Robust Subgroup Multi-Signatures for Consensus

David Galindo[*†], Jia Liu[*]
[*]*Fetch.ai, UK*
[†]*University of Birmingham, UK*

***Abstract*—Multi-signatures are used to attest that a fixed collection of n parties, represented by their respective public keys, have all signed a given message. An emerging application of multi-signatures is to be found in consensus protocols to attest that a qualified subset of a global set of n validators have reached agreement. In this paper, we point out that the traditional security model for multi-signatures is insufficient for this new application, as it assumes that every party in the set participates in the multi-signature computation phase and is honest. None of these assumptions hold in the typical adversarial scenarios in consensus protocols (aka. byzantine agreement). We address this by introducing a new multi-signature variant called *robust subgroup multi-signatures*, whereby any eligible subgroup of signers from the global set can produce a multi-signature on behalf of the group, even in the presence of a byzantine adversary. We provide syntax and security definitions for the new variant. We argue that existing unforgeability security proofs for multi-signatures do not carry over to the consensus setting; a consequence of this observation is that many multi-signature based consensus protocols lack a rigorous security proof for correctness. To remedy this we propose several constructions which we prove secure under widely held cryptographic assumptions using our newly introduced formal definitions and also improve upon multi-signature computation time. Finally, we report on benchmarks from a proof-of-concept implementation.**

1. Introduction

A multi-signature protocol[1] enables n entities indexed by their respective public keys $\{pk_i\}_{i=1}^n$ to sign a message m and produce n individual signatures $\{s_i\}_{i=1}^n$ which are later compressed into a multi-signature σ. Recently there has been a renewed interest in multi-signatures for securing distributed ledgers, e.g. [DGNW20], [MPSW19], [GGJ+20]. Multi-signatures can be used to build secure ledgers more efficiently, namely they can help reduce block storage and/or block correctness verification time or reduce the time to validate that consensus on a block has been reached [BDN18], [DGNW20].

A multi-signature can be trivially obtained from any unforgeable signature scheme by concatenating every individuals' signature on m, while verification simply checks that each atomic signature passes the individual signature verification test. Often designs that improve on this trivial construction in terms of the final signature length or the performance of the verification process are preferred. It is expected that multi-signature designs achieve compactness [BDN18], that is, the length of the final multi-signature is independent of the number of signers. The verification of the short multi-signature σ should convince a verifier that *all* the n entities signed the message m. Multi-signatures can be constructed, for example, from Schnorr signatures e.g. [MPSW19], and from BLS signatures [BGLS03], [Bol02], [BDN18]. Schnorr-based multi-signatures involve multiple rounds of communication, while pairing-based multi-signature schemes can be non-interactive. The latter is highly desirable for applications in distributed settings where the potential participants are not known upfront and the network nodes are sparsely connected, making the establishment of multiple simultaneous point-to-point channels to support the interaction costly. The focus on this paper is therefore on non-interactive multi-signature protocols.

Despite the recent popularity gained by multi-signatures for the specific purpose of recording blockchain consensus, both in the literature [BDN18], [MPSW19], [DGNW20], [GGJ+20] and implemented in production platforms [eth], [Elr19], [Har], [Ten], [Cel], the existing multi-signatures formal models *fail to address* the particularities of consensus protocols. Indeed, existing multi-signatures formal models apply in scenarios where the signing entities are controlled by a single party or where they are highly coordinated, so that the incoming individual signatures are assumed to be valid before

1. We also use the term *scheme*.

being combined in a single multi-signature. Such is the case for instance in a scenario where a user owns multiple wallets that can transfer funds by generating a multi-signature that, if correct, shows every wallet signed the corresponding transfer (this actually constitutes their main usage so far in the cryptocurrency space).

Thus, existing (non-interactive) multi-signature schemes mechanics do not explicitly address the verification and selection of *valid* individual signatures before running the multi-signature creation process. However, in the context of signaling block agreement amongst a group of validators in Proof-of-Stake consensus protocols, it is not reasonable to assume that the individual signatures coming from consensus nodes are valid by default, as some of these nodes may be Byzantine [Shi19]. This applies likewise in the case of aggregating transaction signatures. Therefore individual signatures need to be verified before being aggregated into a multi-signature, as we simply cannot assume they are valid when the combining party runs the combining process. To reflect this, we introduce a notion of *robustness* for multi-signatures, which is a form of correctness but allowing the adversary to submit signatures to the signature combination algorithm. This lets an attacker launch signature poisoning attacks aimed at halting the success of the signature combination algorithm.

A variant of multi-signatures that can be particularly useful for consensus applications is subgroup multi-signatures [MOR01], [Bol02]. Subgroup multi-signatures allow any eligible subgroup of signers from a global set to produce a multi-signature on behalf of the global group. This concept is most useful in scenarios where the group of entities that will participate in the signing process of a given message belong to a fixed set but is not fully determined. For example that is the case in t-out-of-n multi-signatures, where a combined multi-signature is valid if any t members out of a given n-member set have signed.

The current practice of building a subgroup multi-signature by modifying BLS-based multi-signatures [BDN18] with bitmaps, as found for example in [Elr19], [Har], [Cod], [GGJ$^+$20] is *not necessarily* provably *secure*. Let us take the example of Elrond [Elr19], that uses a modified version of the multi-signature scheme MSP [BDN18]. Firstly, we note that no security analysis is provided for the resulting modified multi-signature scheme. The unforgeability of the original MSP scheme is proven conditioned to each signer having prior knowledge of all entities that will definitely sign, which makes MSP unsuitable to be used directly as a subgroup multi-signatures[2]. Technically this condition is crucial for defending against the notorious rogue-key attacks [RY07] and is guaranteed by hashing all the signers in the coefficients $a_i = H_1(\mathsf{pk}_i, \{\mathsf{pk}_1, \cdots, \mathsf{pk}_n\})$ during the signing process. In the security reduction of multi-signature unforgeability, these coefficients enable the generalised forking lemma to output two different multi-signatures signed by the same set of signers after rewinding to the point where a_i is chosen in order to extract the underlying secret key of the target signer $pk^\star$. Elrond's modified version breaks this condition by using the coefficients $a_i = H_1(\mathsf{pk}_i)$ instead, which can no longer guarantee the set of signers is the same when using the generalised forking lemma, thus it cannot be proven secure using the techniques in [BDN18]. Other in production consensus platforms like [Har], [Cod], [GGJ$^+$20] implement a bitmap variant of the proof-of-possession (PoP) based multi-signature MSP-pop [RY07]. However, MSP-pop with bitmap *has not been proven secure* as a subgroup multi-signature.

Our contributions. We introduce *robust subgroup multi-signature* protocols, a multi-signature variant specially designed to be used as a building block for consensus protocols. We define *robustness* for multi-signatures, and present a rigorous definition of *existential unforgeability* for subgroup multi-signature protocols. Robustness evaluates the correctness of the signature combination algorithm in the presence of an adversary that may control a subset of the signers. As a side benefit of explicitly capturing the signature combination function, we are able to optimise the validation of individual signatures by using zero-knowledge proofs instead of time-consuming pairing operations, which improves upon the efficiency of signature combination algorithms of several existing multi-signatures. Regarding existential unforgeability, we point out that the previous syntax and security definitions for subgroup multi-signatures in [Bol02] are incomplete and somewhat informal. This is undesirable from a scientific point of view, and in fact this lack of formality may explain why [Bol02] presents a security proof for their construction while it is known to be insecure [BDN18]. We believe that robust subgroup multi-signatures capture the variant of multi-signatures that best suits consensus applications.

2. Indeed, the main application for their multi-signatures discussed by the authors in [BDN18] is to Bitcoin n-of-n Multisig addresses

We shall present four constructions of robust subgroup multi-signatures, that are named $\mathsf{RSMSP}, \mathsf{RSMSP\text{-}zk}, \mathsf{RSMS\text{-}pop}$ and $\mathsf{RSMS\text{-}pop\text{-}zk}$ respectively. While $\mathsf{RSMS\text{-}pop}$ is a natural adaption of the MSP-pop scheme [RY07] to our subgroup model, the other three constructions $\mathsf{RSMSP}, \mathsf{RSMSP\text{-}zk}$ and $\mathsf{RSMS\text{-}pop\text{-}zk}$ are new. Our RSMSP scheme is a twist of the MSP scheme [BDN18]: the determination of the actual co-signers is deferred to the signature combination phase thanks to our explicit combine function, and we include the set of identities I of the actual co-signers in the hash values $a_i = H_2(\mathsf{pk}_i, I, \{\mathsf{pk}_1, \mathsf{pk}_2, \cdots, \mathsf{pk}_n\})$ for $i \in I$. This trick enables us to remove the restriction that the set of co-signers needed to be fixed upfront. It also enables us to use the Generalised Forking Lemma [BCJ08] to rewind the adversary to the point in the security reduction where the value a_i is generated and produce a different multi-signature forgery with the same set I of co-signers in order to prove existential unforgeability. The variants $\mathsf{RSMSP\text{-}zk}$ (resp. $\mathsf{RSMS\text{-}pop\text{-}zk}$) optimise the combination of individual signatures in RSMSP (resp. $\mathsf{RSMS\text{-}pop}$) by introducing NIZKs to validate individual signatures. Note that the combine function does not use any secret information and can be run by any entity, making it still a non-interactive process. Our experiments show that this optimisation makes the combination process 2x faster. Last but not least, by proving secure the construction $\mathsf{RSMS\text{-}pop}$ using our refined subgroup multi-signature model, we provide the missing evidence that the multi-signatures protocols for consensus in [Har], [Cod], [GGJ+20] *are indeed sound.*

Related work. Sequential aggregate multi-signatures are studied in [BGOY07], [LMRS04], [LOS+06] where each signer modifies the aggregate signature in turn. Such signatures are suitable for applications such as secure route attestation and certificate chains. Pixel [DGNW20] is a forward-secure multi-signature scheme based on hierarchical identity-based encryption (HIBE), which involves a key update method in order to provide forward security. Despite Pixel being used to record consensus on a block its security model only covers n-of-n multi-signatures. Plumo [GGJ+20] implements a SNARK based validation for checking aggregated BLS signatures, with the atomic subgroup multi-signatures being built with the above-referred MSP scheme, with the public keys corresponding to the signers subset are succinctly expressed as a bitmap.

Accountable subgroup multi-signatures [MOR01], [BDN18] allow any subgroups to produce a multi-signature for a message in a provably secure way but involve an interactive key setup per subgroup. This is mainly due to the requirement of not involving individual public keys for multi-signature verification. While this setting can be useful for some applications (e.g., multisig wallets with 2 or 3 addresses [And11]), it is not in general acceptable in the consensus setting. Indeed that would require validator nodes to run a joint setup phase which is expensive. In the decentralised consensus application, validator groups are formed ad hoc by the network nodes which are sparsely connected. An interactive key setup requires synchronisation among n participating nodes which is a difficult challenge for implementation of decentralised applications, especially when n is big. For example, Ethereum 2.0 requires a minimum of 128 validators for each committee. This requirement is the motivation for the multi-signature schemes developed in this paper. The individual public keys are used in the verification of our multi-signatures but they are only stored on the blockchain once when nodes register themselves as validators.

2. Preliminaries

In this section we briefly recall some building blocks needed in the rest of the paper, such as the computational assumptions under which our constructions are proven secure, the equality of discrete logarithms proof system and the generalised forking lemma.

2.1. Bilinear Groups

Definition 2.1 (Asymmetric Pairing Groups). *Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ be (cyclic) groups of prime order q. A map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to a group $\mathbb{G}_T$ is called a* bilinear *map, if it satisfies the following three properties:*

- *Bilinearity: $\mathbf{e}(g_1^x, g_2^y) = \mathbf{e}(g_1, g_2)^{xy}$ for all $x, y \in \mathbb{Z}_p$.*
- *Non-Degenerate: $\mathbf{e}(g_1, g_2) \neq 1$.*
- *Computable: $\mathbf{e}(g_1, g_2)$ can be efficiently computed.*

We assume there exists an efficient bilinear pairing instance generator algorithm $\mathcal{IG}$ that on input a security parameter 1^λ outputs the description of $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q \rangle$.

Asymmetric pairing groups can be efficiently generated [GJNB11]. Pairing group exponentiations and pairing operations can also be efficiently computed [DSD07].

2.2. Computational Assumptions

Definition 2.2 (Computational co-CDH assumption [BDN18]). *Let $X \leftarrow (\mathbb{G}_1, \mathbb{G}_2, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ where $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ are cyclic groups of prime order q, and $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^*$. We define the advantage $\mathbf{Adv}_\mathcal{A}^{\text{co-CDH}}$ of an adversary $\mathcal{A}$ as*

$$\mathbf{Adv}_\mathcal{A}^{\text{co-CDH}} := \Pr\left[\mathcal{A}(X) = g_1^{\alpha\beta}\right]$$

We say $\mathcal{A}$ (τ, ϵ)-breaks the co-CDH problem if it runs in time at most τ and $\mathbf{Adv}_\mathcal{A}^{\text{co-CDH}} \geq \epsilon$. co-CDH is (τ, ϵ)-hard if no such adversary exists.

Definition 2.3 (Computational ψ-co-CDH assumption [BDN18]). *Let $X \leftarrow (\mathbb{G}_1, \mathbb{G}_2, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ where $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ are cyclic groups of prime order q, and $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^*$. Let $\mathcal{O}^\psi(\cdot)$ be an oracle that on input $g_2^\alpha \in \mathbb{G}_2$ returns $g_1^\alpha \in \mathbb{G}_1$. We define the advantage $\mathbf{Adv}_\mathcal{A}^{\psi\text{-co-CDH}}$ of an adversary $\mathcal{A}$ as*

$$\mathbf{Adv}_\mathcal{A}^{\psi\text{-co-CDH}} := \Pr\left[\mathcal{A}^{\mathcal{O}^\psi(\cdot)}(X) = g_1^{\alpha\beta}\right]$$

We say $\mathcal{A}$ (τ, ϵ)-breaks the ψ-co-CDH problem if it runs in time at most τ and $\mathbf{Adv}_\mathcal{A}^{\psi\text{-co-CDH}} \geq \epsilon$. ψ-co-CDH is (τ, ϵ)-hard if no such adversary exists.

2.3. Equality of Discrete Logarithms

We need NIZK proof systems as an ingredient to our construction, namely the *Equality of Discrete Logarithms* proof system. Formally, given a cyclic group $\mathbb{G}$ of order q and $g, h \in \mathbb{G}$, the NIZK proof $(\mathsf{PrEq}_H, \mathsf{VerEq}_H)$ to show $k = \log_g x = \log_h y$ for $x, y \in \mathbb{G}$, $k \in \mathbb{Z}_q$ is described as below [CP93]:

- $\mathsf{PrEq}_H(g, h, x, y, k)$: choose $r \xleftarrow{\$} \mathbb{Z}_q$, compute $R_1 = g^r, R_2 = h^r$ and set $c \leftarrow H(g, h, x, y, R_1, R_2)$. Output is $(c, s = r + k \cdot c)$.
- $\mathsf{VerEq}_H(g, h, x, y, (c, s))$: compute $R_1 \leftarrow g^s/x^c$ and $R_2 \leftarrow h^s/y^c$ and output $c \overset{?}{=} H(g, h, x, y, R_1, R_2)$.

2.4. Generalized Forking Lemma

The forking lemma [PS00] for proving the security of schemes based on Schnorr signatures was generalised to a wider class of schemes [BN06], [BCJ08]. Below we describe the version due to [BCJ08].

Consider an algorithm $\mathcal{A}$ that on input *in* interacts with a random oracle $H : \{0,1\}^* \mapsto \mathbb{Z}_q$.

```
$\underline{\mathcal{GF}_\mathcal{A}(in):}$
$f = (\rho, h_1, \ldots, h_{q_H}) \xleftarrow{\$} \omega$
$(J, \{out_j\}_{j \in J}) \leftarrow \mathcal{A}(in, f)$
If $J = \emptyset$ then output fail
Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \leq \cdots \leq j_n$
For $i = 1, \ldots, n$ do
  $succ_i \leftarrow 0; k_i \leftarrow 0; k_{\max} \leftarrow 8nq_H/\epsilon \cdot \ln(8n/\epsilon)$
  Repeat until $succ_i = 1$ or $k_i > k_{\max}$
    $f'' \xleftarrow{\$} \Omega$ such that $f'|_{j_i} = f|_{j_i}$
    Let $f'' = (\rho, h_1, \ldots, h_{j_i-1}, h''_{j_i}, \ldots, h''_{q_H})$
    $(J'', \{out''_j\}_{j \in J''}) \leftarrow \mathcal{A}(in, f'')$
    If $h''_{j_i} \neq h_{j_i}$ and $J'' \neq \emptyset$ and $j_i \in J''$ then
      $out'_{j_i} \leftarrow out''_{j_i}; succ_i \leftarrow 1$
If $succ_i = 1$ for all $i = 1, \ldots, n$
  Then output $(J, \{out_j\}_{j \in J}, \{out'_j\}_{j \in J})$
  Else output fail
```

Figure 1: Algorithm $\mathcal{GF}$

Let $f = (\rho, h_1, \ldots, h_{q_H})$ be the randomness involved in an execution of $\mathcal{A}$, where ρ is $\mathcal{A}$'s random tape, h_i is the response to $\mathcal{A}$'s i-th query to H, and q_H is its maximal number of random-oracle queries. Let Ω be the space of all such vectors f and let $f|_i = (\rho, h_1, \ldots, h_{i-1})$. We consider an execution of $\mathcal{A}$ on input *in* and randomness f, denoted by $\mathcal{A}(in, f)$, as successful if it outputs a pair $(J, \{out_j\}_{j \in J})$, where J is a multi-set that is a non-empty subset of $\{1, \ldots, q_H\}$ and $\{out_j\}_{j \in J}$ is multi-set of side outputs. We say that $\mathcal{A}$ failed if it outputs $J = \emptyset$. Let p be the probability that $\mathcal{A}(in, f)$ is successful for fresh randomness $f \xleftarrow{\$} \Omega$ and for an input $in \xleftarrow{\$} \mathcal{IG}$ generated an input generator $\mathcal{IG}$.

For a given input *in*, the generalised forking algorithm $\mathcal{GF}$ is defined as shown in Figure 1. We say that $\mathcal{GF}_\mathcal{A}$ succeeds if it doesn't output `fail`.

Lemma 2.1 (Generalised Forking Lemma [BCJ08]). *Let $\mathcal{IG}$ be a randomised algorithm and $\mathcal{A}$ be a randomised algorithm running in time τ making at most q_H random-oracle queries that succeeds with probability ϵ. If $q > 8nq_H/\epsilon$, then $\mathcal{GF}_\mathcal{A}(in)$ runs in time at most $\tau \cdot 8n^2 q_H/\epsilon \cdot \ln(8n/\epsilon)$ and succeeds with probability at least $\epsilon/8$, where the probability is over the choice of $in \xleftarrow{\$} \mathcal{IG}$ and over the coins of $\mathcal{GF}_\mathcal{A}$.*

3. Definitions of Robust Subgroup Multi-signatures

We now introduce syntax and security definitions for our robust subgroup multi-signatures. We describe a notion of *robustness*, a generalised notion of correctness, by allowing adversaries to participate in the signature combination process. Then we define unforgeability in a subgroup model where the adversary outputs a forgery $(J, \sigma^\star)$ where J is a subset of indices of signers from a group $\mathcal{PK}$ chosen by the adversary and $\sigma^\star$ is a multi-signature.

Definition 3.1. *A* robust subgroup multi-signature *scheme (*RSMS*)* $\Pi = (\mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{GroupSet}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifyMul})$ *consists of the following algorithms:*

$\mathsf{KeyGen}(1^\lambda)$**:** *on input a security parameter* λ*, this algorithm outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$ *for each entity where the public key* pk *is made public.*

$\mathsf{GroupSet}(\mathcal{PK})$**:** *on input a set of public keys* $\mathcal{PK}$*, the algorithm forms a group and outputs a group public key* $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK})$ *with* gtag *a unique group tag. When the algorithm fails to form the group,* $\mathsf{gpk} = \perp$.

$\mathsf{KeyAgg}(J, \mathcal{PK})$**:** *on input an index set* J *and a set of public keys* $\mathcal{PK}$*, this algorithm aggregates* $\mathcal{PK}$ *into a single aggregate public key* apk. *Output* apk.

$\mathsf{Sign}(m, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{gtag})$**:** *on input a message* m*, a secret key* sk_i*, a public key* pk_i*, a group tag* gtag*, the algorithm outputs an individual signature* s_i.

$\mathsf{Combine}(m, \mathcal{E}, \mathsf{gpk})$**:** *on input a message, a set of individual signatures* $\mathcal{E} = \{s_i\}_i$ *originating from* $|\mathcal{E}|$ *different entities, a group public key* gpk*, the algorithm identifies an index set* J *of valid individual signatures and combines them into a single multi-signature* σ *and outputs* (J, σ)*. When the algorithm fails to combine the individual signatures,* $(J, \sigma) = (\emptyset, \perp)$.

$\mathsf{VerifyMul}(m, J, \sigma, \mathsf{gpk})$**:** *on input a message* m*, an index set* J*, a group public key* gpk*, the algorithm outputs 1 if* σ *is a valid multi-signature and outputs 0 otherwise.*

Definition 3.2 (Robustness)**.** *The* robustness *of a* RSMS *scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{GroupSet}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifyMul})$ *is defined by a three-stage game:*

Setup *The challenger generates the system parameters* pp *and a challenge entity with key tuple* $(\mathsf{sk}^\star, \mathsf{pk}^\star) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. It gives* $(\mathsf{pk}^\star, \mathsf{pp})$ *to the adversary.*

Signature queries $\mathcal{A}$ *is allowed to make signature queries on any message* m *for any* gtag*, meaning that it has access to oracle* $\mathcal{O}^{\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)}$ *that simulates the honest signer signing a message* m.

Output *Finally, the challenger receives from the adversary a message* $m^\star$ *and a group public key* $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK} = \{\mathsf{pk}_i\}_{i \in U})$*, and a set of individual signatures* $\mathcal{E} = \{s_i\}_{i \in I}$ *from* $|I|$ *different entities such that* $I \subseteq U$*. The adversary wins if*

1) $\mathsf{pk}^\star = \mathsf{pk}_k$ *for some* $k \in U$ *and* $k \notin I$
2) $\mathsf{gpk} \neq \perp$ *and* $\mathsf{gpk} = \mathsf{GroupSet}(\mathcal{PK})$
3) $\mathsf{VerifyMul}(m^\star, J, \sigma^\star, \mathsf{gpk}) = 0$ *where* $s^\star \leftarrow \mathsf{Sign}(m^\star, \mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{gtag})$ *and* $(J, \sigma^\star) \leftarrow \mathsf{Combine}(m^\star, \mathcal{E} \cup \{s^\star\}, \mathsf{gpk})$ *and* $J \backslash \{k\} \neq \emptyset$

We say $\mathcal{A}$ $(\tau, q_S, q_H, \epsilon)$*-breaks the robustness of* RSMS *if* $\mathcal{A}$ *runs in time at most* τ*, makes at most* q_S *signing queries and at most* q_H *random oracle queries, and the above game outputs 1 with probability at least* ϵ. RSMS *is* $(\tau, q_S, q_H, \epsilon)$*-robust if no such adversary exists.*

Definition 3.3 (Unforgeability)**.** *The* unforgeability *of a* RSMS *scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{GroupSet}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifyMul})$ *is defined by a three-stage game:*

Setup *The challenger generates the system parameters* pp *and a challenge key pair* $(\mathsf{sk}^\star, \mathsf{pk}^\star) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. It gives* $(\mathsf{pk}^\star, \mathsf{pp})$ *to the adversary.*

Signature queries $\mathcal{A}$ *is allowed to make signature queries on any message* m *for any group tag* gtag*, meaning that it has access to oracle* $\mathcal{O}^{\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)}$ *that simulates the honest signer signing a message* m.

Output *Finally, the adversary outputs a multi-signature forgery* $(J, \sigma^\star)$*, a message* $m^\star$*, a group public key* $\mathsf{gpk} = (\mathsf{gtag}^\star, \mathcal{PK} = \{\mathsf{pk}_i\}_{i \in U})$*. The adversary wins if*

1) $\mathsf{gpk} \neq \perp$ *and* $\mathsf{gpk} = \mathsf{GroupSet}(\mathcal{PK})$
2) $\mathsf{pk}^\star = \mathsf{pk}_k$ *for some* $k \in U \cap J$
3) $\mathcal{A}$ *made no signing queries on* $(m^\star, \mathsf{gtag}^\star)$ *and* $\mathsf{VerifyMul}(m^\star, J, \sigma^\star, \mathsf{gpk}) = 1$

We say $\mathcal{A}$ *is a* $(\tau, q_S, q_H, \epsilon)$*-forger for* RSMS *if* $\mathcal{A}$ *runs in time at most* τ*, makes at most* q_S *signing queries and at most* q_H *random oracle queries, and wins the above game with probability at least* ϵ. RSMS *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if no such adversary exists.*

4. Our pairing-based robust-subgroup multi-signature scheme

Generally speaking, the major technical problem in constructing a secure multi-signature scheme is to prevent the rogue-key attack [RY07], [MPSW19] which enables an adversary to use a specially crafted public key to deprive all other entities of their signing right. We consider two ways to prevent rogue-key attacks for our robust subgroup multi-signatures: one is using techniques proposed in [MPSW19], [BDN18] to combine individual signatures with hash values derived from the related public keys; the other way is called the proof-of-possession [RY07], [BDN18] which introduces an additional public key $H(\mathsf{pk}_i)^{\mathsf{sk}_i}$ to show the knowledge of the secret key sk_i. Moreover, we optimise the Combine process by introducing NIZKs to validate individual signatures to replace the checking based on time-consuming pairing equations. This optimisation can make the signature combination process x2 faster.

In this section, we present four constructions of robust-subgroup multi-signatures: RSMSP and its optimised version RSMSP-zk, and RSMS-pop and its optimised version RSMS-pop-zk.

4.1. Construction of RSMSP

Our RSMSP is constructed based on MSP scheme proposed in [BDN18]. As mentioned in the introduction, MSP in [BDN18] cannot be used as a subgroup multi-signature scheme in the provably secure sense. Here we propose a twist to MSP in order to construct a secure robust-subgroup multi-signature: we defer the determination of the actual co-signers to the combine phase thanks to the explicit introduction of the Combine function in our robust subgroup multi-signature scheme, and we include the identities of the subset J of co-signers in the hash values $H_2(\mathsf{pk}_j, J, \mathcal{PK})$ generated for each co-signer $j \in J$. In the security proof, this twist allows us to determine the co-signers at the moment when the hash value $c_{j_f} = H_2(\mathsf{pk}^\star, J, \mathcal{PK})$ is generated so that we can use the generalised forking lemma to rewind the adversary to this point and produce another multi-signature with the same set of cosigners J in order to extract the underlying secret key $\mathsf{sk}^\star$ of $\mathsf{pk}^\star$. Note that the Combine function does not use any secret information and can be run by any entity, making it still a non-interactive process.

Construction of RSMSP**.** Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be a bilinear group with prime order q and $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$. Assume hash functions $H_1 : \{0,1\}^* \mapsto \mathbb{Z}_q$, $H_2 : \{0,1\}^* \mapsto \mathbb{G}_1$ and $H_3 : \{0,1\}^* \mapsto \mathbb{G}_1$.

- $\mathsf{KeyGen}(1^\lambda)$**:** Choose $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$, compute $\mathsf{pk} = g_2^{\mathsf{sk}}$, and output $(\mathsf{sk}, \mathsf{pk})$. The public key pk is made public.
- $\mathsf{GroupSet}(\mathcal{PK})$**:** Compute $\mathsf{gtag} = H_1(\mathcal{PK})$. Output $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK})$.
- $\mathsf{KeyAgg}(J, \mathcal{PK})$**:** Parse $\mathcal{PK} = \{\mathsf{pk}_i\}_{i \in I}$. Output $\perp$ if $J \not\subseteq I$. Compute $a_j = H_2(\mathsf{pk}_j, J, \mathcal{PK})$ for each $j \in J$. Output $\mathsf{apk} = \prod_{j \in J} \mathsf{pk}_j^{a_j}$.
- $\mathsf{Sign}(m, \mathsf{sk}_i, \mathsf{gtag})$**:** Output an individual signature $s_i = H_3(\mathsf{gtag}, m)^{\mathsf{sk}_i}$.
- $\mathsf{Combine}(m, \mathcal{E}, \mathsf{gpk})$**:** Parse $\mathcal{E} = \{s_i\}_{i \in I}$ of individual signatures originating from $|I|$ different entities, and $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK})$. The combiner verifies each signature by checking $e(s_i, g_2) = e(H_3(\mathsf{gtag}, m), \mathsf{pk}_i)$. Assume the index set of valid signatures is $J \subseteq I$. The combiner computes the multi-signature as $\sigma = \prod_{j \in J} s_j^{a_j}$ with $a_j = H_2(\mathsf{pk}_j, J, \mathcal{PK})$ for $j \in J$. Output (J, σ).
- $\mathsf{VerifyMul}(m, J, \sigma, \mathsf{gpk})$**:** Parse $\mathsf{gpk} = (\mathsf{gtag}, \{\mathsf{pk}_i\}_{i \in I})$. If $J = \emptyset$ or $J \not\subseteq I$, then output 0. Otherwise, compute $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(J, \mathcal{PK})$. Output 1 if $e(\sigma, g_2) = e(H_3(\mathsf{gtag}, m), \mathsf{apk})$; else output 0.

Remark 1. *Note that when hashing the message m in the signing algorithm, it is a good practice to include the unique group tag* gtag *in the hash function $H_1(\mathsf{gtag}, m)$ since the same node may be able to join different committee groups at the same time using the same public key (e.g., [HMW18]). In this way, in the security definition of unforgeability, the adversary is allowed to issue signing query on different combinations $(m^\star, \cdot)$ and $(\cdot, \mathsf{gtag}^\star)$ of the challenge message and the challenge group tag as long as it does not query $(m^\star, \mathsf{gtag}^\star)$.*

Batch verification. As usual, n multi-signatures on n-distinct pairs of (gtag, m) can be verified as a batch faster than verifying them one by one:

- Compute an aggregate multi-signature $\Sigma = \sigma_1 \cdots \sigma_n \in \mathbb{G}_1$
- Accept all n multisignatures as valid iff

$$e(\Sigma_n, g_2) = \prod_{1 \le i \le n} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$$

Theorem 4.1. RSMSP *is robust.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that $(\tau, q_S, q_H, \epsilon)$-breaks robustness of RSMSP. We will show that this leads to a contradiction. At the end of the robustness game, the adversary outputs $(m^\star, \mathsf{gpk} = (\mathsf{gtag}^\star, \mathcal{PK} = \{\mathsf{pk}_i\}_{i \in U}), \mathcal{E} = \{s_i\}_{i \in I})$ with $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $k \in U$. When the challenger outputs 1, we know $e(\sigma^\star, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{apk})$ where $s \leftarrow \mathsf{Sign}(m^\star, \mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{gtag}^\star)$, $(J, \sigma^\star) \leftarrow \mathsf{Combine}(m^\star, \mathcal{E} \cup \{s\}, \mathsf{gpk})$ and $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(J, \mathcal{PK})$. This means there exists $j \in J$ such that $e(s_j, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{pk}_j)$. We consider two cases. If $j = k$, then this is impossible because $s_j = H_3(\mathsf{gtag}^\star, m^\star)^{\mathsf{sk}^\star}$ and $\mathsf{pk}_j = g_2^{\mathsf{sk}^\star}$.

If $j \neq k$, this directly contradicts to the verification $e(s_j, g_2) = e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{pk}_j)$ performed in the Combine process. This completes the proof. □

Theorem 4.2. RSMSP *is unforgeable under the computational co-CDH assumption in the random oracle model. Formally,* RSMSP *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* $q > 8q_H/\epsilon$ *and co-CDH is* $((\tau + (q_H + q_S) \cdot \tau_{\exp_1} + \ell \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon), \epsilon/(8q_H))$*-hard where* ℓ *is the maximum number of signers involved in a single multi-signature and* $\tau_{\exp_1}$ *is the time required to compute one exponentiation in* $\mathbb{G}_1$.

Proof. Suppose $\mathcal{A}$ is a $(\tau, q_S, q_H, \epsilon)$ forger against RSMSP. We shall first construct an adversary $\mathcal{B}$ using $\mathcal{A}$ as a subroutine and then construct another adversary $\mathcal{F}$ to run $\mathcal{GF}_\mathcal{B}$ to derive a solution to the co-CDH instance.

Given a co-CDH instance $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\beta)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. The goal is to find the solution $g_1^{\alpha\beta}$ to the instance. We first construct $\mathcal{B}$ to run $\mathcal{A}$ as follows:

1) Give to $\mathcal{A}$ the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and the challenge public key $\mathsf{pk}^\star = g_2^\beta$ as input. $\mathcal{B}$ runs $\mathcal{A}$ on randomness $f = (\rho, c_1, \cdots, c_{q_H})$.
2) The random oracle H_1 is answered as follows: initialise a list $\mathcal{L}_{H_1} = \emptyset$. For a query on y, if there exists $(y, c) \in \mathcal{L}_{H_1}$ then output c; otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$, update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup \{(y, c)\}$ and output c.
3) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, if there exists $(y, c) \in \mathcal{L}_{H_2}$, then output c; otherwise
   a) If $y = (\mathsf{pk}, J, \mathcal{PK})$ with $\mathsf{pk} \in \mathcal{PK}$ and $\mathsf{pk}^\star \in \mathcal{PK}$, assume this is the i-th query to H_2. For each $j \in J$ and $\mathsf{pk}_j \neq \mathsf{pk}^\star$, choose a random value $d_j \xleftarrow{\$} \mathbb{Z}_q$ and set $H_2(\mathsf{pk}_j, J, \mathcal{PK}) = d_j$. For $(\mathsf{pk}^\star, J, \mathcal{PK})$, it fixes $H_2(\mathsf{pk}^\star, J, \mathcal{PK}) = c_i$. Output $H_2(y)$.
   b) Else choose a random $d \xleftarrow{\$} \mathbb{Z}_q$ and set $H_2(y) = d$. Output d.
4) The random oracle H_3 is answered as follows: initialise $\mathcal{L}_{H_3} = \emptyset$. Let q_H be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$} [q_H]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_3}$, output h.
   - Otherwise,
     – If this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\alpha$ where g_1^α is from the co-CDH problem.
     – Else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g_1^r$.
     – Update $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (x, r, h)$ and output h.
5) To answer the sign query $\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)$ on a message x, call the H_3 oracle to obtain $(x, r, h) \in \mathcal{L}_{H_3}$,
   - If $r = \perp$, return $\perp$
   - Else output $g_1^{\beta \cdot r}$

Finally $\mathcal{A}$ outputs a forgery $(J, \sigma^\star)$, a message $m^\star$, a group public key $\mathsf{gpk} = (\mathsf{gtag}^\star, \mathcal{PK})$. If $(\mathsf{gtag}^\star, m^\star)$ is not the $\eta^\star$-th query to H_3, then $\mathcal{B}$ aborts. Since $\eta^\star$ is randomly chosen, the probability that $\mathcal{B}$ does not abort is $1/q_H$. Let's assume $\mathcal{B}$ does not abort. Parse $\mathcal{PK} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$. Suppose $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $1 \leq k \leq n$. Let j_f be the index such that $H_1(\mathsf{pk}^\star, J, \mathcal{PK}) = c_{j_f}$. Let $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(J, \mathcal{PK})$ and $a_j = H_2(\mathsf{pk}_j, J, \mathcal{PK})$ for each $j \in J$. $\mathcal{B}$ outputs $(\{j_f\}, \{(\sigma^\star, \mathcal{PK}, J, \mathsf{apk}, \{a_j\}_{j \in J})\})$.

The running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus the additional computations that $\mathcal{B}$ makes. Let $\tau_{\exp_1}$ (resp. $\tau_{\exp_2}$) be the time required to compute one exponentiation in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$). Let q_H be the combined number of random oracle queries to H_1, H_2, H_3. To answer q_H random oracle queries, $\mathcal{B}$ spends at most $q_H \cdot \tau_{\exp_1}$ time. To answer the signing queries, $\mathcal{B}$ spends at most $q_S \cdot \tau_{\exp_1}$ time. To construct apk, it takes $\ell \cdot \tau_{\exp_2}$ time where ℓ is the maximum number of signers involved in a single multi-signature. In total, $\mathcal{B}$'s running time is therefore $\tau + (q_H + q_S) \cdot \tau_{\exp_1} + \ell \cdot \tau_{\exp_2}$. $\mathcal{B}$'s success rate $\epsilon_\mathcal{B}$ is the probability that $\mathcal{A}$ succeeds and that $\mathcal{B}$ correctly guesses the hash index of $\mathcal{A}$'s forgery, thus $\epsilon_\mathcal{B} = \epsilon/q_H$

Now we construct another adversary $\mathcal{F}$ that on input a co-CDH instance and a forger $\mathcal{A}$, outputs a solution to the co-CDH instance. $\mathcal{F}$ runs the generalised forking lemma $\mathcal{GF}_\mathcal{B}$ with algorithm $\mathcal{B}$ constructed above. If $\mathcal{GF}_\mathcal{B}$ outputs $(\{j_f\}, \{out\}, \{out'\})$, then $\mathcal{B}$ proceeds as follows. $\mathcal{B}$ parses $out = (\sigma, \mathcal{PK}, J, \mathsf{apk}, \{a_j\}_{j \in J})$ and $out' = (\sigma', \mathcal{PK}', J', \mathsf{apk}', \{a'_j\}_{j \in J'})$. Let K be the index of $\mathsf{pk}^\star$ in $\mathcal{PK}$. Then we know $a_k = c_{j_f}$ and $a'_k = c'_{j_f}$ and $a_k \neq a'_k$. Since out, out' are obtained from two executions of $\mathcal{B}$ with randomness f, f' such that $f\ |_{j_f} = f'\ |_{j_f}$, we can derive that $\mathcal{PK} = \mathcal{PK}', J = J'$ and $a_j = a'_j$ for $j \in J \setminus \{k\}$. Therefore we have $\mathsf{apk}/\mathsf{apk}' = g_2^{\beta(a_k - a'_k)}$. Since $\mathcal{B}$'s output satisfies $e(\sigma, g_2) = e(g_1^\alpha, \mathsf{apk})$ and $e(\sigma', g_2) =$

$e(g_1^\alpha, \mathsf{apk}')$, we can compute $(\sigma/\sigma')^{1/(a_k - a_k')}$ as a solution to the co-CDH instance.

Using the generalised forking lemma, if $q > 8q_H/\epsilon$, $\mathcal{F}$ runs in time at most $(\tau + (q_H + q_S) \cdot \tau_{\exp_1} + \ell \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon)$ and succeeds with probability at least $\epsilon/(8q_H)$. $\square$

4.2. Optimising RSMSP with NIZKs

We can use NIZKs validate individual signatures in order to speed up the verification of individual signatures in the Combine function in RSMSP. We call this optimised scheme RSMSP-zk.

Construction of RSMSP-zk**.** The algorithms KeyGen, GroupSet, KeyAgg, VerifyMul are implemented exactly the same as RSMSP, but the algorithms Sign and Combine are optimised as below:

Sign$(m, \mathsf{sk}_i, \mathsf{gtag})$**:** Output $s_i = (v_i, \pi_i)$ where $v_i = H_3(\mathsf{gtag}, m)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_4}(g_2, H_3(\mathsf{gtag}, m), \mathsf{pk}_i, v_i; r)$ for randomness $r \xleftarrow{\$} \mathbb{Z}_q$.

Combine$(m, \mathcal{E}, \mathsf{gpk})$**:** Parse $\mathcal{E} = \{(v_i, \pi_i)\}_{i\in I}$ of individual signatures originating from $|I|$ different entities, and $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK})$. The combiner verifies each signature by checking $\mathsf{VerEq}_{H_4}((g_2, H_3(\mathsf{gtag}, m), \mathsf{pk}_i, v_i, \pi_i) = 1$. Assume the index set of valid signatures is $J \subseteq I$. The combiner computes the multi-signature as $\sigma = \prod_{j\in J} v_j^{a_j}$ with $a_j = H_2(\mathsf{pk}_j, J, \mathcal{PK})$ for $j \in J$. Output (J, σ).

Theorem 4.3. RSMSP-zk *is robust in the random oracle model.*

The proof of the robustness of RSMSP-zk can be found in Appendix A.

Theorem 4.4. RSMSP-zk *is unforgeable under the computational co-CDH assumption in the random oracle model. Formally,* RSMSP *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* $q > 8q_H/\epsilon$ *and co-CDH is* $(\tau + (q_H + 3q_S) \cdot \tau_{\exp_1} + (2q_S + \ell) \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon), \epsilon/(8q_H))$*-hard where* ℓ *is the maximum number of signers involved in a single multi-signature and* $\tau_{\exp_1}$ *(resp.* $\tau_{\exp_2}$*) is the time required to compute one exponentiation in* $\mathbb{G}_1$ *(resp.* $\mathbb{G}_2$*).*

Proof. The analysis is similar to Theorem 4.2 except that the Sign queries are answered with simulated NIZKs generated by an additional random oracle $H_4 : \{0,1\}^* \mapsto \mathbb{Z}_q$:

- The random oracle H_4 is programmed as follows: Define a list $\mathcal{L}_{H_4} = \emptyset$. For a query on y, if $(y, c) \in \mathcal{L}_{H_4}$, then output c. Otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$, update the list $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup (y, c)$ and output c.
- To answer the sign query $\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)$ on a message x, call the H_3 oracle to obtain $(x, r, h) \in \mathcal{L}_{H_3}$,
  - If $r = \bot$, return $\bot$
  - Else output $g_1^{\beta \cdot r}$ and a simulated NIZK proof π using the random oracle H_4

To compute a simulated NIZK proof, it costs $\mathcal{B}$ $2(\tau_{\exp_1} + \tau_{\exp_2})$ time. Adding the time $\tau_{\exp_1}$ for computing each signature, $\mathcal{B}$ spends at most $q_S \cdot (3\tau_{\exp_1} + 2\tau_{\exp_2})$ time to answer all the signing queries. In total $\mathcal{B}$'s running time is at most $\tau + (q_H + 3q_S) \cdot \tau_{\exp_1} + (2q_S + \ell) \cdot \tau_{\exp_2}$ and success rate is ϵ/q_H. The running time of the adversary $\mathcal{F}$ that runs $\mathcal{GF}_{\mathcal{B}}$ is $(\tau + (q_H + 3q_S) \cdot \tau_{\exp_1} + (2q_S + \ell) \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon)$ and succeeds with probability at least $\epsilon/(8q_H)$. $\square$

4.3. Construction of RSMS-pop from proof-of-possession

In proof-of-possession (PoP) based multi-signatures, an additional public key is generated as a proof of knowledge of the secret key. Obviously the disadvantage of using PoPs is the extra storage and computation overhead introduced by the PoPs. However, PoPs offer a few competitive advantages as well. The aggregation of signatures and public keys in PoPs are simply multiplications without any exponentiation. The security argument of PoPs does not require the generalised forking lemma which gives much tighter reductions. Constructing secure robust-subgroup multi-signatures from PoPs [RY07], [BDN18] turns out to be quite straightforward, since security reductions of PoPs do not require to fix the set of co-signers. For the completeness of this work, we describe the construction of RSMS-pop and give security proofs below.

Construction of RSMS-pop**.** Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be a bilinear group with prime order q and $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$. Assume hash functions $H_1 : \mathbb{G}_2 \mapsto \mathbb{G}_1$, $H_2 : \{0,1\}^* \mapsto \mathbb{Z}_q$, $H_3 : \{0,1\}^* \mapsto \mathbb{G}_1$.

KeyGen(1^λ)**:** Choose $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$, compute $\mathsf{pk} = (pk, pop)$ with $pk = g_2^{\mathsf{sk}}$ and $pop = H_1(pk)^{\mathsf{sk}}$, and output $(\mathsf{sk}, \mathsf{pk})$. The public key pk is made public and its validity can be checked by $e(pop, g_2) = e(H_1(pk), pk)$.

GroupSet$(\mathcal{PK})$**:** Parse $\mathcal{PK} = \{(pk_i, pop_i)\}_{i\in I}$. For each $i \in I$, check if $e(pop_i, g_2) = e(H_1(pk_i), pk_i)$. If all successful, compute $\mathsf{gtag} = H_2(\mathcal{PK})$ and output $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK})$; else output $\bot$.

$\mathsf{KeyAgg}(J, \mathcal{PK})$**:** Parse $\mathcal{PK} = \{(pk_i, pop_i)\}_{i \in I}$. If $J \not\subseteq I$, then output $\perp$. Compute and output $\mathsf{apk} = \prod_{j \in J} pk_j$.

$\mathsf{Sign}(m, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{gtag})$**:** Output an individual signature $s_i = H_3(\mathsf{gtag}, m)^{\mathsf{sk}_i}$.

$\mathsf{Combine}(m, \mathcal{E}, \mathsf{gpk})$**:** Parse $\mathsf{gpk} = (\mathsf{gtag}, \{\mathsf{pk}_i\}_{i \in U})$ and the set $\mathcal{E} = \{s_i\}_{i \in I}$ of individual signatures originating from $|I|$ different entities such that $I \subseteq U$. The combiner verifies each signature in $\mathcal{E}$ by checking $e(s_i, g_2) = e(H_3(\mathsf{gtag}, m), pk_i)$ where $\mathsf{pk}_i = (pk_i, pop)$. Assume the index set of valid signatures is $J \subseteq I$. The combiner computes the multi-signature as $\sigma = \prod_{j \in J} \sigma_j$ and outputs (J, σ).

$\mathsf{VerifyMul}(m, J, \sigma, \mathsf{gpk})$**:** Parse $\mathsf{gpk} = (\mathsf{gtag}, \mathcal{PK} = \{\mathsf{pk}_i\}_{i \in I})$. If $J = \emptyset$ or $J \not\subseteq I$, then output 0. Otherwise, compute $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(J, \mathcal{PK})$. Output 1 if $e(\sigma, g_2) = e(H_3(\mathsf{gtag}, m), \mathsf{apk})$; else output 0.

Theorem 4.5. RSMS-pop *is robust.*

Proof. Suppose there exists an adversary $\mathcal{A}$ $(\tau, q_S, q_H, \epsilon)$-breaks robustness. We will show that this leads to a contradiction. At the end of the robustness game, the adversary outputs $(m^\star, \mathsf{gpk} = (\mathsf{gtag}^\star, \mathcal{PK} = \{\mathsf{pk}_i\}_{i \in U}), \mathcal{E} = \{s_i\}_{i \in I})$ with $\mathsf{pk}_i = (pk_i, pop_i)$ for each $i \in U$ and $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $k \in U$. When the challenger outputs 1, we know $e(\sigma^\star, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \prod_{j \in J \cup \{k\}} pk_j)$ where $s \leftarrow \mathsf{Sign}(m^\star, \mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{gtag}^\star)$, $(J, \sigma^\star) \leftarrow \mathsf{Combine}(m^\star, \mathcal{E} \cup \{s\}, \mathsf{gpk}^\star)$. This means there exists $j \in J$ such that $e(s_j, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), pk_j)$ and $\mathsf{pk}_j = (pk_j, pop_j)$. This contradicts to the fact that all the individual signatures with indices included in $J^\star$ satisfy $e(s_j, g_2) = e(H_3(\mathsf{gtag}^\star, m^\star), pk_j)$ for $j \in J$. □

Theorem 4.6. RSMS-pop *is unforgeable under the computational co-CDH assumption in the random oracle model. Formally,* RSMS-pop *is $(\tau, q_S, q_H, \epsilon)$-unforgeable if co-CDH is $(\tau + (q_H + q_S + \ell) \cdot \tau_{\exp_1}, \epsilon / q_H)$-hard where ℓ is the maximum number of signers involved in a single multi-signature and $\tau_{\exp_1}$ is the time required to compute one exponentiation in $\mathbb{G}_1$.*

Proof. Given a co-CDH problem $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\beta)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \overset{\$}{\leftarrow} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1) Give to $\mathcal{A}$ the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$, the challenge public key $\mathsf{pk}^\star = (pk^\star, pop^\star)$ where $pk^\star = g_2^\beta$ and $pop^\star = g_1^{\beta \cdot r^\star}$ with $r^\star \overset{\$}{\leftarrow} \mathbb{Z}_p$.
2) The random oracle H_1 is answered as follows: initialise a list $\mathcal{L}_{H_1} = \emptyset$.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     - If $x = pk^\star$, set $h = g_1^{r^\star}$ and update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r^\star, h)$ and output h.
     - Else choose a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$ and set $h = g_1^{\alpha \cdot r}$ where g_1^α is from the co-CDH problem, update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.
3) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, if $(y, c) \in \mathcal{L}_{H_2}$, then output c. Otherwise choose a random $c \overset{\$}{\leftarrow} \mathbb{Z}_q$, update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$ and output c.
4) The random oracle H_3 is answered as follows: initialise $\mathcal{L}_{H_3} = \emptyset$. Let q_{H_3} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \overset{\$}{\leftarrow} [q_{H_3}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_3}$, output h.
   - Otherwise,
     - If this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\alpha$ where g_1^α is from the co-CDH problem.
     - Else choose a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$ and set $h = g_1^r$.
     - Update $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (x, r, h)$ and output h.
5) To answer the sign query $\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)$ on a message x, call the H_3 oracle to obtain $(x, r, h) \in \mathcal{L}_{H_3}$,
   - If $r = \perp$, return $\perp$
   - Else output $g_1^{\beta \cdot r}$

Finally $\mathcal{A}$ outputs a forgery $(J, \sigma^\star)$, a message $m^\star$, a group public key $\mathsf{gpk}^\star = (\mathsf{gtag}^\star, \mathcal{PK})$. If $(m^\star, \mathsf{gtag}^\star)$ is not the $\eta^\star$-th query to H_3, then $\mathcal{B}$ aborts. Since $\eta^\star$ is randomly chosen, the probability that $\mathcal{B}$ does not abort is $1/q_{H_3}$. Let's assume $\mathcal{B}$ does not abort. Parse $\mathcal{PK} = \{\mathsf{pk}_i\}_{i \in U}$ with each $\mathsf{pk}_i = (pk_i, pop_i)$. Suppose $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $k \in U$. Based on the construction of H_1, we have $H_1(pk_i) = g_1^{\alpha \cdot r_i}$ for each $i \in U \setminus \{k\}$. Combing with $e(H_1(pk_i), pk_i) = e(pop_i, g_2)$ obtained from Condition 1 in Definition 3.3, we have $e(g_1^\alpha, pk_i) = e(pop_i^{r_i^{-1}}, g_2)$ for each $i \in U \setminus \{k\}$. Because $e(\sigma^\star, g_2) = e(H_3(\mathsf{gtag}^\star, m^\star), \prod_{i \in J} pk_i)$ by the Condition 3 in Definition 3.3, we can derive $e(\sigma^\star, g_2) = e(g_1^\alpha, g_2^\beta \cdot \prod_{i \in J}^{i \neq k} pk_i) = e(g_1^\alpha, g_2^\beta) \cdot e(g_1^\alpha, \prod_{i \in J}^{i \neq k} pk_i) = e(g_1^\alpha, g_2^\beta) \cdot e(\prod_{i \in J}^{i \neq k} pop_i^{r_i^{-1}}, g_2)$. Let $\delta = \sigma^\star \cdot \prod_{i \in J}^{i \neq k} pop_i^{-r_i^{-1}}$. Then we have $e(\delta, g_2) = e(\sigma^\star \cdot \prod_{i \in J}^{i \neq k} pop_i^{-r_i^{-1}}, g_2) =$

$e(\sigma^\star, g_2)/e(\prod_{i\in J}^{i\neq k} pop_i^{r_i^{-1}}, g_2) = e(g_1^\alpha, g_2^\beta)$. Therefore, $\mathcal{B}$ can return δ as a solution to the co-CDH problem with a success probability at least ϵ/q_H.

The running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus the additional computations that $\mathcal{B}$ makes. Let $\tau_{\exp_1}$ be the time required to compute one exponentiation in $\mathbb{G}_1$. The setup of the challenge verification key costs $\tau_{\exp_1}$. Let q_H be the combined number of random oracle queries to H_1, H_2, H_3, H_4. To answer q_H random oracle queries, $\mathcal{B}$ spends at most $(q_H - 1)\cdot\tau_{\exp_1}$ time. $\mathcal{B}$ spends at most $q_S \cdot \tau_{\exp_1}$ to answer all the signing queries. To construct the solution δ, it takes at most $\ell\cdot\tau_{\exp_1}$ time where ℓ is the maximum number of signers involved in a single multi-signature. In total, $\mathcal{B}$'s running time is $\tau + (q_H + q_S + \ell)\cdot\tau_{\exp_1}$. □

Remark 2. *Theorems 4.5 and 4.6 validate the correctness of using the multi-signatures protocols for achieving consensus in [Har], [Cod], [GGJ+20].*

In the next subsection, we shall discuss how to optimise RSMS-pop with NIZKs to obtain a more efficient scheme RSMS-pop-zk. The performance evaluation given in Section 6 will show our RSMS-pop-zk is 2x faster than RSMS-pop in terms of combination of individual signatures.

4.4. Optimising RSMS-pop with NIZKs

Construction of RSMS-pop-zk. The algorithms KeyGen, KeyAgg, GroupSet, VerifyMul are instantiated in the same way as RSMS-pop, but the algorithms Sign and Combine now use NIZKs to validate individual signatures:

$\mathsf{Sign}(m, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{gtag})$**:** Parse $\mathsf{pk}_i = (pk_i, pop_i)$. Compute $v_i = H_3(\mathsf{gtag}, m)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_4}(H_1(pk_i), H_3(\mathsf{gtag}, m), pop_i, v_i; r)$ for randomness $r \overset{\$}{\leftarrow} \mathbb{Z}_q$. Output $s_i = (v_i, \pi_i)$.

$\mathsf{Combine}(m, \mathcal{E}, \mathsf{gpk})$**:** Parse $\mathsf{gpk} = (\mathsf{gtag}, \{(pk_i, pop_i)\}_{i\in U})$ and the set $\mathcal{E} = \{(v_i, \pi_i)\}_{i\in I}$ of individual signatures originating from $|I|$ different entities such that $I \subseteq U$. The combiner verifies each signature in $\mathcal{E}$ by checking $\mathsf{VerEq}_{H_4}(H_1(pk_i), pop_i, H_3(\mathsf{gtag}, m), v_i, \pi_i) = 1$. Assume the index set of valid signatures is $J \subseteq I$. The combiner computes the multi-signature as $\sigma = \prod_{j\in J} v_j$ and outputs (J, σ).

Theorem 4.7. *Our* RSMS-pop-zk *scheme is robust in the random oracle model.*

The proof of the robustness of RSMS-pop-zk can be found in Appendix A.

Theorem 4.8. RSMS-pop-zk *is unforgeable under the computational co-CDH assumption in the random oracle model. Formally,* RSMS-pop-zk *is $(\tau, q_S, q_H, \epsilon)$-unforgeable if co-CDH is $(\tau + (q_H + 5q_S + \ell)\cdot\tau_{\exp_1}, \epsilon/q_H)$-hard where ℓ is the maximum number of signers involved in a single multi-signature and $\tau_{\exp_1}$ is the time required to compute one exponentiation in $\mathbb{G}_1$.*

Proof. The analysis is the same as Theorem 4.6 except that Sign queries are answered with simulated NIZKs generated by an additional random oracle $H_4 : \{0,1\}^* \mapsto \mathbb{Z}_q$:

- The random oracle H_4 is programmed as follows: Define a list $\mathcal{L}_{H_4} = \emptyset$. For a query on y, if $(y, c) \in \mathcal{L}_{H_4}$, then output c. Otherwise choose a random $c \overset{\$}{\leftarrow} \mathbb{Z}_q$, update the list $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup (y, c)$ and output c.
- To answer the sign query $\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)$ on a message x, call the H_3 oracle to obtain $(x, r, h) \in \mathcal{L}_{H_3}$,
  - If $r = \perp$, return $\perp$
  - Else output $g_1^{\beta\cdot r}$ and a simulated NIZK proof π using the random oracle H_4

For each Sign query, $\mathcal{B}$ needs to generate a simulated NIZK proof which takes $4\tau_{\exp_1}$ time for each proof. Adding the time $\tau_{\exp_1}$ for computing each signature, $\mathcal{B}$ spends at most $q_S\cdot 5\tau_{\exp_1}$ to answer all the signing queries. Therefore, $\mathcal{B}$'s total running time is at most $\tau + (q_H + 5q_S + \ell)\cdot\tau_{\exp_1}$ and $\mathcal{B}$'s success probability is ϵ/q_H. □

5. Extensions

In this section, we extend our robust subgroup multi-signatures with two useful functions: compression and aggregation. We give formal definitions and security analysis for these extensions.

5.1. Compressing two multi-signatures on the same message

It is possible to *compress* two multi-signatures on the same message if signed by disjoint subgroups. In practice, this enables any entity to start the signature combination process without the need of waiting until the entity gets all the individual signatures. This incremental aggregation feature was previously mentioned in [DGNW20] but was never formalised. Below we shall extend our definition of robust subgroup multi-signature scheme RSMS with an additional algorithm called Compress:

$\mathsf{Compress}(m, J_1, \sigma_1, J_2, \sigma_2, \mathsf{gpk})$**:** on input a message m, two multi-signatures (J_1, σ_1),

(J_2, σ_2), and a group public key gpk, the algorithm compress two multi-signatures into a single multi-signature and outputs (J, σ) with $J = J_1 \cup J_2$. When the algorithm fails to compress the multi-signatures the output is $(J, \sigma) = (\emptyset, \perp)$.

Robustness. We extend the robustness in Definition 3.2 to include *compressibility*: for any $m, J_1, J_2, \sigma_1, \sigma_2, \mathsf{gpk}$ such that $J_1 \cap J_2 = \emptyset$ and $J_1, J_2 \neq \emptyset$, if $\mathsf{VerifyMul}(m, J_b, \sigma_b, \mathsf{gpk}) = 1$ for $b = 1, 2$, then $\mathsf{VerifyMul}(m, J, \sigma, \mathsf{gpk}) = 1$ where $(J, \sigma) \leftarrow \mathsf{Compress}(m, J_1, \sigma_1, J_2, \sigma_2, \mathsf{gpk})$.

Unforgeability. The definition of unforgeability does not change since the function Compress does not involve any secret information and thus the adversary can run this algorithm without interacting with the signers.

Instantiations. Unfortunately, RSMSP and RSMSP-zk are not compressible, because two multi-signatures $\sigma_b = \prod_{i \in J_b} s_i^{a_{i,b}}$ with coefficients $a_{i,b} = H_2(\mathsf{pk}_i, J_b, \mathcal{PK})$ and $b = 0, 1$, cannot be compressed into $\sigma = \prod_{i \in J_1 \cup J_2} s_i^{a_i}$ with coefficients $a_i = H_2(\mathsf{pk}_i, J_1 \cup J_2, \mathcal{PK})$. However, the PoP-based schemes RSMS-pop and RSMS-pop-zk can be extended with Compress as follows:

$\mathsf{Compress}(m, J_1, \sigma_1, J_2, \sigma_2, \mathsf{gpk})$**:** Check if $J_1 \cap J_2 = \emptyset$ and $\mathsf{VerifyMul}(m, J_b, \sigma_b, \mathsf{gpk}) = 1$ for $b = 0, 1$. If successful, output $(J_1 \cup J_2, \sigma_1 \cdot \sigma_2)$. Otherwise output $(\emptyset, \perp)$.

Theorem 5.1. *The extended* RSMS-pop *and* RSMS-pop-zk *schemes satisfy robustness in the random oracle model.*

Proof. We only need to show the compressibility defined above. Indeed, since $e(\sigma_i, g_2) = e(H_2(\mathsf{gtag}, m), \mathsf{apk}_i)$ for $i = 1, 2$ it follows that $e(\sigma, g_2) = e(H_2(\mathsf{gtag}, m), \mathsf{apk})$ where $\sigma = \sigma_1 \cdot \sigma_2$ and $\mathsf{apk} = \mathsf{apk}_1 \cdot \mathsf{apk}_2$. □

Theorem 5.2. *The extended* RSMS-pop *and* RSMS-pop-zk *schemes are unforgeable under the computational co-CDH problem in the random oracle model.*

Proof. As mentioned above, adding the Compress algorithm does not change the definition of unforgeability since the adversary can run Compress by themselves. The proofs of unforgeability of RSMS-pop (resp. RSMS-pop-zk) are therefore the same as for Theorem 4.6 (resp. 4.8) and are omitted. □

5.2. Aggregating multi-signatures on different messages

Multi-signatures on different messages can be further aggregated into a single aggregate signature, e.g., [BDN18]. In this section, we show that the multi-signatures on different messages signed by different groups in our robust-subgroup multi-signature schemes can also be aggregated, and we call this extension *aggregate robust-subgroup multi-signatures* (ARSMS).

Definition 5.1 (Aggregate Robust-Subgroup Multi-Signature)**.** *An* aggregate robust-subgroup multi-signature *scheme (*ARSMS*) is defined as* $\Pi = (\mathsf{KeyGen}, \mathsf{GroupSet}, \mathsf{KeyAgg}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifyMul}, \mathsf{Aggregate}, \mathsf{VerifyAgg})$, *where algorithms* $\mathsf{KeyGen}, \mathsf{GroupSet}, \mathsf{KeyAgg}, \mathsf{Sign}, \mathsf{Combine}$ *and* $\mathsf{VerifyMul}$ *are exactly the same as* RSMS *in Definition 3.1, and*

$\mathsf{Aggregate}(\{(m_i, J_i, \sigma_i, \mathsf{gpk}_i)\}_{i \in I})$**:** *on input a set of tuples of message, multi-signature and group public key, the algorithm identifies an index set $K \subseteq I$ of valid multi-signatures and aggregates them into a single aggregate signature Σ and outputs $(\{J_i\}_{i \in K}, \Sigma)$. When the algorithm fails to aggregate signatures, output $\perp$.*

$\mathsf{VerifyAgg}(\{(m_i, J_i, \mathsf{gpk}_i)\}_{i \in K}, \Sigma)$**:** *on input tuples of messages, indices and group public keys, and an aggregate signature Σ, the algorithm outputs 1 if Σ is a valid aggregate signature and outputs 0 otherwise.*

Robustness of ARSMS**.** We define robustness for an ARSMS scheme by extending Definition 3.2 with the robustness for aggregate signatures: if $\mathsf{VerifyMul}(m_i, J_i, \sigma_i, \mathsf{gpk}_i) = 1$ for each $i \in I$, and $\Sigma \leftarrow \mathsf{Aggregate}(\{(m_i, J_i, \sigma_i, \mathsf{gpk}_i)\}_{i \in I})$, then it holds that $\mathsf{VerifyAgg}(\{(m_i, J_i, \mathsf{gpk}_i)\}_{i \in I}, \Sigma) = 1$.

Definition 5.2 (Unforgeability of ARSMS)**.** *The* unforgeability *of an* ARSMS *scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{GroupSet}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{VerifyMul}, \mathsf{Aggregate}, \mathsf{VerifyAgg})$ *is defined by a three-stage game:*

Setup *The challenger generates the system parameters* pp *and a challenge key pair* $(\mathsf{sk}^\star, \mathsf{pk}^\star) \leftarrow \mathsf{KeyGen}(1^\lambda)$. *It gives* $(\mathsf{pk}^\star, \mathsf{pp})$ *to the adversary.*

Signature queries *$\mathcal{A}$ is allowed to make signature queries on any message m for any group tag* gtag*, meaning that it has access to oracle* $\mathcal{O}^{\mathsf{Sign}(\cdot, \mathsf{sk}^\star, \mathsf{pk}^\star, \cdot)}$ *that simulates the honest signer signing a message m.*

Output *Finally, the adversary outputs an aggregate signature forgery* $(\{m_i, J_i, \mathsf{gpk}_i = (\mathsf{gtag}_i, \mathcal{PK}_i)\}_{i\in K}, \Sigma^\star)$*. The adversary wins if*

1) *For each* $i \in K$*,* $\mathsf{gpk}_i \neq \perp$ *and* $\mathsf{gpk}_i = \mathsf{GroupSet}(\mathcal{PK}_i)$
2) *For each* $i \in K$*, let* $\mathcal{PK}_i = \{\mathsf{pk}_{i,j}\}_{j\in I_i}$*, then* $J_i \subseteq I_i$*.* $\mathsf{pk}^\star = \mathsf{pk}_{k^\star,j^\star}$ *for some* $k^\star \in K$ *and* $j^\star \in J_{k^\star}$*.*
3) *The pairs* (gtag_i, m_i) *with* $i \in K$ *are pairwise-distinct.* $\mathcal{A}$ *made no signing queries on* $(m_{k^\star}, \mathsf{gtag}_{k^\star})$*.*
4) $\mathsf{VerifyAgg}(\{(m_i, J_i, \mathsf{gpk}_i)\}_{i\in K}, \Sigma^\star) = 1$*.*

We say $\mathcal{A}$ *is a* $(\tau, q_S, q_H, \epsilon)$*-forger for* ARSMS *if* $\mathcal{A}$ *runs in time at most* τ*, makes at most* q_S *signing queries and at most* q_H *random oracle queries, and wins the above game with probability at least* ϵ*.* ARSMS *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if no such adversary exists.*

Instantiations. All of our robust-subgroup multi-signatures, i.e., RSMSP, RSMSP-zk, RSMS-pop, RSMS-pop-zk, can be extended with the following algorithms:

$\mathsf{Aggregate}(\{(m_i, J_i, \sigma_i, \mathsf{gpk}_i)\}_{i\in I})$**:** Output $\Sigma \leftarrow \prod_{i\in I} \sigma_i$.

$\mathsf{VerifyAgg}(\{(m_i, J_i, \mathsf{gpk}_i)\}_{i\in I}, \Sigma)$**:** For each $i \in I$, parse $\mathsf{gpk}_i = (\mathsf{gtag}_i, \mathcal{PK}_i)$ and compute $\mathsf{apk}_i \leftarrow \mathsf{KeyAgg}(J_i, \mathcal{PK}_i)$. Output 1 if $e(\Sigma, g_2) = \prod_{i\in I} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$; else output 0.

We call the extended schemes ARSMSP, ARSMSP-zk, ARSMS-pop, ARSMS-pop-zk, respectively.

Theorem 5.3. ARSMSP, ARSMSP-zk, ARSMS-pop *and* ARSMS-pop-zk *are robust in the random oracle model.*

Proof. To show robustness, we only need to prove the robustness for aggregate signatures. Let $(\{J_i\}_{i\in K}, \Sigma) \leftarrow \mathsf{Aggregate}(\{(m_i, J_i, \sigma_i, \mathsf{gpk}_i)\}_{i\in I})$. From the instantiation of Aggregate and definition of robustness, we have $\Sigma = \prod_{i\in K} \sigma_i$, and for each $i \in K$, $\mathsf{VerifyMul}(m_i, J_i, \sigma_i, \mathsf{gpk}_i) = 1$. From the instantiation of VerifyMul, we have $e(\sigma_i, g_2) = e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$ where $\mathsf{gpk}_i = (\mathsf{gtag}_i, \mathcal{PK}_i)$ and $\mathsf{apk}_i \leftarrow \mathsf{KeyAgg}(J_i, \mathcal{PK}_i)$ for each $i \in K$. Therefore $e(\Sigma, g_2) = \prod_{i\in K} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$. This completes the proof. □

The proofs of the following theorems for unforgeability can be found in Appendix B.

Theorem 5.4. ARSMSP *and* ARSMSP-zk *are unforgeable under the computational* ψ*-co-CDH assumption in the random oracle model. Formally,*

1) ARSMSP *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* $q > 8q_H/\epsilon$ *and* ψ*-co-CDH is* $((\tau + (q_H + q_S + n) \cdot \tau_{\exp_1} + \ell \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon), \epsilon/(8q_H))$*-hard;*
2) ARSMSP-zk *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* $q > 8q_H/\epsilon$ *and* ψ*-co-CDH is* $(\tau + (q_H + 3q_S + n) \cdot \tau_{\exp_1} + (2q_S + \ell) \cdot \tau_{\exp_2}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon), \epsilon/(8q_H))$*-hard.*

where ℓ *is the maximum number of signers involved in an aggregate signature,* n *is the maximum number of multi-signatures involved in an aggregate signature, and* $\tau_{\exp_1}$ *(resp.* $\tau_{\exp_2}$*) is the time required to compute one exponentiation in* $\mathbb{G}_1$ *(resp.* $\mathbb{G}_2$*).*

Theorem 5.5. ARSMS-pop *and* ARSMS-pop-zk *are unforgeable under the computational co-CDH assumption in the random oracle model. Formally,*

1) ARSMS-pop *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* ψ*-co-CDH is* $(\tau + (q_H + q_S + \ell + n) \cdot \tau_{\exp_1}, \epsilon/q_H)$*-hard.*
2) ARSMS-pop-zk *is* $(\tau, q_S, q_H, \epsilon)$*-unforgeable if* ψ*-co-CDH is* $(\tau + (q_H + 5q_S + \ell + n) \cdot \tau_{\exp_1}, \epsilon/q_H)$*-hard*

where ℓ *is the maximum number of signers involved in a single multi-signature,* n *is the maximum number of multi-signatures involved in an aggregate signature, and* $\tau_{\exp_1}$ *is the time required to compute one exponentiation in* $\mathbb{G}_1$*.*

Compressibility. The schemes ARSMS-pop and ARSMS-pop-zk are compressible and can be extended with the same Compress algorithm for RSMS-pop and RSMS-pop-zk.

6. Performance Evaluation

We implement the four multi-signature schemes, i.e., RSMSP, RSMSP-zk, RSMS-pop and RSMS-pop-zk, using the libray MCL [Mis19]. RSMSP and RSMSP-zk are implemented by swapping $\mathbb{G}_1$ and $\mathbb{G}_2$, i.e., putting the public keys on $\mathbb{G}_1$ and the signatures on $\mathbb{G}_2$. This is because VerifyMul and VerifyAgg involve reconstruction of the aggregate public keys which contains time-consuming exponentiation operations on the public keys. Putting public keys on $\mathbb{G}_1$ can mitigate this issue since the exponentiations on $\mathbb{G}_1$ are typically faster than the ones on $\mathbb{G}_2$. However, this will make Combine slower because the signature aggregations are now performed on $\mathbb{G}_2$. We note that Combine only needs to be run by a limited set of nodes like validators, but VerifyMul and VerifyAgg need to be run by any node that first obtains the multi-signatures. In addition, our RSMSP-zk scheme has optimised Combine using NIZKs which significantly improves the efficiency of Combine and offsets the loss. Therefore, switching $\mathbb{G}_1$ and $\mathbb{G}_2$ can provide better overall performance.

Performance evaluation of the Combine function. The Combine function can be executed by any node in the network since it does not require any secret information. Combine only needs to be run once when computing the multi-signature from a set of individual signatures. Table 1 gives timing for combining a majority number $\lfloor \ell/2 \rfloor + 1$ of individual signatures from a group of size ℓ with $\ell = 64, 128, 256$. RSMSP-zk (resp. RSMS-pop-zk) is the optimised version of RSMSP (resp. RSMS-pop) by validating an individual signature using NIZK proofs instead of pairing equations. This optimisation makes Combine in RSMSP-zk (resp. RSMS-pop-zk) 2x faster than RSMSP (resp. RSMS-pop).

Performance comparison between multi-signatures and ECDSA. We shall compare the performance of the four multi-signature schemes with ECDSA signatures, in order to show that it is feasible to replace ECDSA with multi-signatures. The performance of ECDSA depends on the optimisations implemented in different libraries. The secp256k1 curve is used in Bitcoin for ECDSA signatures. The OpenSSL implementation of secp256k1 is not well optimised which leads to the low efficiency of ECDSA signing and verification operations. The Sodium library [BD19] provides a particularly efficient implementation of ECDSA over the curve Ed25519.

As shown in Table 2, the size of a multi-signature that combines/aggregates n individual signatures are constant, regardless of n. In comparison, n ECDSA signatures is of $64n$ bytes. We take consensus as an example. Suppose each committee has 200 validators and a notarisation of a new block requires at least 101 signatures. With ECDSA signatures, this accounts for 6.464KB storage overhead for each block, while it is just 48 Bytes for RSMS-pop/RSMS-pop-zk and 96 bytes for RSMSP/RSMSP-zk.

In terms of the size of the public key for each entity, RSMS-pop and RSMS-pop-zk incurs the extra overhead introduced by the use of proof-of-possession for preventing rogue-key attacks. The public keys and PoPs only need to be verified once and stored in the blockchain for future reference. In comparison, the public keys in RSMSP and RSMSP-zk are 66% smaller than RSMS-pop and RSMS-pop-zk. For 200 validators, this means saving 19.2KB for storing public keys.

In Table 3, we compare the efficiency of the four multi-signature schemes with ECDSA. The Sign column measures the timing for creating an individual signature which are less than 1ms for all four multi-signature schemes.

The VerifyMul columns in Table 3 present the timings for verifying a multi-signature signed by a threshold (or majority) number of members from a group of ℓ members with $\ell = 64, 128, 256$. The threshold is set to be $\lfloor \ell/2 \rfloor + 1$. When using ECDSA for the same purpose as multi-signatures, this is equivalent to $\lfloor \ell/2 \rfloor + 1$ ECDSA signatures. From the table we can see that RSMS-pop and RSMS-pop-zk outperforms ECDSA and RSMSP and RSMSP-zk in terms of the verification time for a multi-signature. This is because VerifyMul in RSMS-pop and RSMS-pop-zk mainly involves computing two pairings which is almost constant. In comparison, VerifyMul in RSMS-pop and RSMS-pop-zk requires computing a sequence of exponentiations on the public keys in order to reconstruct the aggregate public keys for the subgroup.

The VerifyAgg columns in Table 3 give the timings for verifying an aggregate signature signed by k groups ($k = 64, 128, 256$) where each group consists of 128 members and a majority number (i.e., 65) of members signed. When using ECDSA for the same purpose, this corresponds to $65 \cdot k$ ECDSA signatures. For all four multi-signature schemes, the timing for verifying k multi-signatures using VerifyAgg is less than $k \cdot t$ where t is the time for verifying a multi-signature using VerifyMul. This is because VerifyAgg functions as a batch verification which saves $k - 1$ pairing computations. Similar to VerifyMul, RSMS-pop and RSMS-pop-zk outperforms RSMSP and RSMSP-zk and ECDSA in terms of the verification of aggregate signatures.

References

[And11] G Andresen. Bitcoin improvement proposal (BIP)0011, 2011. https://en.bitcoin.it/wiki/BIP_0011.

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS*, pages 449–458, 2008.

[BD19] Daniel J. Bernstein and Frank Denis. Libsodium - a modern, portable, easy to use crypto library. https://github.com/jedisct1/libsodium, 2019.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018*, pages 435–464, 2018.

Protocol	Curve	Combine (ms)		
		64	128	256
RSMSP	BN256	44.720	89.576	176.031
	BLS12-381	51.803	96.384	190.928
	BN384	50.561	95.561	188.863
RSMSP-zk	BN256	26.228	51.951	101.077
	BLS12-381	26.943	52.931	105.625
	BN384	26.993	53.579	106.625
RSMS-pop	BN256	37.278	73.962	146.960
	BLS12-381	42.156	80.008	159.054
	BN384	42.880	80.214	159.365
RSMS-pop-zk	BN256	18.713	37.138	72.602
	BLS12-381	19.828	38.554	76.972
	BN384	19.212	37.994	76.048

Table 1: Performance evaluation and comparison of the Combine function. The columns of Combine give the timings for combining a threshold number $\lfloor \ell/2 \rfloor + 1$ of individual signatures from a group of size ℓ with $\ell = 64, 128, 256$. That is, 33 (resp. 65, 129) individual signatures from a group of 64 (resp. 128, 256) members.

Protocol	Library	Curve	Security Level (bits)	Public Key (Byte)	Multi-Signature (Byte)
RSMSP / RSMSP-zk	MCL	BN256	100	32	64
		BLS12-381	128	48	96
		BN384	128	48	96
RSMS-pop / RSMS-pop-zk	MCL	BN256	100	96	32
		BLS12-381	128	144	48
		BN384	128	144	48
ECDSA	Sodium	Ed25519	128	32	$64 \cdot n$
	OpenSSL	secp256k1	128	33	

Table 2: Signature size comparison between multi-signatures and ECDSA. The Multi-Signature column gives the size of the multi-signature that combines n signatures.

Protocol	Curve	Sign (ms)	VerifyMul (ms)			VerifyAgg (ms)		
			64	128	256	64×128	128×128	256×128
RSMSP	BN256	0.301	4.708	8.191	15.216	491.361	981.562	1963.130
	BLS12-381	0.311	4.824	8.249	15.235	498.063	1048.156	1971.246
	BN384	0.314	4.823	8.280	15.388	497.357	992.479	1972.311
RSMSP-zk	BN256	0.553	Same as above					
	BLS12-381	0.570						
	BN384	0.577						
RSMS-pop	BN256	0.172	1.153	1.193	1.275	48.674	98.982	198.468
	BLS12-381	0.178	1.229	1.267	1.352	51.117	103.633	207.195
	BN384	0.177	1.225	1.263	1.344	50.987	101.798	204.420
RSMS-pop-zk	BN256	0.463	Same as above					
	BLS12-381	0.474						
	BN384	0.474						
ECDSA	Ed25519	0.028	2.040	4.031	8.266	259.574	522.685	1053.570
	secp256k1	1.418	40.484	78.195	155.240	5203.854	10355.712	20790.631

Table 3: Performance comparison between multi-signatures and ECDSA. The Sign column gives the timings for generating an individual signture. The columns of VerifyMul give the timings for verifying a multi-signature signed by a threshold number of members from a group of size ℓ with $\ell = 64, 128, 256$. The threshold number is set to be $\lfloor \ell/2 \rfloor + 1$. That is, 33 (resp. 65, 129) members from a group of 64 (resp. 128, 256) members. The columns of VerifyAgg give the timings for verifying an aggregate signature signed by 64 (or 128, 256) groups where each group consists of 128 members and a threshold number (i.e., 65) of members signed.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, pages 416–432, 2003.

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. CCS '07, page 276–285, 2007.

[BN06] Mihir Bellare and Gregory Neven. Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. CCS '06, page 390–399, 2006.

[Bol02] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC 2003*, pages 31–46, 2002.

[Cel] Celo. Celo Blockchain. https://github.com/celo-org/celo-blockchain.

[Cod] Codechain. Codechain Foundry. https://github.com/CodeChain-io/foundry/blob/bls-consensus-signature/key/src/bls.rs.

[CP93] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO' 92*, pages 89–105, 1993.

[DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2093–2110. USENIX Association, 2020.

[DSD07] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In *Pairing*, pages 197–207, 2007.

[Elr19] Team Elrond. Elrond: A highly scalable public blockchain via adaptive state shardingand secure proof of stake, 2019. https://elrond.com/assets/files/elrond-whitepaper.pdf.

[eth] Ethereum 2.0 Specifications. https://github.com/ethereum/eth2.0-specs.

[GGJ+20] Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Asa Oines, Marek Olszewski, Michael Straka, Eran Tromer, and Psi Vesely. Plumo: Towards Scalable, Interoperable Blockchains Using Ultra Light Validation Systems. THE 3RD ZKPROOF WORKSHOP, 2020.

[GJNB11] C. C. F. Pereira Geovandro, Marcos A. Simplício Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[Har] Team Harmony. Technical Whitepaper - version 2.0. https://harmony.one/whitepaper.pdf.

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan L. Camenisch, editors, *EUROCRYPT 2004*, pages 74–90, 2004.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, pages 465–485, 2006.

[Mis19] Shigeo Mistunari. mcl - A portable and fast pairing-based cryptography library. https://github.com/herumi/mcl, 2019.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-Subgroup Multisignatures: Extended Abstract. CCS '01, page 245–254, 2001.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Des. Codes Cryptogr.*, 87:2139–2164, 2019.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *JOURNAL OF CRYPTOLOGY*, 13:361–396, 2000.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, pages 228–245, 2007.

[Shi19] Elaine Shi. Streamlined blockchains: A simple and elegant approach (A tutorial and survey). In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2019.

[Ten] Tendermint. Aggregate BLS signatures for votes. https://github.com/tendermint/tendermint/issues/1319.

Appendix A. Missing proofs in Section 4

Proof of Theorem 4.3. Suppose there exists an adversary $\mathcal{A}$ that $(\tau, q_S, q_H, \epsilon)$-breaks robustness of RSMSP-zk. We will show that this leads to a contradiction. At the end of the robustness game, the adversary outputs $(m^\star, \mathsf{gpk} = (\mathsf{gtag}^\star, \mathcal{PK}), \mathcal{E} = \{s_i\}_{i\in I})$ with $\mathcal{PK} = \{\mathsf{pk}_i\}_{i\in U}$ and $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $k \in U$. When the challenger outputs 1, we know $e(\sigma^\star, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{apk})$ where $s \leftarrow \mathsf{Sign}(m^\star, \mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{gtag}^\star)$, $(J, \sigma^\star) \leftarrow \mathsf{Combine}(m^\star, \mathcal{E} \cup \{s\}, \mathsf{gpk})$ and $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(J, \mathcal{PK})$. This means there exists $j \in J$ such that $s_j = (v_j, \pi_j)$ and $e(v_j, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{pk}_j)$. We consider two cases. If $j = k$, then this is impossible because $v_j = H_3(\mathsf{gtag}^\star, m^\star)^{\mathsf{sk}^\star}$ and $\mathsf{pk}_j = g_2^{\mathsf{sk}^\star}$. If $j \neq k$, from the way Combine is instantiated, we know that $\mathsf{VerEq}_{H_4}(g_2, \mathsf{pk}_j, H_3(\mathsf{gtag}^\star, m^\star), v_j, \pi_j) = 1$ for s_j. Using the forking lemma to rewind $\mathcal{A}$ we can obtain a witness sk'_j such that $v_j = H_3(\mathsf{gtag}^\star, m^\star)^{\mathsf{sk}'_j}$ and $\mathsf{pk}_j = g_2^{\mathsf{sk}'_j}$. This gives us $e(v_j, g_2) = e(H_3(\mathsf{gtag}^\star, m^\star), \mathsf{pk}_j)$ which contradicts to the hypothesis. □

Proof of Theorem 4.7. Suppose there exists an adversary $\mathcal{A}$ that $(\tau, q_S, q_H, \epsilon)$-breaks robustness of RSMS-pop. We will show that this leads to a contradiction. At the end of the robustness game, the adversary outputs $(m^\star, \mathsf{gpk}^\star = (\mathsf{gtag}^\star, \mathcal{PK}^\star = \{\mathsf{pk}_i\}_{i\in U}, \mathcal{E} = \{(\sigma_i, \pi_i)\}_{i\in I})$ with $\mathsf{pk}^\star = \mathsf{pk}_k$ for some $k \in U$. When the challenger outputs 1, the verification of $\sigma^\star$ fails, i.e., $e(\sigma^\star, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), \prod_{j\in J^\star\cup\{k\}} \mathsf{pk}_j)$ where $(v, \pi) \leftarrow \mathsf{Sign}(m^\star, \mathsf{sk}^\star, \mathsf{pk}^\star, \mathsf{gtag}^\star)$ and $(J, \sigma^\star) \leftarrow \mathsf{Combine}(m^\star, \mathcal{E} \cup \{(v, \pi)\}, \mathsf{gtag}^\star)$. This means there exists $j \in J$ such that $\mathsf{pk}_j = (pk_j, pop_j)$ and $e(v_j, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), pk_j)$. We consider two cases. If $j = k$, then this is impossible because $v_j = H_3(\mathsf{gtag}^\star, m^\star)^{\mathsf{sk}^\star}$ and $pk_j = g_2^{\mathsf{sk}^\star}$.

When $j \neq k$, we have $\mathsf{VerEq}_{H_4}(H_1(pk_j), pop_j, H_3(\mathsf{gtag}^\star, m^\star), \sigma_j, \pi_j) = 1$ for (v_j, π_j). Using the forking lemma to rewind $\mathcal{A}$ we can obtain a witness sk'_j such that $v_j = H_3(\mathsf{gtag}^\star, m^\star)^{\mathsf{sk}'_j}$ and $pop_j = H_1(pk_j)^{\mathsf{sk}'_j}$. Since $e(H_1(pk_j), pk_j) = e(pop_j, g_2)$, we can derive that $pk_j = g_2^{\mathsf{sk}'_j}$. This contradicts to the hypothesis $e(\sigma_j, g_2) \neq e(H_3(\mathsf{gtag}^\star, m^\star), pk_j)$. □

Appendix B. Missing proofs in Section 5

Proof of Theorem 5.4 (1). Given a $(\tau, q_S, q_H, \epsilon)$-forger $\mathcal{A}$ of ARSMSP, we construct an adversary $\mathcal{B}$ exactly the same as in the proof of Theorem 4.2, except that $\mathcal{A}$ outputs an aggregate signature forgery $(\{m_i, J_i, \mathsf{gpk}_i = (\mathsf{gtag}_i, \mathcal{PK}_i)\}_{i \in K}, \Sigma^\star)$ with $\mathsf{pk}^\star = \mathsf{pk}_{k^\star, j^\star}$ for some $k^\star \in K$ and $j^\star \in J_{k^\star}$. If $(\mathsf{gtag}_{k^\star}, m_{k^\star})$ is not the $\eta^\star$-th query to H_2, $\mathcal{B}$ aborts. The probability that $\mathcal{B}$ does not abort is at least $1/q_H$. Let's assume this is the case. Let $\mathsf{apk}_i \leftarrow \mathsf{KeyAgg}(J_i, \mathcal{PK}_i)$ for each $i \in K$. We have $e(\Sigma^\star, g_2) = \prod_{i \in K} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i) = e(H_3(\mathsf{gtag}_{k^\star}, m_{k^\star}), \mathsf{apk}_{k^\star}) \prod_{i \in K, i \neq k^\star} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$. $\mathcal{B}$ looks up the table $\mathcal{L}_{H_3}$ to find r_i such that $H_3(\mathsf{gtag}_i, m_i) = r_i$ for each $i \in K \setminus \{k^\star\}$. Then $\mathcal{B}$ uses the oracle $\mathcal{O}^\psi$ to compute $\sigma^\star = \Sigma \cdot \prod_{i \in K \setminus \{k^\star\}} \mathcal{O}^\psi(\mathsf{apk}_i^{-r_i})$. We can easily verify that $e(\sigma^\star, g_2) = e(H_3(\mathsf{gtag}_{k^\star}, m_{k^\star}), \mathsf{apk}_{k^\star})$. Now $\mathcal{B}$ obtains a RSMSP forgery and the rest of the proof is similar to Theorem 4.2. $\mathcal{B}$'s running time is increased by extra steps required to compute $\sigma^\star$, which in total is at most $\tau + (q_H + q_S + n) \cdot \tau_{\exp_1} + \ell \cdot \tau_{\exp_2}$ time. $\mathcal{B}$'s success probability is still ϵ / q_H. □

Proof of Theorem 5.4(2). The proof is similar as above, except that $\mathcal{B}$ needs to spend $2(\tau_{\exp_1} + \tau_{\exp_2})$ extra time to compute a simulated NIZK proofs to answer each Sign query. In total, $\mathcal{B}$ spends at most $\tau + (q_H + 3q_S + n) \cdot \tau_{\exp_1} + (2q_S + \ell) \cdot \tau_{\exp_2}$ time. □

Proof of Theorem 5.5(1). Given a $(\tau, q_S, q_H, \epsilon)$-forger $\mathcal{A}$ of ARSMS-pop, we construct an adversary $\mathcal{B}$ exactly the same as in the proof of Theorem 4.6, except that $\mathcal{A}$ outputs an aggregate signature forgery $(\{m_i, J_i, \mathsf{gpk}_i = (\mathsf{gtag}_i, \mathcal{PK}_i)\}_{i \in K}, \Sigma^\star)$ with $\mathsf{pk}^\star = \mathsf{pk}_{k^\star, j^\star}$ for some $k^\star \in K$ and $j^\star \in J_{k^\star}$. If $(\mathsf{gtag}_{k^\star}, m_{k^\star})$ is not the $\eta^\star$-th query to H_2, $\mathcal{B}$ aborts. The probability that $\mathcal{B}$ does not abort is at least $1/q_H$. Let's assume this is the case. Let $\mathsf{apk}_i \leftarrow \mathsf{KeyAgg}(J_i, \mathcal{PK}_i)$ for each $i \in K$. We have $e(\Sigma^\star, g_2) = \prod_{i \in K} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i) = e(H_3(\mathsf{gtag}_{k^\star}, m_{k^\star}), \mathsf{apk}_{k^\star}) \prod_{i \in K, i \neq k^\star} e(H_3(\mathsf{gtag}_i, m_i), \mathsf{apk}_i)$. $\mathcal{B}$ looks up the table $\mathcal{L}_{H_3}$ to find r_i such that $H_3(\mathsf{gtag}_i, m_i) = r_i$ for each $i \in K \setminus \{k^\star\}$. Then $\mathcal{B}$ uses the oracle $\mathcal{O}^\psi$ to compute $\sigma^\star = \Sigma \cdot \prod_{i \in K \setminus \{k^\star\}} \mathcal{O}^\psi(\mathsf{apk}_i^{-r_i})$. We can easily verify that $e(\sigma^\star, g_2) = e(H_3(\mathsf{gtag}_{k^\star}, m_{k^\star}), \mathsf{apk}_{k^\star})$. Now $\mathcal{B}$ obtains a RSMS-pop forgery and the rest of the proof is similar to Theorem 4.6. $\mathcal{B}$'s running time is increased by extra steps required to compute $\sigma^\star$, which in total is at most $\tau + (q_H + q_S + \ell + n) \cdot \tau_{\exp_1}$ time. $\mathcal{B}$'s success probability is still ϵ / q_H. □

Proof of Theorem 5.5(2). The proof is similar as above, except that $\mathcal{B}$ needs to spend $4 \cdot \tau_{\exp_1}$ extra time to compute a simulated NIZK proofs to answer each Sign query. In total, $\mathcal{B}$ spends at most $\tau + (q_H + 5q_S + \ell + n) \cdot \tau_{\exp_1}$ time. □