correctness, if for every* $n > 1$*,* $m \in \mathcal{M}$*, and subset* $I \subseteq [1, n]$ *of signers, we have:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \textit{Setup}(1^\lambda), \{(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \textit{KeyGen}(\mathsf{pp})\}_{i=1}^n, \\ \{(\mathsf{sk}_i^\star, \mathsf{pk}_i^\star) \leftarrow \textit{KeyRand}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{PK})\}_{i=1}^n, \\ \mathsf{apk}_I^* \leftarrow \textit{SubsetKeyAggr}(\mathsf{pp}, \{\mathsf{pk}_i^\star\}_{i\in I}), \\ \textit{Verify}\left(\mathsf{pp}, \mathsf{apk}_I^*, \textit{SigAggr}(\mathsf{pp}, \{\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_i^\star, m)\}_{i\in I}, m), m\right) = 1\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Informally, an SMSKR is unforgeable if an adversary cannot forge a signature that verifies under $\mathsf{apk}_I^\star$ for a set of signers where at least one signer is honest. In other words, assuming n signers, even if an adversary has corrupted all but one signer with public key pk_0, the user should still not be able to forge a signature that verifies under $apk_I^\star$ that includes pk_0. Formally:

Definition 7 (Unforgeability). *An SMSKR scheme over message space* $\mathcal{M}$ *is called unforgeable against chosen message attacks if for all PPT adversaries* $\mathcal{A}$ *playing game* $\mathsf{SMSKR\text{-}UF\text{-}CMA}_{\mathcal{A}}$*, as described in Figure 1, there exists a negligible function* $\mathsf{negl}(\lambda)$ *s.t. we have:*

$$Adv_{\mathcal{A}}^{\mathsf{SMSKR\text{-}UF\text{-}CMA}}(\lambda) = \Pr\left[\mathbf{G}_{\mathcal{A}}^{\mathsf{SMSKR\text{-}UF\text{-}CMA}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ .$$

An SMSKR is called *weakly unforgeable*, i.e. $\mathsf{W\text{-}SMSKR\text{-}UF\text{-}CMA}$ secure, described in Figure 1 if the adversary forces to declare the target subset I before getting access to the oracles. Note that in this case, the adversary might be granted more oracle accesses based on the publicly available functions in any given scheme.

[5] In most cases, computing $\mathsf{pk}_i^\star$ does not need any secret, thus, this algorithm could be defined separately for sk and pk.

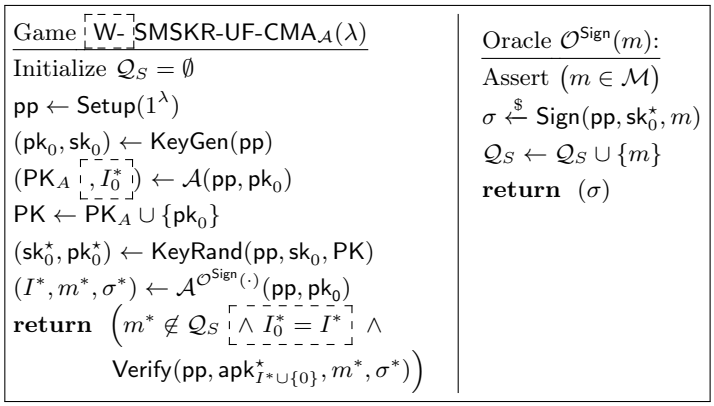

Fig. 1: Games defining the SMSKR-UF-CMA unforgeability of multi signatures and weakly notion as W-SMSKR-UF-CMA.

4 Our SMSKR Construction

In this section, we propose an efficient subset-optimized multi-signature with key randomization scheme. Towards describing our SMSKR scheme, we start by recalling the standard BLS signature scheme [9]. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear pairing as defined in Definition 8. Let g_1, g_2 be generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively and let $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$ be a hash-to-curve function. BLS can be instantiated either as *minSig* where signatures are in $\mathbb{G}_1$ and public keys in $\mathbb{G}_2$, or as *minPK* where signatures are in $\mathbb{G}_2$ and keys are in $\mathbb{G}_1$. Below we take the *minSig* approach. Given the formal definition of digital signatures in Appendix B.3, the BLS signature over an arbitrary messape space $\mathcal{M} := \{0,1\}^*$ consists of the following algorithms:

- **BLS**.$\mathsf{Setup}(1^\lambda)$: Output a bilinear group $\mathsf{pp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$.
- **BLS**.$\mathsf{KeyGen}(\mathsf{pp})$: Given the parameters pp, output a pair of public/secret keys $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathsf{pk} = g_2^{\mathsf{sk}} \in \mathbb{G}_2$.
- **BLS**.$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: Given a secret key sk and a message $m \in \mathcal{M}$, output a signature $\sigma = \mathsf{H}_0(m)^{\mathsf{sk}} \in \mathbb{G}_1$.
- **BLS**.$\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, \sigma, m)$: Given a public key $\mathsf{pk} \in \mathbb{G}_2$, a signature $\sigma \in \mathbb{G}_1$ and a message $m \in \mathcal{M}$, output 1 if $e(\sigma, g_2) = e(\mathsf{H}_0(m), \mathsf{pk})$, and 0 otherwise.

BLS signatures can support multi-signing with public-key aggregation. Given the formal definition of MS schemes in Appendix B.4, we recall the Boneh et al.'s MS scheme [8]. We note that there exist two descriptions of the scheme: one in the full and proceedings version of the paper [8], and a slightly modified version of the scheme described by the authors in a blog-post [7]. We first recall the scheme from the full/proceedings version [8] and then discuss the differences with the blog-post version [7].

Given the same setup as in BLS signatures and an additional hash function $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$, Boneh et al.'s MS scheme works as follows:

- $\mathbf{MS}.\mathsf{Setup}(1^\lambda)$: Run $\mathbf{BLS}.\mathsf{Setup}(1^\lambda)$ and output pp.
- $\mathbf{MS}.\mathsf{KeyGen}(\mathsf{pp})$: Run $\mathbf{BLS}.\mathsf{KeyGen}(\mathsf{pp})$ and output $(\mathsf{sk}_i, \mathsf{pk}_i)$ for all $i \in [1, n]$.
- $\mathbf{MS}.\mathsf{Sign}(\mathsf{pp}, \mathsf{PK}, \{\mathsf{sk}_i\}_{i\in[1,n]}, m)$: On input the set of public keys PK, a secret signing key sk_i and a message m, compute $\sigma_i = \mathsf{H}_0(m)^{a_i \mathsf{sk}_i}$, where $a_i = \mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})$. Send the signature to a designated combiner who computes the final signature to be $\sigma_{\mathsf{PK}} = \prod_{i=1}^{n} \sigma_i$. (The designated combiner can be one of the signers or an external party.)
- $\mathbf{MS}.\mathsf{KeyAggr}(\mathsf{pp}, \mathsf{PK})$: Given $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, then output $\mathsf{apk}_{\mathsf{PK}} = \prod_{i=1}^{n} \mathsf{pk}_i^{\mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})}$.
- $\mathbf{MS}.\mathsf{Verify}(\mathsf{pp}, \mathsf{apk}_{\mathsf{PK}}, \sigma_{\mathsf{PK}}, m)$: Output $\mathbf{BLS}.\mathsf{Verify}(\mathsf{pp}, \mathsf{apk}_{\mathsf{PK}}, \sigma_{\mathsf{PK}}, m)$.

The main difference between the scheme above and its blog-post version [7], is that the latter scheme makes the signature aggregation process distinct while at the same time includes all the randomizations. Users compute their signatures as regular, individual BLS signatures on message m, completely oblivious of who else is signing the message. Then, an aggregator, given the set of public keys for the signers PK, and all individual signatures σ_i, computes the aggregated multi-sig. As the scheme is described in the blog-post, the aggregator has to pay the cost of n exponentiations in $\mathbb{G}_1$, instead of amortizing this cost across signers.

Next, we propose our SMSKR, which is essentially a variant of the original Boneh et al.'s BLS multi-signature scheme [8] enabling adaptive subset signing and key aggregation. Per our definition, we divide the signing algorithm, $\mathbf{MS}.\mathsf{Sign}(\mathsf{pp}, \mathsf{PK}, \mathsf{sk}_i, m)$, into two modules that allow for key randomization. We assume access to the functions from $\mathbf{MS}$ scheme (as described above) over the same groups using $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$ and $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q^*$ hash functions and define our SMSKR construction as follows[6]:

- $\mathbf{SMSKR}.\mathsf{Setup}(1^\lambda)$: Run $\mathbf{MS}.\mathsf{Setup}(1^\lambda)$ and output public parameters, pp.
- $\mathbf{SMSKR}.\mathsf{KeyGen}(\mathsf{pp})$: Run $\mathbf{MS}.\mathsf{KeyGen}(\mathsf{pp})$ and output the key-pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ for all $i \in [1, n]$.
- $\mathbf{SMSKR}.\mathsf{KeyRand}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{PK})$: Given a set of public keys $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, output the randomized public/secret keys $\mathsf{pk}_i^\star = \mathsf{pk}_i^{\mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})}$ and $\mathsf{sk}_i^\star = \mathsf{sk}_i \cdot \mathsf{H}_1(\mathsf{pk}_i \parallel \mathsf{PK})$, respectively.[7]
- $\mathbf{SMSKR}.\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_i^\star, m)$: On input a randomized secret signing key $\mathsf{sk}_i^\star$ and a message m, output $\sigma_i = \mathsf{H}_0(m)^{\mathsf{sk}_i^\star}$.[8]
- $\mathbf{SMSKR}.\mathsf{SigAggr}(\mathsf{pp}, \{\sigma_j\}_{j\in I}, m)$: Given a set of signatures from the corresponding parties in subset I and message m, and output the aggregated multi-signature $\sigma_I = \prod_{j\in I} \sigma_j$.
- $\mathbf{SMSKR}.\mathsf{SubsetKeyAggr}(\mathsf{pp}, \{\mathsf{pk}_j^\star\}_{j\in I})$: Given a subset of the parties denoted by their indices $I \subseteq [1, n]$ and their randomized public keys, output $\mathsf{apk}_I^\star = \prod_{j\in I} \mathsf{pk}_j^\star$.

[6] We proposed SMSKR in the *minSig* mode, it can easily be extended to *minPK*.

[7] The randomization of pk can happen by any third party – no secret required.

[8] Note that the sign algorithm uses an already randomized secret key (and thus there is no need to parse PK again.)

– $\mathsf{SMSKR}.\mathsf{Verify}(\mathsf{pp}, \mathsf{apk}_I^\star, \sigma_I, m)$: Given an aggregated public key $\mathsf{apk}_I^\star \in \mathbb{G}_2$, an aggregated signature $\sigma_I \in \mathbb{G}_1$ and a message $m \in \mathcal{M}$, and output $\mathsf{MS}.\mathsf{Verify}(\mathsf{pp}, \mathsf{apk}_I^\star, \sigma_I, m)$.

Remark 1. In certain blockchain applications, a party i might not have direct access to the private key sk_i, but only to a BLS signing oracle over the original private key. Thus, it could request a signature over m, receive $\sigma_i = \mathsf{H}_0(m)^{\mathsf{sk}_i}$ and then randomize the signature by computing $\sigma_i^* = \sigma_i^{\mathsf{H}_1(\mathsf{pk}_i \| \mathsf{PK})}$. This approach is more expensive because signature randomization requires an operation in $\mathbb{G}_1$, while SMSKR's default approach randomizes the private key, which is a field operation and faster.

Remark 2. In blockchain settings, it is very reasonable to assume that applications already have access to full set of randomized public keys $\mathsf{PK}^\star$ and they only require a bitmap that defines the indices of the subset of entities that signed the message in $\mathsf{SubsetKeyAggr}$ algorithm. To optimize even further, applications could cache common subsets of I and their corresponding aggregated keys.

5 Security Analysis

In this section, we formally prove the proposed SMS in Section 4 is secure. To showcase the complication of the security analysis for our scheme, in Appendix C.1 we first discuss the security of our scheme for a weaker unforgeability adversary which in the security game of Definition 7, declares a target subset I for which it will output its forgery before starting its queries.

However, in the standard definition of unforgeability, in Definition 7, the adversary can adaptively change the subset of target signers even after getting access to the oracles. Therefore, we use a reduction strategy that works even if the target subset changes after a rewind, relying on the Algebraic Group Model (AGM) and Random Oracle Model (ROM). Although we still have a 2^n security loss in the reduction, we show that this loss is intrinsic and can only be avoided by additionally assuming hardness of the RMSS problem, stated in Definition 3.

We follow the security proof of BLS signature in [21] up to a certain point - in particular they provide reduction strategies to address 2 cases that may arise. In addition to those 2 cases, we have an important 3rd case where our analysis highlights a fundamental distinguishing characteristic of the SMSKR construction. Specifically, there is a subset-sum attack possible in SMSKR which can either have a negligibly low probability statistically, or be able to be argued to be computationally hard based on the RMSS assumption. We describe the proof in both scenarios and analyze how we can leverage both the DL and RMSS assumptions depending on concrete subset and universal set sizes.

Theorem 1. *The proposed SMSKR in Section 4 is* $\mathsf{SMSKR\text{-}UF\text{-}CMA}$ *secure, as defined in Definition 7, under the hardness of the Discrete Logarithm problem, as stated in Definition 1, in the AGM+ROM.*

The proof is presented in Appendix C.2. It has a 2^n security loss which we argue that is intrinsic, unless we resort to a hardness assumption related to random modular subset sums (RMSS), such as Definition 3. We describe a concrete attack[9].

Attack. An adversary which can efficiently solve RMSS instances can break the security of the system as we show here. Once this adversary $\mathcal{A}$ receives a target $\mathsf{pk}_0 = g^x$, it chooses $\{(u_i, v_i)\}_{i=1}^n$ randomly as $\mathbb{Z}_q$ elements and sends $\mathsf{PK}_A = \{\mathsf{pk}_i = g^{u_i}\mathsf{pk}_0^{v_i}\}_{i=1}^n$ to the challenger. Let $h_i = \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})$ for all $i \in [0, n]$, where $\mathsf{PK} = \mathsf{PK}_A \cup \{\mathsf{pk}_0\}$. The adversary solves the following RMSS instance:

- Target sum: $-h_0 \mod q$
- Set: $\{h_i \cdot v_i\}_{i=1}^n$

If $n = \Omega(\log q)$, w.h.p. there is a solution. Let's say the solution is $I \subseteq [1, n]$. This means $h_0 + \sum_{i\in I} h_i v_i = 0$. An SMSKR signature on a message m^* with subset $I \cup \{0\}$ is thus $\mathsf{H}_0(m^*)^{h_0 x + \sum_{i\in I} h_i(u_i + v_i x)} = \mathsf{H}_0(m^*)^{\sum_{i\in I} h_i u_i}$. This quantity can be readily computed by the adversary as it is independent of x.

.

Proof with RMSS assumption. The above attack highlights the need to assume that random subset sum problems are hard to compute for an adversary. In fact, we show that the RMSS (Definition 3) and discrete logarithm problems together suffice to prove security of the scheme without an exponential loss in reduction. The proof of the theorem bellow can be found in Appendix C.3.

Theorem 2. *The proposed SMSKR in Section 4 is* $\mathsf{SMSKR\text{-}UF\text{-}CMA}$ *secure, defined in Definition 7, under the hardness of Discrete Logarithm problem and RMSS problem, as stated in Definition 1, in the AGM+ROM.*

On the dependence of assumptions on subset size. Our scheme allows any number of signers to form a subset of size k out of a universe of size n, to aggregatively sign the message. Some applications do restrict k to be within some limits, for example PoS blockchains that require 2/3 of the validators to sign. Here we discuss how our security assumptions depend on the relation between k, n, q and the security parameter λ. Let's say we allow k to range between 1 and an upper bound max_k. The case of the range $[n - max\_k, n]$ is symmetric.

- When the number of possible subsets is negligibly smaller than q, then the probability of existence of a subset sum solution is negligible. In this case, no subset attacks are possible and just the Discrete Log assumption is enough. This case arises if $\sum_{k=1}^{max\_k} \binom{n}{k}/q \leq 2^{-\lambda}$.
- In all other cases, we have to additionally assume RMSS, albeit it will also include the maximum subset size max_k as a parameter.

We do an analysis with some concrete numbers of practical relevance. We pick the group size q to be a 256-bit prime, and security parameter $\lambda = 128$.

[9] As we explain in Appendix C.2 the attack does not apply to [8].

In Figure 2, we plot the upper bound on subset size where the number of possible subsets is less than the tolerance level $q/2^{\lambda} \approx 2^{128}$. To be precise, we plot the value of `max_k_aggregated_sum` in the y-axis against n in the x-axis, such that:

$$\texttt{max\_k\_aggregated\_sum} = \max \left\{ max\_k \in [1, n] \ : \ \sum_{k=1}^{max\_k} \binom{n}{k} \leq 2^{128} \right\}$$

Observe that the plot climbs linearly till $n = 128$. This is expected as the size of the full set of subsets keeps below the threshold up to that point. When $n = 129$, the curve drops abruptly to $k = 64$. This is because, now the threshold is half the size of the full set of subsets, which is 2^{129}, and hence is realized at half the subset size. After this drop, the curve gradually slides down, reaching subset size ≤ 11 for $n = 10,000$. [10]

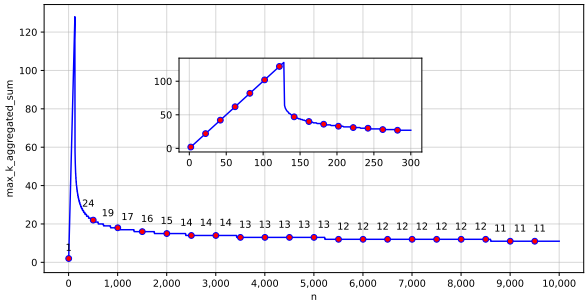

Fig. 2: Plots of upper bound on subset size where the number of possible subsets is less than the tolerance level. The main plot is for n going up to $10,000$. The inner plot is for n going up to 300 signers.

Concrete Attacks on RMSS We discuss the security of our scheme from both asymptotic and concrete perspectives.

Asymptotically, we assume $q = 2^{poly(\lambda)}$, and the number of parties is polynomial in the security parameter, that is, $n = (\log q)^{O(1)}$.

- At ranges where there is negligible probability of existence of an RMSS solution, the security of our scheme can solely depend on the DL assumption. In particular, this is true for $n < \log q - \lambda$, as there aren't enough subsets to assure the existence of an RMSS solution with non-negligible probability.
- For the range $\log q - \lambda < n = (\log q)^{O(1)}$, we don't know of any poly-time algorithms for RMSS. [43,30] provide some improvements, but still sub-exponential. Hence our scheme does not admit any known PPT attack.

Concretely, although the [43] n-sum algorithms (and the [16] attacks) are sub-exponential algorithms, their practical efficiency crucially affects the choice of real world parameters. In particular, [16] utilizes n lists of size $2^{[log(q)/(1+log(n))]}$ each with an algorithm that takes $n.2^{[log(q)/(1+log(n))]}$ time.

[10] `https://github.com/MystenLabs/research/tree/main/cryptography/bls_aggregation_combinatorics`

– Let's say we concretely allow up to 2^{128} time attacks and make the multi-sig verifier reject signature claims for more than $n = 2^{20}$ signers. In that case, taking q to be $108 \times 21 = 2268$ bits is sufficient. For Proof of Stake blockchains, like Ethereum, having a million signers is a reasonable assumption.[11]
– On the other hand if we take q to be 256 bits, like BLS12-381, we can only allow up to 128 signers. This is because the probability of existence of an RMSS solution would be $< 2^{-128}$ and hence depending on DL would be enough. This parameter range is sufficient for Aptos, Sui, Supra and so on, which use < 120 signers/validators.
– For larger n, where RMSS solutions exist with non-negligible probability then they can be found with the [43] protocol with time $< 2^{39}$. For instance, with $n = 200$, an RMSS solution exists with probability 2^{-56} and if one exists then can be found using [43] with time $< 2^{38}$ and non-negligible probability.

Conclusion. The flexibility in choosing subsets, after committing to a superset of public keys, prevented us from using the Boneh et al. [8] template for security proof. This prompted us to explore the AGM model, led us to discover the subset-sum attack, and mitigate that using the RMSS assumption.

6 Implementation and Evaluation

Implementation Details. We implement the scheme presented in Section 4 in Rust on top of the curve `bls12-381`. We provide two production-ready implementations[12], one where the signature is a group element of $\mathbb{G}_1$ and the public key is a group elements of $\mathbb{G}_2$ (noted as *minSig*), and a second where the signature is a group element of $\mathbb{G}_2$ and the public key is group elements of $\mathbb{G}_1$ (noted as *minPk*). Our implementations are built on top of the library `blst` [40] that provides base group operations over the curve `bls12-381`. We implement the randomization components of the scheme as a self-contained Randomize trait allowing to augment existing BLS implementations to support SMSKR with minimal modifications. As a result, supporting SMSKR only requires the addition of 50 LOC to define the Randomize trait and and extra 150 LOC to implement it.

Evaluation Results. We evaluate the performance of our production-ready SMSKR implementations described above. We perform our benchmarks on both a cheap Amazon Web Services (AWS) instance and a Macbook Pro equipped with an M1 processor. Our AWS experiments illustrate the performance of SMSKR on low-end devices. We select a *t3.medium* instance that comes with 2 virtual CPUs (1 physical core) on a 2.5 GHz Intel Xeon Platinum 8259 and 4GB of RAM. Our experiments on the Macbook Pro (presented in Appendix D) illustrate the performance of our scheme on high-end devices with powerful CPUs.

[11] current number of validators is 800000 https://beaconscan.com/statistics.
[12] `https://github.com/MystenLabs/fastcrypto`, (`6eb758ba78612e5e22a2748dd7a4b2c8b3724377`)

We select a Macbook Pro equipped with an M1 Pro and 16 GB of RAM. All our evaluations use Rust 1.65 and run with `cargo criterion` [4]. We open-source our benchmarking scripts to enable reproducible results. [13]

Our experiments aim to demonstrate the following claims. **(C1)** All functions of SMSKR are lightweight and performant even on low-end devices, **(C2)** SMSKR scales well when the number of signers increases, and **(C3)** SMSKR strictly outperforms the baseline scheme multi-signature of Boneh et al. [8] (and the performance benefit increases with the number of signers).

6.1 Microbenchmarks

Table 1 illustrates the performance of both our implementations (*minSig* and *minPk*) on a single CPU core. The implementation of SMSKR.SubsetKeyAggr is deeply embedded into the function SMSKR.Verify. We thus report the performance of both functions together in the last row of the table. All functions are evaluated for 100 signers, except SMSKR.Sign which is independent of the number of signers. We compute the average time over 100 runs.

The table shows that key generation is cheap, taking respectively about 250 and 180 μs on our low-end AWS instance and on our high-end M1 Macbook Pro. Signing is also cheap and can be performed in less than 500 μs even on our low-end machine. Aggregating 100 signatures is the cheapest operation taking only a few microseconds on any machine. Finally, verifying 100 signatures takes 1.39 ms on our low-end machine and half that time (0.72 ms) on our high-end machine. There is little difference between our *minSig* and *minPk* implementations when operating with 100 signers. Section 6.2 illustrates that the performance difference between these implementations increases rapidly with the number of signers.

The results of Table 1 illustrate that even the most expensive function: SMSKR.Verify, running on a low-end machine takes less than 2 ms. Thus, SMSKR is lightweight and performant even on low-end devices validating our claim **C1**.

6.2 Scalability

Figure 3 illustrate the performance of SMSKR when the number of signers increases. We omit SMSKR.KeyGen and SMSKR.Sign from our analysis since the former function is only ran once at setup and the latter is independent of the number of signers. Similarly to Section 6.1 the verification process (green lines) includes both SMSKR.SubsetKeyAggr and SMSKR.Verify. The signature aggregation process (blue lines) is simply a call to SMSKR.SigAggr.

Figure 3 shows the time required to aggregate and verify signatures on our low-end machine. As expected, the time required for aggregation and verification increases linearly with the number of public keys. The signature and key aggregation processes require one EC addition for each signature (and public key) to aggregate. This cost quickly dominates the cost of any other operation (including

[13] `https://github.com/MystenLabs/fastcrypto/blob/mskr-bench/fastcrypto/src/mskr_bench.rs`,(`4d1bad60b6db5bfbb448d98d89a72cfaebab6e56`)

Function	AWS		M1	
	minSig (ms)	*minPk* (ms)	*minSig* (ms)	*minPk* (ms)
KeyGen	0.25	0.25	0.18	0.18
Sign	0.44	0.44	0.31	0.31
SigAggr	0.06	0.17	0.05	0.11
Verify	1.39	1.46	0.72	0.75

Table 1: Micro-benchmark of the main SMSKR functions on a a low-end *t3.medium* AWS instance and a high-end Macbook Pro with a M1 CPU. Each data point represents the average time (over 100 runs) in milliseconds (ms) required to evaluate the function. All functions are evaluated for 100 public keys (except SMSKR.Sign that is independent of the number of public keys).

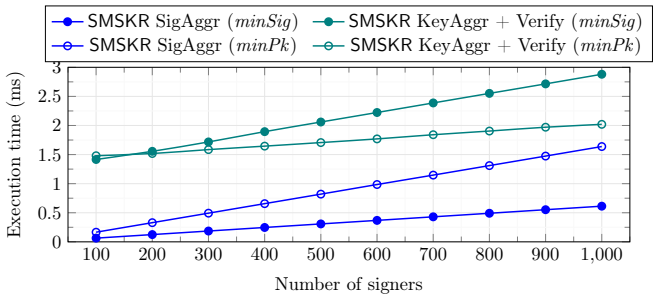

Fig. 3: Scalability of SMSKR on a low-end *t3.medium* AWS instance. Every data point on the graph is the average of 100 runs.

the single pairing check required by the verification process) as the number of signers increases. Both our *minSig* and *minPk* SMSKR implementations require less than 200 μs to aggregate signatures in a setting with less than 100 signers. SMSKR's signature aggregation scales well: our *minSig* and *minPk* implementations respectively require 300 μs and 800 μs to aggregate 500 signatures, and only 0.6 ms and 1.6 ms (respectively) to aggregate 1,000 signatures. We observe that our *minSig* signature aggregation implementation is faster than our *minPk* implementation: our *minSig* (resp. *minPk*) implementation represents signatures in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) and EC additions are faster in $\mathbb{G}_1$ than in $\mathbb{G}_2$. The SMSKR verification process (SMSKR.SubsetKeyAggr and SMSKR.Verify) also scales well. Both our *minSig* and *minPk* implementations take about 1.5ms to run with 100 signers and respectively require 3 ms and 2 ms to run the verification process with 1,000 signers (validating our scalability claim **C2**). Contrarily to signature aggregation, the verification process of our *minPk* implementation is faster than *minSig*. This is expected as our *minPk* implementation represents public keys in $\mathbb{G}_1$ (while *minSig* represents them in $\mathbb{G}_2$) where their aggregation is faster.

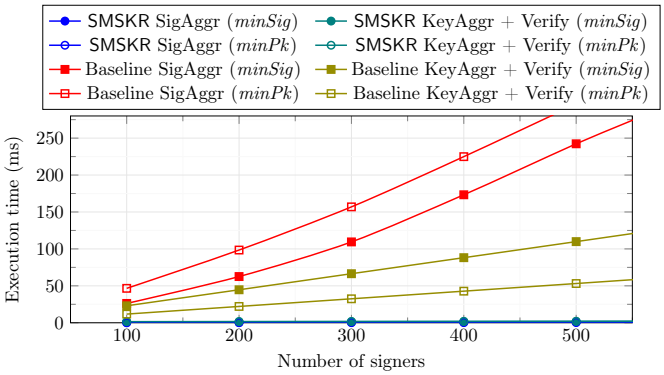

Fig. 4: Comparative performance of SMSKR with the baseline scheme of Boneh et al. [8] on a low-end *t3.medium* AWS instance. Every data point on the graph is the average of 100 runs.

Figure 7 in Appendix D shows the time required to aggregate and verify signatures on our high-end machine (thus better performance). In Appendix D we also provide a "zoomed" view on SMSKR's performance when the number of signers ranges from 10 to 100.

6.3 Baseline Comparison

Figure 4 compares the performance of SMSKR with the baseline scheme of Boneh et al. [8].[14] The figure shows that signatures aggregation of our SMSKR *minSig* and *minPk* implementations outperforms the baseline by two orders of magnitude, regardless of the number of signers. Our SMSKR *minSig* and *minPk* implementations respectively save 25 ms and 50 ms with respect to the baseline when aggregating 100 signatures, and a staggering 250 ms and 300 ms when aggregating 500 signatures. Similarly, the verification process of both SMSKR *minSig* and *minPk* implementations outperforms the baseline by respectively 50x and 30x. The baseline scheme of Boneh et al. [8] randomizes each signature before aggregation. Furthermore, it multiplies each signature and public key by a random exponent before their aggregation, thus paying the cost of one EC addition and one scalar multiplication for every signature and public key. This accounts for the performance differences with SMSKR that randomizes secret keys (upon setup) rather than individual signatures and entirely forgoes any scalar multiplication during signature aggregation.

Figure 8 in Appendix D compares the time required to aggregate and verify SMSKR signatures with the baseline on our high-end machine. The performance benefits are similar to the experiments on our low-end machine as the more powerful CPU scales performance roughly linearly. These figures validate our

[14] Note that in both Figure 4 and Figure 8 the performance results for all SMSKR operations collapse to a single line.

final claim **C3** by showing that SMSKR strictly outperforms the baseline and that the performance benefit increases linearly with the number of signers.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments. We also thank the a16z crypto team for reviewing the paper, recommending improvements, and providing future extension ideas. In particular, we thank Dan Boneh (Stanford University) and Valeria Nikolaenko (a16z crypto research). Mahdi Sedaghat was supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by CyberSecurity Research Flanders with reference number VR20192203.

References

1. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM CCS 2008: 15th Conference on Computer and Communications Security. pp. 449–458. ACM Press, Alexandria, Virginia, USA (Oct 27–31, 2008). https://doi.org/10.1145/1455770.1455827
2. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006: 13th Conference on Computer and Communications Security. pp. 390–399. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006). https://doi.org/10.1145/1180405.1180453
3. Benet, J., Greco, N.: Filecoin: A decentralized storage network. Protocol Labs (2018), https://filecoin.io/filecoin.pdf
4. bheisler: cargo-criterion. https://github.com/bheisler/cargo-criterion (2022)
5. Boldyreva, A.: Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. pp. 31–46. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-36288-6_3
6. Boldyreva, A., Gentry, C., O'Neill, A., Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) ACM CCS 2007: 14th Conference on Computer and Communications Security. pp. 276–285. ACM Press, Alexandria, Virginia, USA (Oct 28–31, 2007). https://doi.org/10.1145/1315245.1315280
7. Boneh, D., Drijvers, M., Neven, G.: Bls multi-signatures with public-key aggregation. https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html (2018)
8. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 435–464. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03329-3_15
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science, vol. 2248, pp. 514–532. Springer, Heidelberg, Germany, Gold Coast, Australia (Dec 9–13, 2001). https://doi.org/10.1007/3-540-45682-1_30

10. Brickell, E.F.: Solving low density Knapsacks. In: Chaum, D. (ed.) Advances in Cryptology – CRYPTO'83. pp. 25–37. Plenum Press, New York, USA, Santa Barbara, CA, USA (1983)
11. Coster, M.J., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P.: An improved low-density subset sum algorithm. In: Davies, D.W. (ed.) Advances in Cryptology – EUROCRYPT'91. Lecture Notes in Computer Science, vol. 547, pp. 54–67. Springer, Heidelberg, Germany, Brighton, UK (Apr 8–11, 1991). https://doi.org/10.1007/3-540-46416-6_4
12. Crites, E., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold Structure-Preserving Signatures. Cryptology ePrint Archive, Paper 2022/839 (2022), https://eprint.iacr.org/2022/839, to appear at Asiacrypt'23
13. Crites, E.C., Komlo, C., Maller, M.: How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. IACR Cryptol. ePrint Arch. p. 1375 (2021), https://eprint.iacr.org/2021/1375
14. Deirmentzoglou, E., Papakyriakopoulos, G., Patsakis, C.: A survey on long-range attacks for proof of stake protocols. IEEE Access **7**, 28712–28725 (2019). https://doi.org/10.1109/ACCESS.2019.2901858, https://doi.org/10.1109/ACCESS.2019.2901858
15. Drake, J.: Pragmatic signature aggregation with BLS - Sharding (May 2018), https://ethresear.ch/t/pragmatic-signature-aggregation-with-bls/2105
16. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy. pp. 1084–1101. IEEE Computer Society Press, San Francisco, CA, USA (May 19–23, 2019). https://doi.org/10.1109/SP.2019.00050
17. Edginton, B.: Upgrading Ethereum, https://eth2book.info/bellatrix/
18. El Bansarkhani, R., Sturm, J.: An efficient lattice-based multisignature scheme with applications to bitcoins. In: Foresti, S., Persiano, G. (eds.) CANS 16: 15th International Conference on Cryptology and Network Security. Lecture Notes in Computer Science, vol. 10052, pp. 140–155. Springer, Heidelberg, Germany, Milan, Italy (Nov 14–16, 2016). https://doi.org/10.1007/978-3-319-48965-0_9
19. Ethereum Core developers: Ethereum Proof-of-Stake Consensus Specifications, https://github.com/ethereum/consensus-specs
20. Frieze, A.M.: On the lagarias-odlyzko algorithm for the subset sum problem. SIAM Journal on Computing **15**(2), 536–539 (1986)
21. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_2
22. Fukumitsu, M., Hasegawa, S.: A lattice-based provably secure multisignature scheme in quantum random oracle model. In: Nguyen, K., Wu, W., Lam, K.Y., Wang, H. (eds.) ProvSec 2020: 14th International Conference on Provable Security. Lecture Notes in Computer Science, vol. 12505, pp. 45–64. Springer, Heidelberg, Germany, Singapore (Nov 29 – Dec 1, 2020). https://doi.org/10.1007/978-3-030-62576-4_3
23. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008). https://doi.org/https://doi.org/10.1016/j.dam.2007.12.010, applications of Algebra to Cryptography
24. Groth, J.: Non-interactive distributed key generation and key resharing (2021), https://eprint.iacr.org/2021/339, report Number: 339

25. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology **9**(4), 199–216 (Sep 1996). https://doi.org/10.1007/BF00189260
26. Itakura, K.: A public-key cryptosystem suitable for digital multisignatures (1983)
27. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. Journal of the ACM (JACM) **32**(1), 229–246 (1985)
28. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. IACR Cryptol. ePrint Arch. p. 374 (2022), https://eprint.iacr.org/2022/374
29. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 465–485. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_28
30. Lyubashevsky, V.: On random high density subset sums. Electron. Colloquium Comput. Complex. **TR05** (2005)
31. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001: 8th Conference on Computer and Communications Security. pp. 245–254. ACM Press, Philadelphia, PA, USA (Nov 5–8, 2001). https://doi.org/10.1145/501983.502017
32. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: Simple two-round Schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part I. Lecture Notes in Computer Science, vol. 12825, pp. 189–221. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84242-0_8
33. Nick, J., Ruffing, T., Seurin, Y., Wuille, P.: MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020: 27th Conference on Computer and Communications Security. pp. 1717–1731. ACM Press, Virtual Event, USA (Nov 9–13, 2020). https://doi.org/10.1145/3372297.3417236
34. Nicolosi, A., Krohn, M.N., Dodis, Y., Mazières, D.: Proactive two-party signatures for user authentication. In: ISOC Network and Distributed System Security Symposium – NDSS 2003. The Internet Society, San Diego, CA, USA (Feb 5–7, 2003)
35. Ohta, K., Okamoto, T.: A digital multisignature scheme based on the Fiat-Shamir scheme. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) Advances in Cryptology – ASIACRYPT'91. Lecture Notes in Computer Science, vol. 739, pp. 139–148. Springer, Heidelberg, Germany, Fujiyoshida, Japan (Nov 11–14, 1993). https://doi.org/10.1007/3-540-57332-1_11
36. Pan, J., Wagner, B.: Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 597–627. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_21, https://doi.org/10.1007/978-3-031-30589-4_21
37. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (Jun 2000). https://doi.org/10.1007/s001450010003
38. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) Advances in Cryptology

– EUROCRYPT 2007. Lecture Notes in Computer Science, vol. 4515, pp. 228–245. Springer, Heidelberg, Germany, Barcelona, Spain (May 20–24, 2007). https://doi.org/10.1007/978-3-540-72540-4_13
39. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 239–252. Springer New York, New York, NY (1990)
40. Supranational: blst. https://github.com/supranational/blst (2022)
41. Tessaro, S., Zhu, C.: Threshold and multi-signature schemes from linear hash functions. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 628–658. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_22, https://doi.org/10.1007/978-3-031-30589-4_22
42. Vesely, P., Gurkan, K., Straka, M., Gabizon, A., Jovanovic, P., Konstantopoulos, G., Oines, A., Olszewski, M., Tromer, E.: Plumo: An Ultralight Blockchain Client. In: Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers. pp. 597–614. Springer-Verlag, Berlin, Heidelberg (May 2022). https://doi.org/10.1007/978-3-031-18283-9_30, https://doi.org/10.1007/978-3-031-18283-9_30
43. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002). https://doi.org/10.1007/3-540-45708-9_19
44. zkcrypto: bls12_381. https://github.com/zkcrypto/bls12_381 (2022)

A BLS signatures in the Blockchain Space.

BLS signatures have recently seen an increased adoption in the blockchain space. Chia network adopted BLS[15] as its main signature scheme, primarily motivated by its non-interactiveness to generate threshold signatures. Instead of requiring multiple communication rounds and a dealer to ensure t-of-n participants had signed, a BLS aggregated signature can be incrementally aggregated. Celo developed a SNARK friendly BLS signature[16] on the BLS12-377 curve using a bespoke hash-to-curve function[17] to allow for efficient verification of multiple signatures. This benefited the Plumo ultralight client [42] to be more efficient where the signers do not need to know in advance about the public keys of the other signers.

Dfinity is designed based on a Random Beacon that acts as a verifiable random function[18] to produce verifiable and deterministic randomness with the threshold version of BLS signatures [24]. Each participant signs a message independently, and the aggregated signature itself serves as the deterministic random

[15] https://github.com/Chia-Network/bls-signatures.
[16] https://github.com/celo-org/celo-bls-snark-rs.
[17] https://github.com/celo-org/celo-proposals/blob/master/CIPs/cip-0022.md.
[18] https://dfinity.org/pdf-viewer/library/dfinity-consensus.pdf.

number that no party controls. Filecoin [3] uses BLS as one of the four signature schemes admissible for the blockchain's actors.

Although ECDSA signatures can be verified much faster individually, BLS signatures on the same message can be verified much faster in the aggregated form, therefore making it more practical for multiple validators attesting the same block or transaction. In particular, n aggregated signatures on the same message can be performed with just 2 pairings instead of $n+1$ [8]. Notably, the most recent Ethereum Consensus client deployed on the mainnet has adopted BLS signatures for validators to attest block proposals [19,17] addressing the verification bottleneck [15].

B Omitted Definitions and Preliminaries

B.1 Bilinear Pairings

Definition 8 (Bilinear pairing). *Given a security parameter λ, a bilinear group generator $\mathsf{BG}(1^\lambda)$ returns a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, q, g_1, g_2)$, where $\mathbb{G}_1, \mathbb{G}_2$ are groups of the same prime order q with the generators g_1, g_2, respectively. Also, let $\mathbb{Z}_q$ be the field of order q.A bilinear pairing is an* efficiently *computable map, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, satisfying the following properties:*

- ***Bilinearity:*** $\forall\ P \in \mathbb{G}_1, Q \in \mathbb{G}_2, a, b \in \mathbb{Z}_q$*:* $e(P^a, Q^b) = e(P^a, Q)^b = e(P, Q^b)^a = e(P, Q)^{ab}$.
- ***Non-degeneracy:*** $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$.

Bilinear pairings can be of a few types depending on whether there is an efficient isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ in both directions (Type 1), only one direction (Type 2), or in neither direction (Type 3) [23]. Type 3 pairings are the most efficient setting for a relevant security parameter and they are commonly deployed. In general, BLS and derived protocols work for all three types - so we ignore the distinctions in the following sections.

B.2 Forking Lemma

Pointcheval and Stern [37] first formalized the Forking Lemma which is used for bounding the success probability of reductions employing rewinding and reprogramming random oracles. In our proofs, we will adopt the more general version called General Forking Lemma, as described by Bellare and Neven [2].

Lemma 1 (General Forking Lemma [2]). *Fix an integer $q \geq 1$ and a set H of size $h \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that on input $x, h_1, ..., h_q$ returns a pair (idx, σ), the first element of which is an integer in the range $[0, q]$ and the second element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input parameter. The accepting probability of $\mathcal{A}$, denoted acc, is defined as the probability that $idx \geq 1$ in the experiment*

$$x \xleftarrow{\$} IG;\ h_1, ..., h_q \xleftarrow{\$} H;\ (idx, \sigma) \xleftarrow{\$} \mathcal{A}(x, h_1, ..., h_q)$$

Algorithm $F_{\mathcal{A}}(x)$

Pick coins ρ for $\mathcal{A}$ at random
$h_1, ..., h_q \xleftarrow{\$} H$
$(idx, \sigma) \xleftarrow{\$} \mathcal{A}(x, h_1, ..., h_q; \rho)$
if $(idx = 0)$:
    return $(0, \epsilon, \epsilon)$
$h'_I, ..., h'_q \xleftarrow{\$} H$
$(idx', \sigma') \xleftarrow{\$} \mathcal{A}(x, h_1, ..., h_{idx-1}, h'_{idx}, ..., h'_q; \rho)$
if $(idx = idx'$ and $h_{idx} \neq h'_{idx})$:
    return $(1, \sigma, \sigma')$
else : **return** $(0, \epsilon, \epsilon)$

Fig. 5: Forking Algorithm.

The forking algorithm $F_{\mathcal{A}}$ associated to $\mathcal{A}$, in Figure 5, is the randomized algorithm that takes input x and proceeds as follows:
Let

$$frk = Pr[b = 1 : x \xleftarrow{\$} IG; (b, \sigma, \sigma') \xleftarrow{\$} F_{\mathcal{A}}(x)] \ .$$

Then

$$frk \geq acc \cdot \left(\frac{acc}{q} - \frac{1}{h} \right) \ .$$

Alternatively,

$$acc \leq \frac{q}{h} + \sqrt{q \cdot frk} \ .$$

When we apply the forking lemma in our security proofs, we will assume an adversary breaking the unforgeability property of our signature and build from it an algorithm $\mathcal{A}$ that works under the assumptions of the forking lemma. The intuition is that $h_1, ..., h_q$ can be seen as the set of replies to the random oracle queries made by the original adversary. The forking adversary implements the rewinding and the two executions of $\mathcal{A}$ performed by $F_{\mathcal{A}}$ use the same random coins ρ.

B.3 Digital Signatures

Digital signatures are a widely used cryptographic primitive that serves as an electronic equivalent of a written signature. They ensure communication privacy, data integrity, message and sender authenticity, and sender non-repudiation. Next we formally define digital signatures and their security requirements.

Definition 9 (Digital Signature). *A digital signature scheme over message space $\mathcal{M}$ is a tuple of the following polynomial-time algorithms:*

- $\mathsf{pp} \leftarrow \mathbf{DS}.\mathsf{Setup}(1^\lambda)$: Setup is a probabilistic algorithm which takes as input the security parameter λ and outputs the set of public parameters pp.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathbf{DS}.\mathsf{KeyGen}(\mathsf{pp})$: Key generation is a probabilistic algorithm which takes as input pp and outputs a pair of signing/verification keys $(\mathsf{sk}, \mathsf{pk})$.
- $\sigma \leftarrow \mathbf{DS}.\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: The signing algorithm takes as input pp, a secret signing key sk, and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $0/1 \leftarrow \mathbf{DS}.\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: Verification is a deterministic algorithm which takes as input pp, a public verification key pk, a message $m \in \mathcal{M}$, and a purported signature σ, and outputs either 0 (reject) or 1 (accept).

Definition 10 (Correctness). *A digital signature is called correct, if we have:*

$$\Pr\left[\begin{array}{l}\forall\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathit{KeyGen}(\mathsf{pp}), m \in \mathcal{M} : \\ \mathit{Verify}(\mathsf{pp}, \mathsf{pk}, m, \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)) = 1\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Definition 11 (Unforgeability under Chosen Message Attack (UF-CMA)). *A digital signature scheme over message space $\mathcal{M}$ is UF-CMA secure if for all PPT adversaries $\mathcal{A}$ playing game $\mathit{UF\text{-}CMA}_{\mathcal{A}}$, as described in Figure 6, there exists a negligible function $\mathsf{negl}(\lambda)$ s.t. we have:*

$$Adv_{\mathcal{A}}^{\mathit{UF\text{-}CMA}}(\lambda) = \Pr\left[\mathbf{G}_{\mathcal{A}}^{\mathsf{UF\text{-}CMA}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ .$$

Game $\mathbf{G}_{\mathcal{A}}^{\text{UF-CMA}}(1^\lambda)$	Oracle $\mathcal{O}^{\mathsf{Sign}}(m)$
$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$	$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$	$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}(\cdot)}(\mathsf{pp}, \mathsf{pk})$	**return** σ
return $(m^* \notin \mathcal{Q} \ \wedge\ \mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m^*, \sigma^*))$	

Fig. 6: The UF-CMA security game.

B.4 Multi-Signature Schemes

We recall the definitions for multi-signatures by roughly following Bellare and Neven [2] and Drijvers et al. [16].

Definition 12 (Multi-Signature). *For a given security parameter λ, a multi-signature scheme with key aggregation consists of the following algorithms:*

- $\mathsf{pp} \leftarrow \mathbf{MS}.\mathsf{Setup}(1^\lambda)$: On input the security parameter λ in its unary representation, it outputs the scheme's parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathbf{MS}.\mathsf{KeyGen}(\mathsf{pp})$: Given the parameters pp, outputs a pair of public/secret keys $(\mathsf{pk}, \mathsf{sk})$.

– $\sigma_{\mathsf{PK}} \leftarrow \mathbf{MS}.\mathsf{Sign}(\mathsf{pp}, \mathsf{PK}, \{\mathsf{sk}_i\}_{i\in[1,n]}, m)$: On input the set of public keys PK, a signing secret key sk_i and a message m, the signer outputs the signature σ_i. A designated combiner outputs the combined signature σ_{PK}. (Instead of a designates combiner, this algorithm could be interactive.)
– $\mathsf{apk}_{\mathsf{PK}} \leftarrow \mathbf{MS}.\mathsf{KeyAggr}(\mathsf{pp}, \mathsf{PK})$: Given the public parameters pp and set of public keys PK as inputs. Output a single aggregated key $\mathsf{apk}_{\mathsf{PK}}$ for all the input pubic keys $\mathsf{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$.
– $0/1 \leftarrow \mathbf{MS}.\mathsf{Verify}(\mathsf{pp}, \mathsf{apk}_{\mathsf{PK}}, \sigma_{\mathsf{PK}}, m)$: Output 1 if the signature σ_{PK} verifies for message m under $\mathsf{apk}_{\mathsf{PK}}$, and 0 otherwise.

C Omitted Proofs

C.1 Security Analysis (for a weaker Adversary)

We start by noting that our suggested protocol change is not a simple modification of [8] as it requires a drastically different security proof. In [8] it is critical that the key randomization process happens at the same time as key aggregation as this allows the security reduction to handle all these hash queries as one which in turn allows to fix a specific set of public keys for the adversary's forgery even after rewinding.

To showcase the complication of the security analysis for our scheme, we first discuss the security of our scheme for a weaker unforgeability adversary which in the security game of Definition 7, defines a target subset I for which it will output its forgery before starting its queries. For this weaker case, our security analysis follows [8].

Theorem 3. *The proposed SMSKR is weakly unforgeable, i.e.* W-SMSKR-UF-CMA *secure, as defined in Definition 7, under the hardness of the computational co-Diffie-Hellman problem in the random-oracle model.*

Proof. Following the proof of [8], let $\mathcal{F}$ be a $(\tau, q_S, q_H, \epsilon)$ forger that breaks the unforgeability of SMSKR as defined in Definition 7.

Let *par* denote the bilinear group parameters and assume it is given as input everywhere below. Let IG be an algorithm that generates instances for the co-CDH problem, i.e. it outputs $(A, B_1, B_2) = (g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})$ for $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$.

We build an algorithm $\mathcal{A}$ that on input (A, B_1, B_2) proceeds as follows. $\mathcal{A}$ picks an index $k \xleftarrow{\$} \{1, \ldots, q_H\}$ and runs the forger $\mathcal{F}$ on input the honest public key $\mathsf{pk}_0 \leftarrow B_2$ with random tape ρ. $\mathcal{F}$ defines PK and its target forgery subset I.

H_0 queries. When $\mathcal{F}$ makes an H_0 query, $\mathcal{A}$ picks $r_i \xleftarrow{\$} \mathbb{Z}_q$ and returns $g_1^{r_i}$ for all i except the k'th query for which it returns A. We assume w.l.o.g. that $\mathcal{F}$ makes no repeated H_0 queries.

H_1 queries. When $\mathcal{F}$ queries H_1 on $(\mathsf{pk}_i, \mathsf{PK})$, $\mathcal{A}$ just returns a random value in $h_i \in \mathbb{Z}_q$.

Signing queries. The aggregated subset key apk^* is computed as in the actual protocol. When $\mathcal{F}$ makes a signing query on message m, $\mathcal{A}$ looks up $\mathsf{H}_0(m)$. If the lookup returns the value A, then $\mathcal{A}$ aborts. Else, the value must be of the form g_1^r, and $\mathcal{A}$ can simulate the honest signer by computing and returning $\sigma_i^* = B_1^r$. When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ aborts. If $\mathcal{F}$ successfully outputs a forgery for a message m for which $\mathsf{H}_0(m) \neq A$ then $\mathcal{A}$ also aborts. Otherwise, $\mathcal{F}$ outputs a forgery (σ^*, m^*, I) such that:

$$e(\sigma^*, g_2) = e(A, \mathbf{SMSKR}.\mathsf{SubsetKeyAggr}(\mathsf{pp}, \{\mathsf{pk}_j\}_{j \in I \cup \{0\}}))$$

Let j_f be the forgery index, i.e. the index for which $\mathcal{F}$ queried $\mathsf{H}_1(\mathsf{pk}_0, \mathsf{PK}) = h_{j_f}$. Let $a_j = \mathsf{H}_1(\mathsf{pk}_j, \mathsf{PK})$ for $\mathsf{PK} = \{\mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$.
Then $\mathcal{A}$ outputs $(j_f, \{\sigma, \mathsf{PK}, I, apk_I, a_1, \ldots, a_n\})$. The success probability of $\mathcal{A}$ is the probability that $\mathcal{F}$ succeeds and that $\mathcal{A}$ guessed the hash index of $\mathcal{F}$ forgery correctly. This happens with probability at least $1/q_{H_0}$, making $\mathcal{A}$'s overall success probability $\epsilon_{\mathcal{A}} = \epsilon/q_{H_0}$.

To complete the proof, we construct an algorithm $\mathcal{B}$ that, on input a co-CDH instance $(A, B_1, B_2) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$ and a forger $\mathcal{F}$, solves the co-CDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$. $\mathcal{B}$ will invoke the generalized forking algorithm GF_A (as defined in Lemma 1) on input (A, B_1, B_2) with the algorithm $\mathcal{A}$ running as described above. (Note that the co-CDH instance is distributed identically to the output of the IG). If GF_A aborts, then $\mathcal{B}$ outputs fail. If GF_A outputs (j_f, out^1, out^2), then $\mathcal{B}$ proceeds as follows: $\mathcal{B}$ parses the two outputs as: $out^1 = \{\sigma^1, \mathsf{PK}^1, I^1, apk_I^1, a_1^1, \ldots, a_n^1\}$ and $out^2 = \{\sigma^2, \mathsf{PK}^2, I^2, apk_I^2, a_1^2, \ldots, a_n^2\}$. By the forking lemma, we know that those two executions were identical up to the j_f'th H_1 query. In particular, this means that the arguments of the j_f'th query are identical, i.e., $\mathsf{PK}^1 = \mathsf{PK}^2$, $I^1 = I^2$, and $n^1 = n^2$. Let $h_{j_f^1} = a_i^1$ and $h_{j_f^2} = a_i^2$, then $a_i^1 \neq a_i^2$. Given that the two subsets are the same, we have $apk_I^1/apk_I^2 = \mathsf{pk}_0{}^{a_i^1 - a_i^2}$. Thus, $(\sigma^1/\sigma^2)^{1/(a_i^1 - a_i^2)}$ is a solution to the co-CDH instance. The probability of success can be bound with the forking lemma to be:

$$\epsilon' = q_{H_0}/q + \sqrt{q \cdot \epsilon / q_{H_0}}$$

Security Analysis (a loose reduction) As noted above, the proof of Theorem 3 would not go through if the adversary had not fixed the subset I for its forgery ahead of time (and before forking). If the adversary was allowed to output forgeries for different subsets before and after forking, then the security proof would only go through in the case that the same subset was used after rewinding. Given that the are 2^n possible subsets, this would imply an additional 2^n loss to the security reduction above:

$$\epsilon' = 2^n(q_{H_0}/q + \sqrt{q \cdot \epsilon / q_{H_0}})$$

resulting in a loose reduction.

The above reduction could potentially use the double-forking technique of [32] for a more tight proof but still with important security loss.

C.2 Proof of Theorem 1

Proof. Let $\mathcal{A}_{alg}$ be an algebraic adversary for the multi-signature unforgeability game defined in Definition 7. We build an adversary $\mathcal{A}_{DL}$, against the hardness of DL problem which uses $\mathcal{A}_{alg}$ as a subroutine.

Initialization. At the beginning, $\mathcal{A}_{DL}$ receives a Discrete Log challenge $(g, Z := g^z)$, where z is the desired discrete logarithm output. The challenger $\mathcal{A}_{DL}$ samples pk_0 uniformly from $\mathbb{G}$ in a couple of different ways as we will outline below. Briefly, we will describe two algorithms $\mathcal{B}$ and $\mathcal{C}$ that $\mathcal{A}_{DL}$ will call with probability $1/2$ each. Algorithm $\mathcal{B}$ will embed the challenge Z in the target public key pk_0, while algorithm $\mathcal{C}$ will embed Z in the hash $\mathsf{H}_0(m_i)$ query responses. After sampling pk_0 either way, $\mathcal{A}_{DL}$ sends it to $\mathcal{A}_{alg}$ as the public key of the target party. Regardless of which algorithm is executed, define $x \in \mathbb{Z}_q$ implicitly such that $\mathsf{pk}_0 = g^x$.

Query phase. The challenger $\mathcal{A}_{DL}$ simulates two main oracles, described as follows:

- H_0 **and** H_1 **queries** : the random oracles H_0 and H_1. Let $H_i = \mathsf{H}_0(m_i) = g^{r_i}$ denote the responses to the q_H hash H_0 queries. These are also sampled uniformly from $\mathbb{G}$ in different ways by algorithms $\mathcal{B}$ and $\mathcal{C}$: $\mathcal{B}$ samples r_i directly, while $\mathcal{C}$ embed Z in the responses. The random oracle H_1 is simulated by returning random elements from $\mathbb{Z}_q$.
- **Signing oracle,** $\mathcal{O}^{\mathsf{Sign}}(\cdot)$: The challenger $\mathcal{A}_{DL}$ also simulates signature queries in the following way. If the query is m_j, it first simulates computations of $\mathsf{H}_0(m_j)$ and $\mathsf{H}_1(\mathsf{pk}_j||\mathsf{PK})$ and then simulates and returns signature $\Sigma_j^{\dagger} = \mathsf{H}_0(m_j)^{x \cdot \mathsf{H}_1(\mathsf{pk}_j||\mathsf{PK})}$. The quantity $\Sigma_j = (\Sigma_j^{\dagger})^{\mathsf{H}_1(\mathsf{pk}_j||\mathsf{PK})^{-1}} = \mathsf{H}_0(m_j)^x$ can be publicly computed by querying H_1. While x is explicitly known to algorithm $\mathcal{C}$, it can be implicitly simulated by algorithm $\mathcal{B}$, as we will describe below. W.l.o.g, we also assume that $\mathcal{A}_{alg}$ queries H_0 with the target message m^*.

At some point, $\mathcal{A}_{alg}$ returns a set of keys $\mathsf{PK}_A = \{\mathsf{pk}_1, ..., \mathsf{pk}_n\}$. Let $\mathsf{PK} = \mathsf{PK}_A \cup \{\mathsf{pk}_0\}$. As it's an algebraic adversary, it also returns representations $\{(u_i, v_i, \vec{w_i})\}_i$ such that $\mathsf{pk}_i = g^{u_i} \mathsf{pk}_0^{v_i} \prod_{j=1}^{q_H} H_j^{w_{ij}}$ for all $i \in \{1, \ldots, n\}$. It's possible that some of the H_0 queries are sent after outputting PK_A - for those $\mathcal{A}_{alg}$ can set the w_{ij} exponents to 0.

Forgery phase. Finally, $\mathcal{A}_{alg}$ returns a forgery Σ^* on a message $m^* \notin Q$ and a set of indices $I \subseteq [1, n]$ together with a representation $\vec{a} = (\hat{a}, a', \bar{a}_1, ..., \bar{a}_{q_H}, \tilde{a}_1, ..., \tilde{a}_{q_S})$, consisting of $\mathbb{Z}_q$ elements, such that:

$$\Sigma^* = \mathsf{H}_0(m^*)^{x \cdot \mathsf{H}_1(\mathsf{pk}_0||\mathsf{PK}) + \sum_{i \in I} \mathsf{sk}_i \cdot \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})} = g^{\hat{a}} \mathsf{pk}_0^{a'} \prod_{i=1}^{q_H} H_i^{\bar{a}_i} \prod_{j=1}^{q_S} \Sigma_j^{\tilde{a}_j}$$

Here g is the generator of the group, pk_0 is the public key of the target party, $H_i = \mathsf{H}_0(m_i) = g^{r_i}$ are the responses to the q_H hash H_0 queries and $\Sigma_j =$

$\mathsf{H}_0(m_j)^x = g^{xr_j}$ are computed from the signatures $\Sigma_j^\dagger = \mathsf{H}_0(m_j)^{x \cdot \mathsf{H}_1(\mathsf{pk}_j||\mathsf{PK})}$ returned by the signing oracle. Let $\mathsf{H}_0(m^*) = g^{r^*}$. Let $h_i = \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})$, for $i \in [0, n]$. Implicitly, $\mathsf{sk}_i = u_i + \sum_{j=1}^{q_H} r_j w_{ij} + v_i x$, for all $i \in [1, n]$. This equation is equivalent to:

$$\left[xh_0 + \sum_{i \in I} \left(u_i + \sum_{j=1}^{q_H} r_j w_{ij} + v_i x \right) h_i \right] r^* = x \left(a' + \sum_{i=1}^{q_S} \tilde{a}_i r_i \right) + \left(\hat{a} + \sum_{i=1}^{q_H} \bar{a}_i r_i \right) .$$

Therefore,

$$x = \frac{(\hat{a} + \sum_{i=1}^{q_H} \bar{a}_i r_i) - \sum_{i \in I} (u_i + \sum_{j=1}^{q_H} r_j w_{ij}) h_i r^*}{(h_0 + \sum_{i \in I} v_i h_i) r^* - (a' + \sum_{i=1}^{q_S} \tilde{a}_i r_i)} . \tag{1}$$

Define $H = h_0 + \sum_{i \in I} v_i h_i$. We define events E and F, which will let different strategies succeed for the reduction. Let E be the event that $H = 0$, and let F be the event that $H \cdot r^* - (a' + \sum_{i=1}^{q_S} \tilde{a}_i r_i) = 0$. The challenger $\mathcal{A}_{DL}$ randomly chooses one of two algorithms $\mathcal{B}$ or $\mathcal{C}$ (described below) with probability $1/2$ and executes the chosen one.

Algorithm $\mathcal{B}$: The algorithm $\mathcal{B}$ sets $\mathsf{pk}_0 = Z$, the Discrete Log challenge. It can simulate a signature on a message m_i by setting $\Sigma_i^\dagger = Z^{r_i \cdot \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})}$, such that $\mathsf{H}_0(m_i) = g^{r_i}$. If event F occurs, then $\mathcal{B}$ aborts. If event $\neg F$ occurs, then it can compute $z = x$, by Equation (1), as the denominator is not 0. Therefore, $Adv_{DL}(B) = Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg}) \Pr[\neg F]$.

Algorithm $\mathcal{C}$: The algorithm $\mathcal{C}$ generates $\mathsf{pk}_0 = g^x$ by sampling x itself. It also generates the H_0 responses by sampling b_i and $\hat{r}_i$ and setting $\mathsf{H}_0(m_i) = g^{r_i} = Z^{b_i} g^{\hat{r}_i}$. If event $E \vee \neg F$ occurs, then $\mathcal{C}$ aborts. Otherwise, assume event $\neg E \wedge F$ occurs.
Given F, we get:

$$H \cdot (zb^* + \hat{r}^*) = H \cdot r^* = a' + \sum_{i=1}^{q_S} \tilde{a}_i r_i = a' + \sum_{i=1}^{q_S} \tilde{a}_i (\hat{r}_i + z b_i) .$$

Hence,

$$z = \frac{(a' + \sum_{i=1}^{q_S} \tilde{a}_i \hat{r}_i) - H \cdot \hat{r}^*}{H \cdot b^* - \sum_{i=1}^{q_S} \tilde{a}_i b_i} .$$

Note that the value of b^* is information-theoretically hidden from $\mathcal{A}_{alg}$ and is also absent from the sum $\sum_{i=1}^{q_S} \tilde{a}_i b_i$, as the forgery message mustn't have been queried to the signing oracle. Although the b_i's are also hidden to $\mathcal{A}_{alg}$, note that it could set all the $\tilde{a}_i$'s to 0 and hence force the sum $\sum_{i=1}^{q_S} \tilde{a}_i b_i$ to be 0. Given $\neg E$, H is non-zero, thus the denominator is w.h.p. $\neq 0$. Therefore, $Adv_{DL}(C) = Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg}) \cdot \Pr[\neg E \wedge F]$.

Event E: We show that the probability of this event is negligible given Discrete Log hardness. Let $\mathcal{A}_E$ be an adversary which wins if it makes event E

happen. We construct a challenger AE_{DL} which rewinds $\mathcal{A}_E$ and applies General Forking Lemma (Lemma 1) to break Discrete Log hardness.

We now describe the algorithm $\mathcal{A}_E$. It runs like algorithm $\mathcal{B}$ as described above in simulating the target public key, H_0, H_1, and signature queries, to the adversary $\mathcal{A}_{alg}$. Let $\mathcal{A}_{alg}$ return a set of public keys and representations $\mathsf{PK}_A, \{(u_i, v_i, \vec{w}_i)\}_{i=1}^n$ and produce a forgery $(\Sigma^*, m^*, I, \vec{a})$ as described before. If event E didn't happen, then $\mathcal{A}_E$ returns $(0, \epsilon)$. Otherwise, let u be the index of the first query of the form $(\mathsf{pk}_{i_u}||\mathsf{PK})$ to H_1, where $\mathsf{PK} = \mathsf{PK}_A \cup \{\mathsf{pk}_0\}$ and $\mathsf{pk}_{i_u} \in \mathsf{PK}$. Let $h_i = \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})$ for $i \in [0, n]$, and r_i be such that $\mathsf{H}_0(m_i) = g^{r_i}$ for $i \in [1, q_H]$. Then $\mathcal{A}_E$ returns $(u, (\mathsf{PK}, \{(u_i, v_i, \vec{w}_i)\}_{i=1}^n, \{r_i\}_{i=1}^{q_H}, \{h_i\}_{i=0}^n, I))$. Based on this, the General Forking Lemma algorithm $GF_{\mathcal{A}_E}$ returns:

$$(\mathsf{PK}, \{(u_i, v_i, \vec{w}_i)\}_{i=1}^n, \{r_i\}_{i=1}^{q_H}, \{h_i\}_{i=0}^n, I),$$
$$(\mathsf{PK}', \{(u'_i, v'_i, \vec{w}'_i)\}_{i=1}^n, \{r'_i\}_{i=1}^{q_H}, \{h'_i\}_{i=0}^n, I')$$

Since the u-th query is identical for the two executions, we must have $\mathsf{PK} = \mathsf{PK}'$. Also, by construction the sets $\{h_i\}_{i=0}^n$ and $\{h'_i\}_{i=0}^n$ in the two executions are independently random.

We first show that the probability that the vectors $(v_1, ..., v_n)$ and $(v'_1, ..., v'_n)$ are equal is negligible if $n = O(\log q)$.

Lemma 2. *For a given vector $\vec{v} = (v_1, ..., v_n) \in \mathbb{Z}_q^n$, the probability that for some $I \subseteq [1, n]$, $h_0 + \sum_{i \in I} h_i v_i = 0$ with $h_0, h_1, ..., h_n \leftarrow \mathbb{Z}_q$, is $< 2^n/q$.*

Proof. Let E_H denote the event that $\exists\ I \subseteq [1, n] : h_0 + \sum_{i \in I} h_i v_i = 0$. For any fixed I, the probability of E_H is $1/q$. Therefore, if we union bound the probabilities over all possible $I \subseteq [1, n]$, then the probability of E_H is at most $2^n/q$.

Since $(h'_0, ..., h'_1)$ are chosen independent of $(v_1, \ldots, v_n)$, we must have that with high probability (w.h.p.) $(v_1, \ldots, v_n) \neq (v'_1, ..., v'_n)$, by the above lemma. In that case there is an index k, such that $v'_k \neq v_k$. The Discrete Log challenger AE_{DL} then calculates the discrete log of pk_0 as $(u_k + \sum_{j=1}^{q_H} r_j w_{kj} - u'_k - \sum_{j=1}^{q_H} r'_j w'_{kj})/(v'_k - v_k)$.

Using Generalized Forking Lemma, we get:

$$Adv(\mathcal{A}_E) \leq q_H/q + \sqrt{q_H \cdot Adv(AE_{DL})/(1 - 2^n/q)}$$

Since the sample space of $\mathcal{A}_E$ matches that of $\mathcal{A}_{DL}$, therefore $\Pr[E] \leq q_H/q + \sqrt{q_H \cdot Adv(AE_{DL})/(1 - 2^n/q)}$

Putting everything together, we get that

$$Adv_{DL}(\mathcal{A}_{DL}) = 1/2(Adv_{DL}(\mathcal{B}) + Adv_{DL}(\mathcal{C}))$$

$$= 1/2 Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg})(\Pr[\neg F] + \Pr[\neg E \wedge F]) = 1/2 Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg})(1 - \Pr[E \wedge F])$$

$$\geq 1/2 Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg})(1 - \Pr[E])$$

Therefore,

$$Adv_{\mathsf{SMSKR}}(\mathcal{A}_{alg}) \leq 2Adv_{DL}(\mathcal{A}_{DL})(1 - \Pr[E])^{-1} \leq 2Adv_{DL}(\mathcal{A}_{DL})(1 + 2\Pr[E])$$

$$\leq 2Adv_{DL}(\mathcal{A}_{DL}) \left(1 + \frac{2q_H}{q} + 2\sqrt{\frac{q_H \cdot Adv(AE_{DL})}{(1 - 2^n/q)}}\right)$$

$$\leq 2Adv_{DL}(\mathcal{A}_{DL}) \left(1 + \frac{2q_H}{q} + 2\sqrt{q_H \cdot Adv(AE_{DL})}\left(1 + \frac{2^n}{q}\right)\right) \ .$$

Remark 3. Similar to [21], we assume symmetric bilinear groups in the proof. We note that the proof is extensible to asymmetric groups by assuming the hardness of discrete logarithm in both groups. In the reduction the unforgeability challenger can receive DL challenges in both groups and choose which challenge to embed depending on the context.

Is the 2^n security loss intrinsic? We argue that the 2^n security loss incurred in the above reduction is intrinsic, unless we resort to a hardness assumption related to random modular subset sums (RMSS), such as Definition 3. We describe a concrete attack.

Attack. An adversary which can efficiently solve RMSS instances can break the security of the system as we show here. Once this adversary $\mathcal{A}$ receives a target $\mathsf{pk}_0 = g^x$, it chooses $\{(u_i, v_i)\}_{i=1}^n$ randomly as $\mathbb{Z}_q$ elements and sends $\mathsf{PK}_A = \{\mathsf{pk}_i = g^{u_i}\mathsf{pk}_0^{v_i}\}_{i=1}^n$ to the challenger. Let $h_i = \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})$ for all $i \in [0, n]$, where $\mathsf{PK} = \mathsf{PK}_A \cup \{\mathsf{pk}_0\}$. The adversary solves the following RMSS instance:

- Target sum: $-h_0 \mod q$
- Set: $\{h_i \cdot v_i\}_{i=1}^n$

If $n = \Omega(\log q)$, w.h.p. there is a solution. Let's say the solution is $I \subseteq [1, n]$. This means $h_0 + \sum_{i \in I} h_i v_i = 0$. A SMSKR signature on a message m^* with subset $I \cup \{0\}$ is thus $\mathsf{H}_0(m^*)^{h_0 x + \sum_{i \in I} h_i(u_i + v_i x)} = \mathsf{H}_0(m^*)^{\sum_{i \in I} h_i u_i}$. This quantity can be readily computed by the adversary as it is independent of x.

Is the Boneh et al.'s multi-signature scheme immune from this attack? In the multi-signature scheme of [8] the hash is computed on the exact subset which is signing the multi-sig. Thus, the exact subset is committed in the random oracle exponent multipliers. There is no room to apply it to different subsets as is the case with SMSKR. Hence the above attack does not apply to [8].

C.3 Proof with RMSS assumption.

The above attack highlights the need to assume that random subset sum problems are hard to compute for an adversary. In fact, now we show that the RMSS (Definition 3) and discrete logarithm problems together suffice to prove security of the scheme without an exponential loss in reduction. We recall the Theorem statement from the main body.

Theorem 4. *The proposed SMSKR in Section 4 is* SMSKR-UF-CMA *secure, defined in Definition 7, under the hardness of Discrete Logarithm problem and RMSS problem, as stated in Definition 1, in the AGM+ROM.*

Proof. We only show that the probability of event E is bound by Discrete Log and RMSS hardness. The rest of the proof is same as the last one.

Let $\mathcal{A}_E$ be an adversary which wins if it makes event E happen. We construct a challenger AE_{DL} which selects randomly, with probability $1/2$ each, from two rewinding strategies $GF^{eq}_{\mathcal{A}_E}$ and $GF^{\neg eq}_{\mathcal{A}_E}$ and applies forking to break the hardness of the discrete logarithm problem.

We now describe the algorithm $\mathcal{A}_E$. It runs like algorithm $\mathcal{B}$ as described above in simulating the target public key, H_0, H_1, and signature queries, to the adversary $\mathcal{A}_{alg}$. Let $\mathcal{A}_{alg}$ return a set of public keys and representations $\mathsf{PK}_A, \{(u_i, v_i, \vec{w}_i)\}_{i=1}^n$ and produce a forgery $(\Sigma^*, m^*, I, \vec{a})$ as described before. If event E didn't happen, then $\mathcal{A}_E$ returns $(0, \epsilon)$. Otherwise, let u be the index of the first query of the form $(\mathsf{pk}_{i_u}||\mathsf{PK})$ to H_1, where $\mathsf{PK} = \mathsf{PK}_A \cup \{\mathsf{pk}_0\}$ and $\mathsf{pk}_{i_u} \in \mathsf{PK}$. Let $h_i = \mathsf{H}_1(\mathsf{pk}_i||\mathsf{PK})$ for $i \in [0, n]$. Then $\mathcal{A}_E$ returns $(u, (\mathsf{PK}, \{(u_i, v_i, \vec{w}_{ij})\}_{i=1}^n, \{r_i\}_{i=1}^{q_H}, \{h_i\}_{i=0}^n, I))$. Based on this, the algorithm $GF^{\neg eq}_{\mathcal{A}_E}$ returns:

$$1, (\mathsf{PK}, \{(u_i, v_i, \vec{w}_i)\}_{i=1}^n, \{r_i\}_{i=1}^{q_H}, \{h_i\}_{i=0}^n, I),$$

$$(\mathsf{PK}', \{(u'_i, v'_i, \vec{w}'_i)\}_{i=1}^n, \{r'_i\}_{i=1}^{q_H}, \{h'_i\}_{i=0}^n, I')$$

Since the u-th query is identical for the two executions, we must have $\mathsf{PK} = \mathsf{PK}'$. Let E_{eq} denote the event $\forall i \in [1, n] : v_i = v'_i$. If E_{eq} occurs, then AE_{DL} aborts. Otherwise, there is an index k, such that $v'_k \neq v_k$. The Discrete Log challenger AE_{DL} then calculates the discrete log of pk_0 as $(u_k + \sum_{j=1}^{q_H} r_j w_{kj} - u'_k - \sum_{j=1}^{q_H} r'_j w'_{kj})/(v'_k - v_k)$.

Algorithm $GF^{eq}_{\mathcal{A}_E}$ gets an RMSS challenge $(S = \{s_i\}_{i=1}^n, t)$. It sends $h_0, h_1, ..., h_n \leftarrow \mathbb{Z}_q$ as usual in the first execution and gets back $(v_1, ..., v_n)$. In the rewinded execution, it sends $h'_0 = -t, h'_1 = s_i v_i^{-1}, \ldots, h'_n = s_n v_n^{-1}$. Observe that this respects the distribution of the original game and is also independently random from $h_0, h_1, ..., h_n$. Now if event $\neg E_{eq}$ occurs, then $\mathcal{A}_{DL}$ aborts. Otherwise, we have $(v_1, ..., v_n) = (v'_1, ..., v'_n)$. Therefore, we have $-t + \sum_{i \in I'} v_i s_i v_i^{-1} = 0$. In other words, $t = \sum_{i \in I'} s_i$, and hence I' is a valid solution to the RMSS problem.

Summing up, we have:

$$\Pr[\mathcal{A}_E \text{ wins}] = 1/2(\Pr[\mathcal{A}_E \text{ wins } DL] + \Pr[\mathcal{A}_E \text{ wins } RMSS])$$

$$= 1/2\ (\Pr[GF^{\neg eq}_{\mathcal{A}_E} \text{ wins }] \cdot \Pr[\neg E_{eq}] + \Pr[GF^{eq}_{\mathcal{A}_E}] \cdot \Pr[E_{eq}])$$

Now, observe that,

$$\Pr[GF^{eq}_{\mathcal{A}_E} \text{ wins }] = \Pr[GF^{\neg eq}_{\mathcal{A}_E} \text{ wins }] \geq \Pr[\mathcal{A}_E \text{ wins }] \cdot (\Pr[\mathcal{A}_E \text{ wins }]/q_H - 1/q)$$

Therefore,

$$2\ \Pr[AE \text{ wins }] \geq \Pr[GF^{eq}_{\mathcal{A}_E}] \geq \Pr[\mathcal{A}_E \text{ wins }] \cdot (\Pr[\mathcal{A}_E \text{ wins }]/q_H - 1/q)$$

Therefore, following [2], we get:

$$\Pr[\mathcal{A}_E \text{ wins }] \leq q_H/q + \sqrt{q_H \cdot 2\Pr[AE \text{ wins }]}$$

$$= q_H/q + \sqrt{q_H \cdot (\Pr[AE \text{ wins } DL] + \Pr[AE \text{ wins } RMSS])}$$

D Additional Benchmarks

Beyond our two production-ready implementations, we also provide an additional minimal prototype implementation[19] for didactic proposes (and that we do not benchmark). Its scope is to illustrate the implementation of our scheme with clarity without the numerous performance optimizations of our production-ready implementations. This minimal prototype is build on top of the library `bls12_381` [44] providing base group operations over the curve `bls12-381`.

Below we describe some additional benchmarks of our production-ready implementations on a high-end device: a Macbook Pro equipped with an M1 Pro and 16 GB of RAM.

D.1 High-end devices benchmarks

Figure 7 shows the time required to aggregate and verify signatures on our high-end machine. The graphs are similar to Figure 3 but display better performance. Signature aggregation (1,000 signatures) takes respectively 0.5 ms and 1.3 ms for our *minSig* and *minPk* implementations. The verification process (1,000 signers) takes respectively 1.7 ms and 1.3 ms for our *minSig* and *minPk* implementations. Figure 3 and Figure 7 validate our scalability claim **C2**.

Figure 8 compares the time required to aggregate and verify SMSKR signatures with the baseline on our high-end machine. The performance benefits are similar to the experiments on our low-end machine as the more powerful CPU scales performance roughly linearly. For 500 signers our SMSKR *minSig* and *minPk* signature aggregation respectively save 150 ms and 180 ms; and our SMSKR *minSig* and *minPk* signature verification implementations respectively save around 75 ms and 40 ms with respect to the baseline.

D.2 Scalability

For completeness, Figure 9 and Figure 10 provide a "zoomed" view on SMSKR's performance when the number of signers ranges from 10 to 100. We observe that signature aggregation is only affected by a few microseconds and the verification time does not visibly change.

[19] Upon request, we can provide the code.

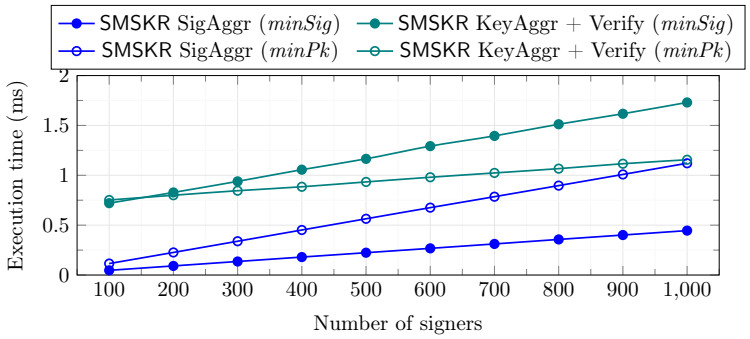

Fig. 7: Scalability of SMSKR on a high-end Macbook Pro equipped with a M1 processor. Every data point on the graph is the average of 100 runs.

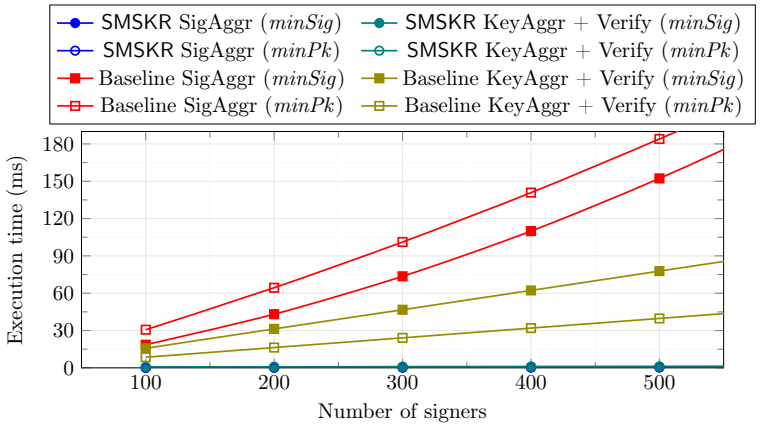

Fig. 8: Comparative performance of SMSKR with the baseline scheme of Boneh et al. [8] on a high-end Macbook Pro with a M1 processor. Every data point on the graph is the average of 100 runs.

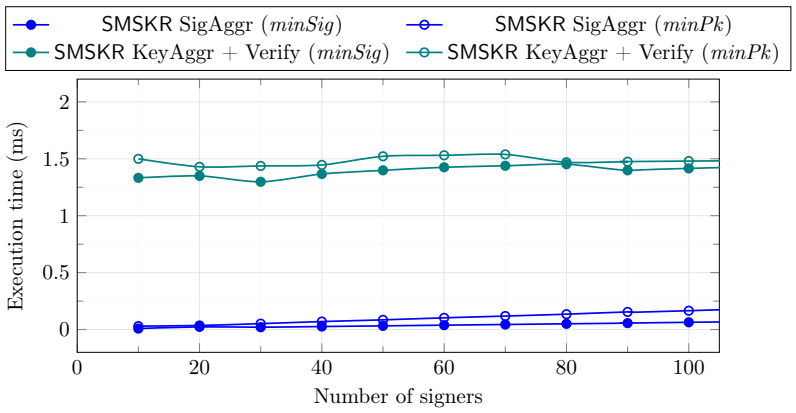

Fig. 9: Performance of SMSKR on a low-end *t3.medium* AWS instance. Every data point on the graph is the average of 100 runs.

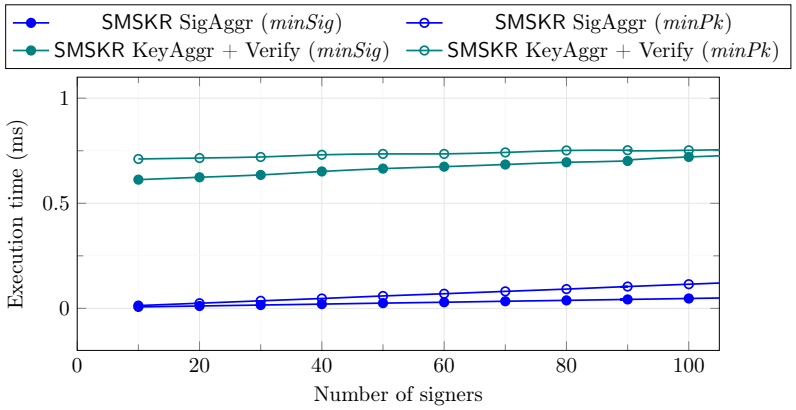

Fig. 10: Performance of SMSKR on a high-end Macbook Pro equipped with a M1 processor. Every data point on the graph is the average of 100 runs.