On the Security of Two-Round Multi-Signatures

Manu Drijvers[1,2], Kasra Edalatnejad[3], Bryan Ford[3], Eike Kiltz[4], Julian Loss[4], Gregory Neven[1], and Igors Stepanovs[5]

[1] DFINITY
[2] ETH Zurich
[3] EPFL
[4] Ruhr-University Bochum
[5] UCSD

Abstract. A multi-signature scheme allows a group of signers to collaboratively sign a message, creating a single signature that convinces a verifier that every individual signer approved the message. The increased interest in technologies to decentralize trust has triggered the proposal of highly efficient two-round Schnorr-based multi-signature schemes designed to scale up to thousands of signers, namely BCJ by Bagherzandi et al. (CCS 2008), MWLD by Ma et al. (DCC 2010), CoSi by Syta et al. (S&P 2016), and MuSig by Maxwell et al. (ePrint 2018). In this work, we point out serious security issues in all currently known two-round multi-signature schemes (without pairings). First, we prove that none of the schemes can be proved secure without radically departing from currently known techniques. Namely, we show that if the one-more discrete-logarithm problem is hard, then no algebraic reduction exists that proves any of these schemes secure under the discrete-logarithm or one-more discrete-logarithm problem. We point out subtle flaws in the published security proofs of the above schemes (except CoSi, which was not proved secure) to clarify the contradiction between our result and the existing proofs. Next, we describe practical sub-exponential attacks on all schemes, providing further evidence to their insecurity. Being left without two-round multi-signature schemes, we present mBCJ, a variant of the BCJ scheme that we prove secure under the discrete-logarithm assumption in the random-oracle model. Our experiments show that mBCJ barely affects scalability compared to CoSi, allowing 16384 signers to collaboratively sign a message in about 2 seconds, making it a highly practical and provably secure alternative for large-scale deployments.

1 Introduction

A multi-signature scheme allows a group of signers, each having their own key pair (pk_i, sk_i), to collaboratively sign a single message m. The result is a single signature σ that can be verified using the set of public keys $\{pk_1, \ldots, pk_n\}$, assuring a verifier that every signer approved message m. While multi-signature schemes have been studied for decades [IN83,OO93,MOR01,Bol03,LOS+06,BN06,BCJ08,MWLD10], they have recently received renewed interest because of the rise of distributed applications that aim to decentralize trust such as Bitcoin [Nak08] and more generally blockchain. Such applications typically involve many users or nodes that need to approve particular actions, which naturally matches the multi-signature setting where many signers must collaborate in order to create a joint multi-signature.

Motivated by such applications, Syta et al. [STV+16] presented the CoSi multi-signature scheme, a highly scalable multi-signature scheme that allows a tree of 8192 signers to sign in less than two seconds. Since its recent introduction, CoSi has already led to a large body of follow-up work, including a distributed protocol to create secure randomness [SJKK+17], improving the scalability of Bitcoin [SJKK+17], and introducing a decentralized software update framework [NKJ+17].

More recently, the Bitcoin community is actively looking into integrating Schnorr signatures as these could support multi-signatures and aggregate signatures, allowing many signatures that go into the same block to be merged into one, significantly reducing the overall size of the blockchain [bit17]. To this end, Maxwell et al. published the MuSig scheme [MPSW18a] that is tailored specifically to the needs of Bitcoin. The MuSig scheme was presented with a security proof under the one-more discrete-logarithm assumption, while the security of CoSi was never formally analyzed.

Scheme	KVf	KAg	Sign	Vf	Rounds	pk domain	Signature domain	PK domain	Security
BCJ1 [BCJ08]	$1\mathbb{G}^2$		$1\mathbb{G}^2 + 1\mathbb{G}^3$	$3\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{G}^2 \times \mathbb{Z}_q^3$	$\mathbb{G}$	Insecure
BCJ2 [BCJ08]			$1\mathbb{G} + 2\mathbb{G}^2$	$1\mathbb{G}^{n+1} + 2\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G}^3 \times \mathbb{Z}_q^3$	$\mathbb{G}$	Insecure
MWLD [MWLD10]			$1\mathbb{G}^2$	$1\mathbb{G}^{n+2}$	2	$\mathbb{G}$	$\mathbb{Z}_q^3$	$\mathbb{G}^n$	Insecure
CoSi [STV+16]	$1\mathbb{G}^2$		$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{Z}_q^2$	$\mathbb{G}$	Insecure
MuSig [MPSW18a]		$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}$	Insecure
mBCJ (this work)	$1\mathbb{G}^2$		$1\mathbb{G}^2 + 1\mathbb{G}^3$	$3\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{G}^2 \times \mathbb{Z}_q^3$	$\mathbb{G}$	DL, ROM
BN [BN06]			$1\mathbb{G}$	$1\mathbb{G}^{n+1}$	3	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}^n$	DL, ROM
BDN-MSDL [BDN18,MPSW18b]		$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	3	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}$	DL, ROM
B-Pop [Bol03,RY07]	$2P$		$1\mathbb{G}_1$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$	co-CDH, ROM
WM-Pop [LOS+06,RY07]	$2P$		$1\mathbb{G}_1 + 1\mathbb{G}_2$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_T$	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_T$	co-CDH
BDN-MSP [BDN18]		$1\mathbb{G}_2^n$	$1\mathbb{G}_1$	$2P$	1	$\mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$	co-CDH, ROM

Table 1. Efficiency of multi-signatures in the key verification model. The first block of schemes depicts all known two-round multi-signature schemes without pairings. For completeness, we also include existing three-round schemes as well as non-interactive pairing-based schemes in the second and third blocks. Columns 2–5 show the computational efficiency of the individual algorithms by counting the number of (multi-)exponentiations and pairings, where "$\mathbb{G}$" denotes an exponentiation in group $\mathbb{G}$, "$\mathbb{G}^k$" denotes an k-multi-exponentiation in group $\mathbb{G}$, "P" denotes a pairing operation, and n denotes the number of signers. Column 6 shows the number of communication rounds and columns 7–9 show the size of the individual signer's public key, the signature, and the aggregated public key, respectively, where any "proof-of-possession" of the secret key is considered to be part of the public key, and q denotes the order of the groups. Column 10 shows the assumptions under which the scheme is proved secure, if any, where "ROM" indicates a proof in the random-oracle model.

One of the main problems when designing provably secure Schnorr-based multi-signature schemes is that in order to simulate the honest signer, the reduction cannot simply use the zero-knowledge property and program the random oracle, because the random-oracle entry that must be programmed depends on the output of the adversarial signers. Bellare and Neven [BN06] got around this issue by introducing a preliminary round in the signing protocol where signers exchange commitments to their first rounds. Bagherzandi et al.'s BCJ scheme [BCJ08] eliminated the need for this extra round by using homomorphic trapdoor commitments, while Ma et al.'s MWLD scheme [MWLD10] simulates signatures by exploiting the witness-indistinguishability of Okamoto signatures [Oka93].

Impossibility result. Our first result essentially shows that none of the existing two-round schemes can be proved secure under standard assumptions. More precisely, we prove that if the one-more discrete logarithm problem (OMDL) is hard, then there cannot exist an algebraic black-box reduction that proves the CoSi, MuSig, BCJ, or MWLD schemes secure under the discrete logarithm (DL) or OMDL assumption.

This is surprising, because all of these schemes, barring CoSi, were published with a security proof under the DL (BCJ, MWLD) or OMDL (MuSig) assumption. We explain the obvious contradiction by pointing out subtle flaws in their proofs. The problem is that simulating signing queries in combination with a rewinding argument (so-called *forking* [PS00]) is especially delicate, because the forger may be forked at a point where it has an "open" signing query. If that is the case, the reduction has to come up with a second response for the same first round of the signing protocol, which leaks the signing key that it was hoping to extract from the forger. The actual impossibility proof is a bit more involved, but it exploits exactly this difficulty in simulating signing queries.

The class of reductions covered by our result essentially encompasses all currently known proof techniques, including those that rewind the adversary an arbitrary number of times. Also, given that all of the covered schemes are derived from Schnorr signatures [Sch91], it would be very surprising if their security could be proved under an assumption that is not implied by DL or OMDL. So while in theory our result does not completely rule out the existence of a security proof, in practice it does mean that a security proof under standard assumptions is extremely unlikely as it would have to use currently unknown techniques—unless OMDL turns out to be easy while DL is still hard, in which case our result becomes moot.

Sub-exponential attacks. If the above impossibility result still leaves a glimmer of hope that the existing schemes might be safe for practical use, our second result clearly indicates that this is not the case. Namely, we provide attacks that apply to all schemes based on Wagner's algorithm for the generalized birthday problem [Wag02]. Because the attacks run in sub-exponential time, but not in polynomial time, they don't supersede our impossibility result, but for reasonable parameter choices they are efficient enough to form a realistic threat. We show that an attacker performing $\ell - 1$ concurrent signing queries can create a forgery in $O(\ell \cdot 2^{\lg q/(1+\lg \ell)})$ time and space, where q is the order of the group. For practical values such as $\lg q = 256$, this means that even for 15 concurrent signing queries, a forgery can be computed in 2^{62} steps, which is feasible on modern clusters. For 127 concurrent queries, one can compute a forgery in less than 2^{45} steps. We describe our attacks for CoSi, but they also extend to MuSig, BCJ, and MWLD, as well as to important applications of CoSi such as the RandHound scheme from [SJKK+17]. While asymptotically speaking these attacks can be thwarted by increasing the group order, doing so will render the schemes impractical, and even then provable security remains an elusive goal due to our impossibility result.

A secure two-round multi-signature scheme. Our negative results apply to all known two-round multi-signature schemes without pairings. There are pairing-based non-interactive multi-signature schemes from BLS signatures [BLS01,Bol03,RY07,BDN18] and from Waters signatures [Wat05,LOS+06], but one may wonder whether secure two-round multi-signatures without pairings exist at all. We answer this question in the positive by presenting mBCJ, a variant of the BCJ scheme that uses the same homomorphic commitment scheme to let signers commit to their first rounds as the BCJ scheme, but lets the commitment parameters be determined by the output of a random oracle applied to the message being signed. We provide a rigorous security proof of the mBCJ scheme under the DL assumption in the random-oracle model.

Table 1 summarizes the efficiency and security of multi-signature schemes. Our mBCJ scheme is the only secure two-round multi-signature scheme. Even though it is less efficient in terms of signature size and verification time than some of its (insecure) precedents, it does have the advantage of supporting key aggregation. For applications that involve large numbers of signers, it is crucial that verification time can be kept constant (i.e., independent of the number of signers) by verifying against a constant-size aggregate public key. The verifier computes the aggregate public key once from the set of individual public keys to have (amortized) constant-time verification afterwards.

One may wonder what the cost is of provable security by comparing the mBCJ scheme against the highly efficient CoSi scheme. To investigate the real-world effects of this difference, we performed large-scale experiments on prototype implementations of both schemes. For a network round-trip delay of 200 milliseconds, we found that a group as large as 16,384 signers can collaboratively sign a message using mBCJ in about 2 seconds, showing no significant difference with CoSi. The mBCJ scheme on average needs 73% more CPU time than CoSi for small amounts of signers. For large amounts of signers the difference becomes smaller, and the average CPU time per signer remains under 0.25 milliseconds when signing with 16,384 signers. Our results show that mBCJ is only marginally less efficient than CoSi, so that any protocol based on the insecure CoSi scheme should instead be built on the provably secure mBCJ scheme.

Restricted security of CoSi. Our impossibility proof and attacks on CoSi crucially rely on the adversary being able to engage in concurrent queries with the signing oracle. A natural question to ask is whether CoSi could still be secure when the adversary is limited to sequential signing queries. We show that this is indeed the case, but only in the weaker knowledge of secret key (KOSK) model. More specifically, we prove CoSi secure against sequential attacks (or even logarithmically many concurrent attacks) under the OMDL assumption in the KOSK setting in the random-oracle model. Hence, one could still consider using CoSi in very specific scenarios where the KOSK setting can be considered reasonable and where one can naturally limit the number of parallel signing sessions to a small constant.

Versions of this paper. An earlier version of this paper [DEFN18a] contained the impossibility proof for the CoSi and MuSig schemes, and incorrectly suggested a new scheme called DG-CoSi as a provably secure alternative. A second version [DEFN18b] added the BCJ and MWLD schemes (Sections 3.4 and 3.3) to the list of schemes covered by our meta-reduction, and it is not hard to see that the meta-reduction applies to DG-CoSi as well. The proof of DG-CoSi in the earlier version contained a flawed argument saying that, because each individual execution of the adversary is independent of the simulator's choice for the signing

key, then the key computed from the outputs of the adversary must be independent from the simulator's choice as well. This is not correct, because an adversary could base its output on signing oracle responses to cause the reduction to extract the key that it already knows. We are very grateful to an anonymous referee for catching this mistake.

This version adds sub-exponential attacks on all currently known two-round multi-signature schemes in Section 4, pointing out the danger of using these schemes in the real world. It also adds the new mBCJ scheme and the restricted security proof of CoSi in Section 6.

2 Preliminaries

2.1 Discrete Logarithm Problems

Definition 1 (Discrete Log Problem). *For a group $\mathbb{G} = \langle g \rangle$ of prime order q, we define $\mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}$ of an adversary $\mathcal{A}$ as*

$$\Pr\Big[y = g^x : y \xleftarrow{\$} \mathbb{G}, x \xleftarrow{\$} \mathcal{A}(y)\Big],$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of y. $\mathcal{A}$ (τ, ϵ)-breaks the discrete log problem if it runs in time at most τ and has $\mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}} \geq \epsilon$. Discrete log is (τ, ϵ)-hard if no such adversary exists.

Definition 2 (n-One-More Discrete Log Problem [BNPS03,BP02]). *For a group $\mathbb{G} = \langle g \rangle$ of prime order q, let $\mathcal{O}^{\mathtt{dlog}}(\cdot)$ be a discrete logarithm oracle that can be called at most n times. We define $\mathsf{Adv}^{\mathsf{n\text{-}omdl}}_{\mathbb{G}}$ of an adversary $\mathcal{A}$ as*

$$\Pr\Big[\bigwedge_{i=0}^{n} y_i = g^{x_i} : (y_0, \ldots, y_n) \xleftarrow{\$} \mathbb{G}^{n+1}, (x_0, \ldots, x_n) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathtt{dlog}}(\cdot)}(y_0, \ldots, y_n)\Big],$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of $y_0, \ldots, y_n$. $\mathcal{A}$ (τ, ϵ)-breaks the n-one-more discrete log problem if it runs in time at most τ and has $\mathsf{Adv}^{\mathsf{n\text{-}omdl}}_{\mathbb{G}} \geq \epsilon$. n-one-more discrete log is (τ, ϵ)-hard if no such adversary exists.

2.2 Algebraic Algorithms

Boneh and Venkatesan [BV98] define *algebraic* algorithms to study the relation between breaking RSA and factoring. An algorithm working in some group is algebraic if it only uses the group operations to construct group elements. More precisely, it can test equality of two group elements, perform the group operation on two elements to obtain a new element, and invert a group element. This means that an algebraic algorithm that receives group elements $y_1, \ldots, y_n$ as input can only construct new group elements y for which it knows $\alpha_1, \ldots, \alpha_n$ such that $y = \prod_{i=1}^{n} y_i^{\alpha_i}$.

We use the formalization by Paillier and Vergnaud [PV05]:

Definition 3. *An algorithm A that on input group elements $(y_1, \ldots, y_n)$ is* algebraic *if it admits a polynomial time algorithm* Extract *that given the code of A and its random tape outputs $(\alpha_1, \ldots, \alpha_n)$ such that $h = \prod_{i=1}^{n} y_i^{\alpha_i}$ for any group element h that A outputs.*

2.3 Generalized Forking Lemma

The original forking lemma was formulated by Pointcheval and Stern [PS00] to analyze the security of Schnorr signatures [Sch91]. The lemma rewinds a forger $\mathcal{A}$ against the Schnorr signature scheme in the random-oracle model (ROM) to a "crucial" random-oracle query (typically, the query involved in a forgery) and runs $\mathcal{A}$ again from the crucial query with fresh random-oracle responses. The lemma basically says that if $\mathcal{A}$ has

non-negligible success probability in a single run, then the forking algorithm will generate two successful executions with non-negligible probability.

Bellare and Neven [BN06] generalized the forking lemma to apply to any algorithm $\mathcal{A}$ in the random-oracle model using a single rewinding, while Bagherzandi, Cheon, and Jarecki [BCJ08] generalized the lemma even further to multiple subsequent rewindings on multiple crucial queries. We recall here a slight modification of the latter version.

Let $\mathcal{A}$ be an algorithm that is given an input in as well as randomness $f = (\rho, h_1, \ldots, h_{q_\mathrm{H}})$, where ρ is $\mathcal{A}$'s random tape and $h_1, \ldots, h_{q_\mathrm{H}}$ are random values from $\mathbb{Z}_q$. 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 with access to oracle $\mathcal{O}$, denoted $\mathcal{A}^{\mathcal{O}}(in, f)$, as *successful* if it outputs a tuple $(J, \{out_j\}_{j\in J}, aux)$, where J is a multi-set that is a non-empty subset of $\{1, \ldots, q_\mathrm{H}\}$, $\{out_j\}_{j\in J}$ is a multi-set of side outputs, and aux is an additional set of auxiliary outputs. We say that $\mathcal{A}$ failed if it outputs $J = \emptyset$. Let ϵ be the probability that $\mathcal{A}(in, f)$ is successful for fresh randomness $f \xleftarrow{\$} \Omega$ and for an input $in \xleftarrow{\$} \mathsf{IG}$ generated by an input generator IG.

For a given input in, the generalized forking algorithm $\mathcal{GF}_\mathcal{A}$ is defined as follows:

$\mathcal{GF}_\mathcal{A}(in)$:
  $f = (\rho, h_1, \ldots, h_{q_\mathrm{H}}) \xleftarrow{\$} \Omega$
  $(J, \{out_j\}_{j\in J}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(in, f)$
  If $J = \emptyset$ then output `fail`
  $Aux \leftarrow aux$
  Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \le \ldots \le j_n$
  For $i = 1, \ldots, n$ do
    $succ_i \leftarrow 0$; $k_i \leftarrow 0$; $k_{\max} \leftarrow 8nq_\mathrm{H}/\epsilon \cdot \ln(8n/\epsilon)$
    Repeat until $succ_i = 1$ or $k_i > k_{\max}$
      $f'' \xleftarrow{\$} \Omega$ such that $f''|_{j_i} = f|_{j_i}$
      Let $f'' = (\rho, h_1, \ldots, h_{j_i-1}, h''_{j_i}, \ldots, h''_{q_\mathrm{H}})$
      $(J'', \{out''_j\}_{j\in J''}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(in, f'')$
      $Aux \leftarrow Aux \cup aux$
      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 $(J, \{out_j\}_{j\in J}, \{out'_j\}_{j\in J}, Aux)$
  Else output `fail`

We say that $\mathcal{GF}_\mathcal{A}$ succeeds if it doesn't output `fail`. The main difference to Bagherzandi et al.'s forking lemma [BCJ08] is $\mathcal{A}$'s access to the oracle $\mathcal{O}$ and the additional auxiliary output aux that gets accumulated in Aux over all runs of $\mathcal{A}$, including failed runs. If the oracle $\mathcal{O}$ is deterministic, meaning that it always answers the same query with the same response, it is easy to see that these extensions do not impact the bounds of their forking lemma, so the following statement continues to hold.

Lemma 1 (Generalized Forking Lemma [BCJ08]). *Let IG be a randomized algorithm and $\mathcal{A}$ be a randomized algorithm running in time τ with access to a deterministic oracle $\mathcal{O}$ that succeeds with probability ϵ. If $q > 8nq_\mathrm{H}/\epsilon$, then $\mathcal{GF}_\mathcal{A}(in)$ runs in time at most $\tau \cdot 8n^2 q_\mathrm{H}/\epsilon \cdot \ln(8n/\epsilon)$ and succeeds with probability at least $\epsilon/8$, where the probability is over the choice of $in \xleftarrow{\$} \mathsf{IG}$ and over the coins of $\mathcal{GF}_\mathcal{A}$.*

2.4 Security of Multi-Signatures

We follow the syntax and security model due to Bagherzandi et al. [BCJ08], which follows the so-called key-verification model, as introduced by Bagherzandi and Jarecki [BJ08], where individual public keys must be verified by the signature verifier. We adapt the model to support signers that are organized in a tree structure for more efficient communication. Prior work always assumed a communication setting where every cosigner communicates directly with the initiator, which our tree-based modeling supports by choosing a tree in which

every cosigner is a direct child of the initiator. Moreover, we formalize the notion of an "aggregated key" of a group of signers, by adding an algorithm that computes a single aggregated public key from a set of public keys, and this aggregated key will be used by the verification algorithm. The idea of splitting key aggregation from verification is that if a group of signers will repeatedly sign together, a verifier will only once compute the aggregate public key and reuse that for later verifications. If the aggregated key is smaller than the set of public keys, or even constant size, this will allow for more efficient schemes. Note that this change does not exclude multi-signature schemes that do not have this feature: indeed, such schemes can simply use the identity function as key aggregation algorithm.

A multi-signature scheme consists of algorithms Pg, Kg, Sign, KAg, KVf, and Vf. A trusted party generates the system parameters $par \leftarrow \mathsf{Pg}$. Every signer generates a key pair $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(par)$, and signers can collectively sign a message m by each calling the interactive algorithm $\mathsf{Sign}(par, sk, \mathcal{T}, m)$, where $\mathcal{T}$ describes a tree between the signers that defines the intended communication between the signers. At the end of the protocol, the root of the tree $\mathcal{T}$ obtains a signature σ. Algorithm KAg on input parameters par and a set of public keys $\mathcal{PK}$ outputs a single aggregate public key PK. A verifier can check the validity of a signature σ on message m under an aggregate public key PK by running $\mathsf{Vf}(par, PK, m, \sigma)$ which outputs 0 or 1 indicating that the signatures is invalid or valid, respectively. Anybody can check the validity of a public key by using key verification algorithm $\mathsf{KVf}(par, pk)$.

First, a multi-signature scheme should satisfy completeness, meaning that 1) for any $par \leftarrow \mathsf{Pg}$ and any $(pk, sk) \leftarrow \mathsf{Kg}(par)$, we have $\mathsf{KVf}(par, pk) = 1$, and 2) for any n, if we have $(pk_i, sk_i) \leftarrow \mathsf{Kg}(par)$ for $i = 1, \ldots, n$, and any tree $\mathcal{T}$ containing exactly these n signers, and for any message m, if all signers input $\mathsf{Sign}(par, sk_i, \mathcal{T}, m)$, then the root of $\mathcal{T}$ will output a signature σ such that $\mathsf{Vf}(par, \mathsf{KAg}(par, \{pk_i\}_{i=1}^n), m, \sigma) = 1$.

Second, a multi-signature scheme should satisfy unforgeability. Unforgeability of a multi-signature scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{KAg}, \mathsf{Vf}, \mathsf{KVf})$ is defined by a three-stage game.

Setup. The challenger generates the parameters $par \leftarrow \mathsf{Pg}$ and a challenge key pair $(pk^*, sk^*) \xleftarrow{\$} \mathsf{Kg}(par)$. It runs the adversary on the public key $\mathcal{A}(par, pk^*)$.

Signature queries. $\mathcal{A}$ is allowed to make signature queries on a message m with a tree $\mathcal{T}$, meaning that it has access to oracle $\mathcal{O}^{\mathsf{Sign}(par, sk^*, \cdot, \cdot)}$ that will simulate the honest signer interacting in a signing protocol to sign message m with intended communication tree $\mathcal{T}$. Note that $\mathcal{A}$ may make any number of such queries concurrently.

Output. Finally, the adversary halts by outputting a multi-signature forgery σ, a message m and a set of public keys $\mathcal{PK}$. In the key-verification setting, the adversary wins if $pk^* \in \mathcal{PK}$, $\mathsf{KVf}(par, pk) = 1$ for every $pk \in \mathcal{PK}$ with $pk \neq pk^*$, $PK \leftarrow \mathsf{KAg}(par, \mathcal{PK})$, $\mathsf{Vf}(par, PK, \sigma, m) = 1$, and $\mathcal{A}$ made no signing queries on m. A special case of the key-verification model is the plain public key model, where there is no need to verify individual public keys, i.e., KVf always returns 1.[6] In the weaker knowledge-of-secret-key (KOSK) setting, the adversary is required to additionally output corresponding secret keys sk_{pk} for all $pk \in \mathcal{PK}, pk \neq pk^*$.

Definition 4. *We say $\mathcal{A}$ is a $(\tau, q_\mathrm{S}, q_\mathrm{C}, q_\mathrm{H}, \epsilon)$-forger for multi-signature scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ if it runs in time τ, makes q_S signing queries such that at most q_C signing protocols are concurrently active (i.e., started but not yet finished) at any given time, makes q_H random oracle queries, and wins the above game with probability at least ϵ. MS is $(\tau, q_\mathrm{S}, q_\mathrm{C}, q_\mathrm{H}, \epsilon)$-unforgeable if no $(\tau, q_\mathrm{S}, q_\mathrm{C}, q_\mathrm{H}, \epsilon)$-forger exists.*

2.5 The CoSi Multi-Signature Scheme

CoSi is a multi-signature scheme introduced by Syta et al. [STV+16] that follows a long line of work on Schnorr-based multi-signatures [MOR01,BN06,RY07,BCJ08,MWLD10]. With a two-round signing protocol

[6] The distinction between the key-verification model and plain public key model is a bit informal, as they are in fact equivalent: any multi-signature scheme that is unforgeable in the key-verification model is also secure in the plain public key model, where the key verification is simply considered part of the verification algorithm.

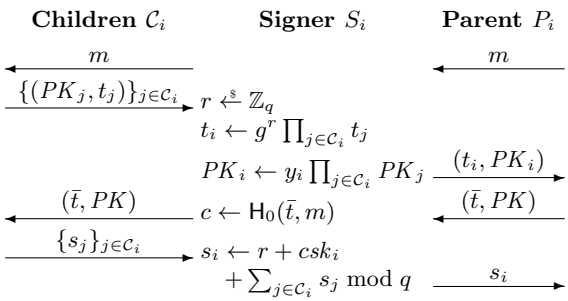

Fig. 1. The CoSi signing protocol for signer S_i with secret key sk_i and public key $pk = (y_i, \pi_i)$. If S_i is the leader then, instead of sending (t_i, PK_i) to its parent, it sends $(\bar{t}, PK) = (t_i, PK_i)$ to its children, and instead of sending s_i to its parent, it outputs $(c, s) = (c, s_i)$ as the signature.

and verification as efficient as verifying a single Schnorr signature, CoSi is more efficient than prior schemes, but it was proposed without a formal security proof. By organizing signers in a tree structure, thousands of signers can create a multi-signature in seconds, as demonstrated by the open source implementation.[7] CoSi has already led to a large body of follow-up work [KJG+16,SJKK+17,BKJ+17,NKJ+17,KKJG+18] since its recent introduction.

Below, we describe a slight variant of the original CoSi scheme where we add a proof of possession to the public key, and where during signing the cosigners receive the final commitment value $\bar{t}$ instead of just the hash $c = \mathsf{H}(\bar{t}, m)$. The first change was already suggested in the original paper, the latter is necessary because the scheme is insecure without it, since signers are unable to ensure that they are signing the message that they intended to sign.

Parameters generation. The Pg algorithm sets up a group $\mathbb{G} = \langle g \rangle$ of order q, where q is a κ-bit prime. It also chooses two hash functions $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \mathbb{Z}_q$. Output $par \leftarrow (\mathbb{G}, g, q, \mathsf{H}_0, \mathsf{H}_1)$.

Key generation. The key generation algorithm $\mathsf{Kg}(par)$ samples $sk \xleftarrow{\$} \mathbb{Z}_q$ and computes $y \leftarrow g^{sk}$. It then generates a proof of possession $\pi = (c, s)$ by choosing $r \xleftarrow{\$} \mathbb{Z}_q$ and computing $s \leftarrow r + \mathsf{H}_1(g^r, y)$. Output $pk \leftarrow (y, \pi)$ and sk.

Signing. Signing is a four-step protocol. A signer S_i on input $\mathsf{Sign}(par, sk_i, m, \mathcal{T})$ behaves as follows.
Announcement. If S_i is the leader (i.e., the root of tree $\mathcal{T}$), it initiates the protocol by sending an announcement to its children, which consists of a unique identifier for this signing session $ssid$. If S_i is not the leader, it waits to receive an announcement message and forwards it to its children in $\mathcal{T}$. After doing so, S_i proceeds with the commitment phase.
Commitment. Let $\mathcal{C}_i$ denote the set of children of S_i in tree $\mathcal{T}$. S_i waits to receive all values (t_j, PK_j) for $j \in \mathcal{C}_i$. Note that if S_i has no children (i.e., it is a leaf in tree $\mathcal{T}$), it will proceed immediately. S_i chooses $r_i \xleftarrow{\$} \mathbb{Z}_q$ and computes $t_i \leftarrow g^{r_i} \cdot \prod_{j \in \mathcal{C}_i} t_j$ and $PK_i \leftarrow y_i \cdot \prod_{j \in \mathcal{C}_i} PK_j$ for $pk_i = (y_i, \pi_i)$. If S_i is not the leader, it sends t_i to its parent. If S_i is the leader, S_i proceeds with the challenge phase.
Challenge. If S_i is the leader, it sets $\bar{t} \leftarrow t_i$ and $PK \leftarrow PK_i$, computes $c \leftarrow \mathsf{H}_0(\bar{t}, m)$, and sends $(\bar{t}, PK)$ to its children. If S_i is not the leader, it waits to receive a message $(\bar{t}, PK)$, computes $c \leftarrow \mathsf{H}_0(\bar{t}, m)$, and sends $(\bar{t}, PK)$ to its children.
Response. S_i waits to receive all values s_j for $j \in \mathcal{C}_i$ (note that if S_i is a leaf it will proceed immediately), and then computes $s_i \leftarrow r_i + c \cdot sk_i + \sum_{j \in C_i} s_j$. It sends s_i to its parent, unless S_i is the root, then S_i sets $s \leftarrow s_i$ and outputs $\sigma \leftarrow (c, s)$.

Key Aggregation. On input a set of public keys $\mathcal{PK}$, KAg checks for all $(y, (c, s)) \in \mathcal{PK}$ that $c = \mathsf{H}_1(g^s y^{-c}, y)$. If so, it outputs aggregate public key $PK \leftarrow \prod_{(y,\pi) \in \mathcal{PK}} y$.

Verification. Vf on input an aggregate public key PK, a signature $\sigma = (c, s)$, and a message m, checks that

$$c \stackrel{?}{=} \mathsf{H}_0\left(g^s \cdot PK^{-c}, m\right) \ .$$

[7] The implementation is available at github.com/dedis/cothority.

2.6 The BCJ Multi-Signature Schemes

Whereas Bellare and Neven [BN06] solved the problem of signature simulation in the security proof by letting all signers commit to their contribution in a preliminary round of the signing protocol, the BCJ scheme due to Bagherzandi et al. [BCJ08] avoids this extra round by using a multiplicatively homomorphic equivocable commitment scheme. Since the only known instantiation of such a commitment scheme is included in the same paper, we describe the BCJ scheme for that instantiation here. We also adapt the scheme to support signers in a tree structure.

Parameters generation. Pg sets up a group $\mathbb{G}$ of order q with generators g_1, h_1, g_2, and h_2, where q is a κ-bit prime. Choose hash functions $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$ and output $par \leftarrow (\mathbb{G}, g_1, h_1, g_2, h_2, q, \mathsf{H}_0, \mathsf{H}_1)$.

Key generation. The key generation algorithm $\mathsf{Kg}(par)$ takes $sk \xleftarrow{\$} \mathbb{Z}_q$ and sets $y \leftarrow g_1^{sk}$. Compute proof-of-posession $\pi = (c, s)$ by taking $r \xleftarrow{\$} \mathbb{Z}_q$, $c \leftarrow \mathsf{H}_1(y, g_1^r)$, $s \leftarrow r + c \cdot sk$. Let $pk \leftarrow (y, \pi)$ and output (pk, sk).

Signing. Signing is the four-step protocol. A signer S_i on input $\mathsf{Sign}(par, sk_i, m, \mathcal{T})$ behaves as follows.
Announcement. If S_i is the leader (i.e., the root of tree $\mathcal{T}$), it initiates the protocol by sending an announcement to its children, which consists of a unique identifier for this signing session $ssid$. If S_i is not the leader, it waits to receive an announcement message and forwards it to its children in $\mathcal{T}$. After doing so, S_i proceeds with the commitment phase.
Commitment. Let $\mathcal{C}_i$ denote the set of children of S_i in tree $\mathcal{T}$. S_i waits to receive all values $(t_{j,1}, t_{j,2}, PK_j)$ for $j \in \mathcal{C}_i$. Note that if S_i has no children (i.e., it is a leaf in tree $\mathcal{T}$), it will proceed immediately. S_i chooses $(r_i, \alpha_{i,1}, \alpha_{i,2}) \xleftarrow{\$} \mathbb{Z}_q^3$ and computes $t_{i,1} \leftarrow g_1^{\alpha_{i,1}} h_1^{\alpha_{i,2}} \cdot \prod_{j \in \mathcal{C}_i} t_{j,1}$ and $t_{i,2} \leftarrow g_2^{\alpha_{i,1}} h_2^{\alpha_{i,2}} \cdot g_1^{r_i} \cdot \prod_{j \in \mathcal{C}_i} t_{j,2}$, and $PK_i \leftarrow y_i \cdot \prod_{j \in \mathcal{C}_i} PK_j$. If S_i is not the leader, it sends $(t_{i,1}, t_{i,2}, PK_i)$ to its parent. If S_i is the leader, S_i proceeds with the challenge phase.
Challenge. If S_i is the leader, it sets $\bar{t}_1 \leftarrow t_{i,1}$, $\bar{t}_2 \leftarrow t_{i,2}$, and $PK \leftarrow PK_i$. It computes $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, and sends $(\bar{t}_1, \bar{t}_2, PK)$ to its children. If S_i is not the leader, it waits to receive a message $(\bar{t}_1, \bar{t}_2, PK)$, computes $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, and sends $(\bar{t}_1, \bar{t}_2, PK)$ to its children.
Response. S_i waits to receive all values $(s_j, \gamma_{j,1}, \gamma_{j,2})$ for $j \in \mathcal{C}_i$ (note that if S_i is a leaf it will proceed immediately), and then computes $s_i \leftarrow r_i + c \cdot sk_i + \sum_{j \in C_i} s_j$, $\gamma_{i,b} \leftarrow \alpha_{i,b} + \sum_{j \in C_i} \gamma_{j,b}$ for $b \in \{1,2\}$. It sends $(s_i, \gamma_{i,1}, \gamma_{i,2})$ to its parent, unless S_i is the root, then S_i sets $s \leftarrow s_i$, $\gamma_1 \leftarrow \gamma_{i,1}$, $\gamma_2 \leftarrow \gamma_{i,2}$, and outputs $\sigma \leftarrow (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$.

Key Aggregation. KAg on input a set of public keys $\mathcal{PK}$ parses every $pk_i \in \mathcal{PK}$ as $(y_i, (c_i, s_i))$, and if this public key has not been validated before, check that $c_i = \mathsf{H}_1(y_i, g_1^{s_i} y_i^{-c_i})$. Output aggregate public key $PK \leftarrow \prod_{(y,\pi) \in \mathcal{PK}} y$.

Verification. Vf on input aggregate public key PK, a signature $\sigma = (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$, and a message m, compute $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$ and check that $\bar{t}_1 \stackrel{?}{=} g_1^{\gamma_1} h_1^{\gamma_2}$ and $\bar{t}_2 \stackrel{?}{=} g_2^{\gamma_1} h_2^{\gamma_2} g_1^{s} PK^{-c}$.

2.7 The MWLD Multi-Signature Scheme

The MWLD scheme due to Ma et al. [MWLD10] addresses the signature simulation problem by using a witness-indistinguishable proof based on Okamoto signatures [Oka93], yielding shorter signatures and more efficient signing than the BCJ scheme. We present a slightly modified scheme here to support signers organized in a tree structure.

Parameters generation. Pg sets up a group $\mathbb{G}$ of order q with generators g and h, where q is a κ-bit prime. Choose two hash functions $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$ and output $par \leftarrow (\mathbb{G}, g, h, q)$.

Key generation. The key generation algorithm $\mathsf{Kg}(par)$ takes $(sk_1, sk_2) \xleftarrow{\$} \mathbb{Z}_q^2$ and sets $pk \leftarrow g^{sk_1} h^{sk_2}$.

Signing. Signing is the four-step protocol. A signer S_i on input $\mathsf{Sign}(par, (sk_1, sk_2), m, \mathcal{T})$ behaves as follows.
Announcement. If S_i is the leader (i.e., the root of tree $\mathcal{T}$), it initiates the protocol by sending an announcement to its children, which consists of a unique identifier for this signing session $ssid$. If S_i is not the leader,

it waits to receive an announcement message and forwards it to its children in $\mathcal{T}$. After doing so, S_i proceeds with the commitment phase.
Commitment. Let $\mathcal{C}_i$ denote the set of children of S_i in tree $\mathcal{T}$. S_i waits to receive all values (t_j, L_j) for $j \in \mathcal{C}_i$. Note that if S_i has no children (i.e., it is a leaf in tree $\mathcal{T}$), it will proceed immediately. S_i chooses $(r_{i,1}, r_{i,2}) \xleftarrow{\$} \mathbb{Z}_q^2$ and computes $t_i \leftarrow g^{r_{i,1}} h^{r_{i,2}} \cdot \prod_{j \in \mathcal{C}_i} t_j$ and $L_i \leftarrow \{pk_i\} \cup (\bigcup_{j \in \mathcal{C}_i} L_j)$. If S_i is not the leader, it sends (t_i, L_i) to its parent. If S_i is the leader, S_i proceeds with the challenge phase.
Challenge. If S_i is the leader, it sets $\bar{t} \leftarrow t_i$ and $L \leftarrow L_i$, computes $c \leftarrow \mathsf{H}_0(\bar{t}, L, m)$, and sends $(\bar{t}, L)$ to its children. If S_i is not the leader, it waits to receive a message $(\bar{t}, L)$, computes $c \leftarrow \mathsf{H}_0(\bar{t}, L, m)$, and sends $(\bar{t}, L)$ to its children. *Response.* S_i waits to receive all values $(s_{j,1}, s_{j,2})$ for $j \in \mathcal{C}_i$ (note that if S_i is a leaf it will proceed immediately), and then computes $v_i = \mathsf{H}_1(c, pk_i)$ and $s_{i,b} \leftarrow r_{i,b} + v_i \cdot sk_{i,b} + \sum_{j \in C_i} s_{j,b}$ for $b \in \{1, 2\}$. It sends $(s_{i,1}, s_{i,2})$ to its parent, unless S_i is the root, then S_i sets $s_1 \leftarrow s_{i,1}$, $s_2 \leftarrow s_{i,2}$, and outputs $\sigma \leftarrow (c, s_1, s_2)$.

Key Aggregation. This scheme does not support a compressed public key, i.e., $\mathsf{KAg}(par, \mathcal{PK}) = \mathcal{PK}$.

Verification. Vf on input a set of public keys $\mathcal{PK}$, a signature $\sigma = (c, s_1, s_2)$, and a message m, checks that

$$c \stackrel{?}{=} \mathsf{H}_0\left(g^{s_1} h^{s_2} \cdot \prod_{pk_i \in \mathcal{PK}} pk_i^{-\mathsf{H}_1(c, pk_i)}, \mathcal{PK}, m\right) .$$

3 The Security of Two-Round Multi-Signatures using Rewinding

In this section, we analyze the security of existing two-round multi-signature schemes that use rewinding in their security proof. We first look at CoSi and present a metareduction, proving that if the OMDL assumption is hard, there cannot exist an algebraic black-box reduction that proves CoSi secure under the OMDL assumption, making it unlikely that CoSi can be proven secure. Then, we show that the same metareduction with small modifications can be applied to MuSig, the MWLD scheme, and the BCJ scheme, showing that all those schemes cannot be proven secure with an algebraic black-box reduction to OMDL if OMDL is hard, and indicating that the presented security proofs for those schemes contain flaws.

3.1 Impossiblity of Proving CoSi Secure

We first provide an intuition behind the impossibility of proving CoSi secure by sketching why common proof techniques for Schnorr signatures fail in the case of CoSi. We then formalize this and use a metareduction to *prove* that there cannot be a security proof for CoSi in the ROM under the OMDL assumption.

In the classical security proof of Schnorr signatures under the DL assumption [PS00], the reduction feeds its discrete-logarithm challenge y as public key $pk = y$ to the adversary. It uses the zero-knowledge property of the Schnorr protocol to simulate signatures without knowing the secret key. More precisely, the reduction first picks (c, s) at random, then chooses t such that the verification equation $g^s = t \cdot pk^c$ holds, and programs the random oracle $\mathsf{H}(t, m) = c$. The reduction then applies the forking lemma to extract two forgeries from the adversary, from which the discrete logarithm of $pk = y$ can be computed.

The crucial difference between standard Schnorr signatures and CoSi is that in CoSi, the final $\bar{t}$-value included in the hash is the product of individual t_i-values, rather than being determined by a single signer. Therefore, whenever the honest signer is not the leader in the signing query, the adversary learns the final $\bar{t}$ value *before* the simulator does, and can prevent the simulator from programming the random-oracle entry $\mathsf{H}(\bar{t}, m)$. One way around this is to prove security under the OMDL assumption [BP02,MPSW18a], so that the simulator can use its discrete-logarithm oracle to simulate signing queries. Namely, the simulator would use its first target point y_0 as public key $pk = y_0$ and use target points $y_1, \ldots, y_n$ as values $t_1, \ldots, t_n$ when simulating signing queries. Using the forking lemma, it can extract the discrete logarithm of $pk = y_0$, and, based on this value and the responses to its previous discrete-logarithm queries, compute the discrete logarithms of the other target points $t_1, \ldots, t_n$. Overall, the reduction computes the discrete logarithms of $n + 1$ target points using only n queries to the DL oracle.

Unfortunately, this intuitive argument conveys a subtle flaw. Namely, the forking lemma may rewind the adversary to a point where it has an "open" signing query, meaning, a signing query where the simulator already output its t_i value but did not yet receive the final $\bar{t}$ value. The problem is that the adversary may choose a different $\bar{t}$ value in its second execution than it did in its first execution, thereby forcing the simulator to make a second DL query for the same signing query and ruining the simulator's chances to solve the OMDL problem. Indeed, Maxwell et al. [MPSW18a] overlooked this subtle issue that invalidates their security proof. Note that the same problem does not occur in the proof of Schnorr as an identification scheme [BP02] because the adversary does not have access to an identification oracle during the challenge phase.

So in order to correctly simulate signing queries in a rewinding argument, the reduction must be able to provide correct responses s_i and s'_i for the same value t_i but for different challenge values $c = \mathsf{H}(\bar{t}, m)$ and $c' = \mathsf{H}(\bar{t}', m)$. This means, however, that the reduction must already have known the secret key corresponding to pk, as it could have computed it itself as $sk = (s_i - s'_i)/(c - c') \bmod q$. Stronger even, the adversary can give the reduction a taste of its own medicine by forcing the reduction to provide two such responses s_i and s'_i, and extract the value of sk from the reduction!

This sudden turning of the tables, surprising as it may be at first, already hints that the reduction was doomed to fail. Indeed, our proof below exploits this exact technique to build a successful forger: in its first execution, the forger uses the DL oracle to compute a forgery, but in any subsequent rewinding, it will extract the secret key from the reduction and simply create a forgery using the secret key. The meta-reduction thereby ensures that it uses at most one DL oracle query for each of the k "truly different" executions of the forger. By additionally embedding a OMDL target point in its forgery, the meta-reduction reaches a break-even of k DL oracle queries to invert k target points. If the reduction succeeds in solving the n-OMDL problem given access to this forger, then the meta-reduction can use its solution to solve the $(n+k)$-OMDL problem.

While this captures the basic idea of our proof, some extensions are needed to make it work for *any* reduction. For example, one could imagine a reduction using a modified forking technique that makes sure that the same challenge value $c = \mathsf{H}(\bar{t}, m)$ is always used across timelines, e.g., by guessing the index of that random-oracle query. To corner such a reduction, our forger makes several signing queries in parallel and chooses one of two challenges at random for each query. When the reduction rewinds the forger, the reduction will with overwhelming probability be forced to respond to a different challenge on at least one of the signing queries, allowing the forger to extract.

Below, we formally prove that if the OMDL assumption holds, then there cannot exist a reduction (with some constraints, as discussed later) that proves the security of CoSi under the OMDL assumption. Our proof roughly follows the techniques of Baldimtsi and Lysyanskaya [BL13] for Schnorr-based blind signature schemes, in the sense that we also present a forger and a meta-reduction that, given a reduction that solves the OMDL problem when given black-box access to a forger, solves the OMDL problem by extracting a discrete logarithm from the reduction. Our proof is different, however, in the sense that we cover a different class of reductions (algebraic black-box reductions, as opposed to "naive random-oracle replay reductions"), and because the multi-signature scheme requires a more complicated forger because challenges used by the signing oracle must be random-oracle outputs, as opposed to arbitrary values in the case of [BL13]. The class of reductions that we exclude is large enough to encompass all currently known proof techniques for this type of schemes, making it extremely unlikely that CoSi will ever be proven secure under the DL or OMDL assumption.

Theorem 1. *If the $(n+k)$-OMDL problem is $(\tau + \tau_{\text{ext}} + O(n + k\ell), \epsilon - k^2/2^\ell)$-hard, then there exists no algebraic black-box reduction $\mathcal{B}$ that proves* CoSi *to be $((2\ell+1)\tau_{\text{exp}} + O(\ell), \ell, \ell, 3, 1 - 1/q)$-unforgeable in the KOSK setting in the random-oracle model under the assumption that the n-OMDL problem is (τ, ϵ)-hard. Here, τ_{ext} is the running time of* Extract *as per Definition 3, τ_{exp} is the time to perform an exponentiation in $\mathbb{G}$, and k is the amount of times that $\mathcal{B}$ runs $\mathcal{A}$ through rewinding, and ℓ is a security parameter.*

Before proving the theorem, we provide some guidance on how to interpret its result. In a nutshell, the theorem says that if the OMDL problem is hard, then there is hardly any hope to prove CoSi secure under

the DL or OMDL assumption, even in the KOSK setting and in the random-oracle model. It thereby also excludes, *a fortiori*, any security proofs in the key-verification and plain public-key settings or in the standard model.

For concreteness, let us set $k = \ell = 128$, and let us say that we have a forger that breaks CoSi with overwhelming probability using just 257 exponentiations, 128 signature queries, and 3 random-oracle queries. That would indeed be a pretty serious security breach, certainly serious enough to rule out any further use of CoSi in practice. Yet still, even for such a strong forger, Theorem 1 says that, if the 128-OMDL problem is hard, then there cannot exist a reduction $\mathcal{B}$ that uses this forger to solve the DL problem. More generally, it says that if the $(n + 128)$-OMDL problem is hard, then neither can there exist a reduction that solves the n-OMDL problem. If such a reduction would exist, then that reduction would immediately give rise to a solution for the $(n + 128)$-OMDL problem *without* needing access to any forger, meaning that the OMDL assumption was false to begin with.

The only room left by Theorem 1 are for a number of alternative proof approaches, but none of them look particularly hopeful. First, the theorem becomes moot when the OMDL problem turns out to be easy but the DL problem remains hard, or when the $(n + k)$-OMDL problem is easy but the n-OMDL problem is still hard. At present, however, there is no evidence suggesting that any of these problems may be easier than the others. Second, it does not rule out the existence of non-algebraic or non-black-box reductions. The former type of reduction would imply strange properties of the underlying group. The latter would have to obtain a special advantage from inspecting the code of the forger, rather than just being able to execute it. While some cryptographic uses of non-black-box techniques exist [Bar04], to the best of our knowledge they have never been used in practical constructions such as CoSi. Finally, our theorem does not rule out security proofs under assumptions that are not implied by n-OMDL or proving security in the generic group model [Sho97]. However, this would mean that much stronger assumptions are required than one would expect from a Schnorr-based protocol.

Proof (Proof of Theorem 1). We prove the theorem by constructing a forger $\mathcal{F}$ and a meta-reduction $\mathcal{M}$ such that, if there exists a reduction $\mathcal{B}$ that uses $\mathcal{F}$ to break the n-OMDL problem, then $\mathcal{M}$ can use $\mathcal{B}$ to break the $(n + k)$-OMDL problem. Figure 2 depicts the execution setting of all three algorithms.

Let $y_0, \ldots, y_{n+k}$ denote the $n + k + 1$ OMDL challenge points that $\mathcal{M}$ receives as input. It will provide $\mathcal{B}$ with an environment that simulates the n-OMDL game by handing $y_0, \ldots, y_n$ as input to $\mathcal{B}$ and responding to $\mathcal{B}$'s $\mathcal{O}^{\mathtt{dlog}}$ queries using its own $\mathcal{O}^{\mathtt{dlog}}$ oracle. We have to provide reduction $\mathcal{B}$ with a successful forger $\mathcal{F}$ against CoSi, where $\mathcal{B}$ is free to run and rewind $\mathcal{F}$. To simplify the arguments about rewinding, we will describe a deterministic forger $\mathcal{F}$, so that the behavior of $\mathcal{F}$ only depends on the inputs and oracle responses provided by $\mathcal{B}$, not on its random coins.

We describe a forger $\mathcal{F}$ in terms of three subroutines target, rand, and forge that $\mathcal{F}$ can call out to but that will be implemented by the meta-reduction $\mathcal{M}$. Subroutine target takes $\ell+1$ group elements $(pk, t_1, \ldots, t_\ell)$ as input and on the i-th invocation with a combination of inputs that it hasn't been called with before, returns $\mathcal{M}$'s target point y_{n+i}. Any invocations of target on previously used inputs consistently return the same output. The subroutine rand implements a truly random function $\mathbb{G}^{\ell+1} \times \mathbb{Z}_q^3 \to \{0,1\}^\ell$, which is simulated by $\mathcal{M}$ through lazy sampling. The subroutine forge, finally, creates a forgery by returning an s-value, given a $\bar{t}$ value, a public key, and a c-value; we will specify later how $\mathcal{M}$ implements this routine.

Let pk_i be the public key that $\mathcal{B}$ provides to $\mathcal{F}$ in its i-th execution of $\mathcal{F}$. The forger $\mathcal{F}$ then proceeds as follows:

- On input pk_i, $\mathcal{F}$ initiates ℓ signing queries on the same message m and for the same tree $\mathcal{T}$ consisting of two signers: a leader with public key $pk = g$ and a child that is the target signer with public key pk_i.
- After receiving the results of the first round $t_{i,1}, \ldots, t_{i,\ell}$, $\mathcal{F}$ sets $\bar{t}_i^* \leftarrow \mathsf{target}(pk_i, t_{i,1}, \ldots, t_{i,\ell})$.
- $\mathcal{F}$ makes a random-oracle query $\mathsf{H}(\bar{t}_i^*, m^*)$ for a fixed message $m^* \neq m$, yielding a response c_i^*.
- $\mathcal{F}$ makes two additional random-oracle queries on $\mathsf{H}(1_\mathbb{G}, m)$ and $\mathsf{H}(g, m)$, yielding responses $c_{i,0}$ and $c_{i,1}$, respectively.
- If $c_{i,0} = c_{i,1}$, then $\mathcal{F}$ aborts. Otherwise, it continues the ℓ open signing sessions by generating random bits $b_{i,1} \| \ldots \| b_{i,\ell} \leftarrow \mathsf{rand}((pk_i, t_{i,1}, \ldots, t_{i,\ell}), (c_i^*, c_{i,0}, c_{i,1}))$ and sending the final $\bar{t}$-value for the j-th signing session as $\bar{t}_{i,j} \leftarrow g^{b_{i,j}}$ for $j = 1, \ldots, \ell$.

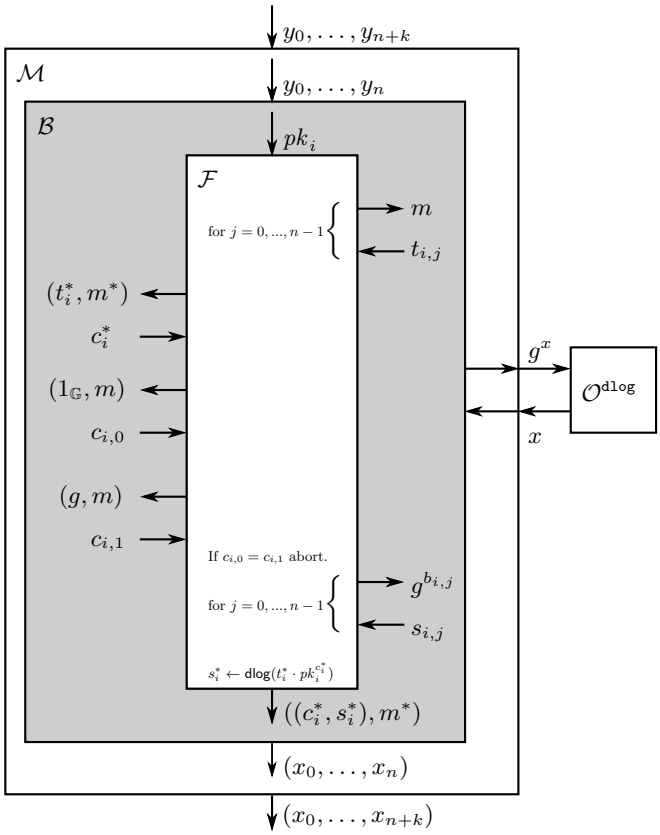

Fig. 2. Our metareduction $\mathcal{M}$ in the proof of Theorem 1, which simulates forger $\mathcal{F}$ towards any reduction $\mathcal{B}$ that would prove the security of CoSi under the OMDL assumption, and uses $\mathcal{B}$ to break the OMDL problem.

- When $\mathcal{F}$ receives the values $s_{i,1}, \ldots, s_{i,\ell}$ in the ℓ signing protocols, it verifies that $g^{s_{i,j}} = t_{i,j} \cdot pk_i^{c_{i,b_{i,j}}}$, aborting if an invalid signature is detected.
- $\mathcal{F}$ outputs a forgery (c_i^*, s_i^*) on m^* with public keys $\mathcal{PK} = \{pk_i\}$ by computing $s_i^* \leftarrow \mathsf{forge}(\bar{t}_i^*, pk_i, c_i^*)$.

Observe that $\mathcal{F}$ makes ℓ signing queries, three random-oracle queries, and performs at most $(2\ell + 1)$ exponentiations so that $\mathcal{F}$ runs in time $(2\ell + 1)\tau_{\exp} + O(\ell)$. It outputs a successful forgery unless $c_{i,0} = c_{i,1}$, which happens with probability $1/q$. Therefore, $\mathcal{F}$ is a $((2\ell + 1)\tau_{\exp} + O(\ell), \ell, 3, 1 - 1/q)$-forger for CoSi. Note that $\mathcal{F}$ works in the KOSK setting because the forgery doesn't include any signer other than the target signer.

Suppose that there exists an algebraic reduction $\mathcal{B}$ that, when given black-box access to the above forger $\mathcal{F}$, (τ, ϵ)-breaks the n-OMDL problem. We now describe a meta-reduction $\mathcal{M}$ that breaks the $(n+k)$-OMDL problem, where k is the number of times that $\mathcal{B}$ runs $\mathcal{F}$. As mentioned earlier, $\mathcal{M}$, on input target points $y_0, \ldots, y_{n+k}$, runs $\mathcal{B}$ on input $y_0, \ldots, y_n$ and forwards $\mathcal{B}$'s $\mathcal{O}^{\mathtt{dlog}}$ queries to its own $\mathcal{O}^{\mathtt{dlog}}$ oracle. It implements the subroutines target and rand as explained above, and implements the forge subroutine as follows:

- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}_i^*, pk_i, c_i^*)$ and there exists a previous execution $i' \neq i$ that already computed the secret key sk_i corresponding to pk_i, then the subroutine computes and return the requested s-value as $s_i^* \leftarrow s_{i'}^* + (c_i^* - c_{i'}^*) \cdot sk_i \bmod q$.
- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}_i^*, pk_i, c_i^*)$ and there exists a previous execution $i' \neq i$ with $(pk_{i'}, t_{i',1}, \ldots, t_{i',\ell}) = (pk_i, t_{i,1}, \ldots, t_{i,\ell})$, then it checks whether $(c_{i',b_{i',1}}, \ldots, c_{i',b_{i',\ell}}) = (c_{i,b_{i,1}}, \ldots, c_{i,b_{i,\ell}})$. If so, then $\mathcal{M}$ halts and outputs failure. If not, then there exists at least one index j such that $c_{i',b_{i',j}} \neq c_{i,b_{i,j}}$, so that $\mathcal{M}$ can compute the secret key sk_i corresponding to pk_i as

$sk_i \leftarrow \frac{s_{i,j} - s_{i',j}}{c_{i,b_{i,j}} - c_{i',b_{i',j}}} \bmod q$. It can then compute and return the requested s-value as $s_i^* \leftarrow s_{i'}^* + (c_i^* - c_{i'}^*) \cdot sk_i \bmod q$.

– Else, $\mathcal{M}$ uses $\mathcal{O}^{\mathtt{dlog}}$ and returns $s_i^* \leftarrow \mathcal{O}^{\mathtt{dlog}}(\bar{t}_i^* \cdot pk_i^{c_i^*})$.

If $\mathcal{B}$ is successful, then $\mathcal{B}$ will output $x_0, \ldots, x_n$ such that $y_i = g^{x_i}$ for $i = 0, \ldots, n$ after having made at most n queries to its $\mathcal{O}^{\mathtt{dlog}}$ oracle. Now $\mathcal{M}$ proceeds to compute the discrete logarithms $x_{n+1}, \ldots, x_{n+k}$ of $y_{n+1}, \ldots, y_{n+k}$ as follows.

Let P be the partition of $\{1, \ldots, k\}$ where i and i' are considered equivalent (and are therefore in the same component $C \in P$) if the i-th and i'-th executions are such that $(pk_i, t_{i,1}, \ldots, t_{i,\ell}) = (pk_{i'}, t_{i',1}, \ldots, t_{i',\ell})$. Because of the way $\mathcal{M}$ instantiated the target subroutine, we know that $\mathcal{M}$ used the same target point y_{j_C} as the value $\bar{t}_i^*$ for all executions i that are in the same component $C \in P$, meaning that during the full simulation of $\mathcal{B}$, $\mathcal{M}$ used target points $y_{n+1}, \ldots, y_{n+|P|}$. Let P_0 be the set of components $C \in P$ such that $\mathcal{F}$ never invoked the forge subroutine in any execution $i \in C$, let P_1 contain $C \in P$ such that $\mathcal{F}$ invoked the forge exactly once over all executions $i \in C$, and let P_{2+} contain the components $C \in P$ such that $\mathcal{F}$ invoked forge at least twice in total over all executions $i \in C$. It is clear that $|P| = |P_0| + |P_1| + |P_{2+}|$.

We will now show that $\mathcal{M}$, using a total of $|P|$ queries to its $\mathcal{O}^{\mathtt{dlog}}$ oracle, can derive a system of $|P|$ independent linear equations in the $|P|$ unknowns $x_{n+1}, \ldots, x_{n+|P|}$. Namely, for every component $C \in P_0$, $\mathcal{M}$ simply makes a discrete-logarithm query $\alpha_C \leftarrow \mathcal{O}^{\mathtt{dlog}}(y_{j_C})$, which adds an equation of the form

$$x_{j_C} \;=\; \alpha_C \;. \tag{1}$$

For every component $C \in P_1$, there exists exactly one execution $i \in C$ that caused $\mathcal{M}$ to make a query $s_i^* \leftarrow \mathcal{O}^{\mathtt{dlog}}(y_{j_C} \cdot pk_i^{c_i^*})$. Since $\mathcal{B}$ is algebraic and only obtains group elements $g, y_0, \ldots, y_{n+k}$ as input, for all pk_i output by $\mathcal{B}$, $\mathcal{M}$ can use $\mathsf{Extract}$ to obtain coefficients $\beta_i, \beta_{i,0}, \ldots, \beta_{i,n+k} \in \mathbb{Z}_q$ such that $sk_i = \log_g(pk_i) = \beta_i + \sum_{j=0}^{n+k} \beta_{i,j} x_j \bmod q$. For every $C \in P_1$ it therefore has an equation of the form

$$s_i^* \;=\; x_{j_C} + c_i^* (\beta_i + \sum_{j=0}^{n+k} \beta_{i,j} x_j) \bmod q \;. \tag{2}$$

Note that $x_0, \ldots, x_n$ are known values above, as they were output by $\mathcal{B}$. For every component $C \in P_{2+}$, $\mathcal{M}$ made one discrete-logarithm query $s_i^* \leftarrow \mathcal{O}^{\mathtt{dlog}}(y_{j_C} \cdot pk_i^{c_i^*})$ during the first invocation of forge, and extracted the value of sk_i during the second invocation of forge. It can therefore add an equation of the form

$$s_i^* \;=\; x_{j_C} + c_i^* sk_i \bmod q \;. \tag{3}$$

Finally, for the unused target points y_j, $j \in \{n + |P| + 1, \ldots, n + k\}$, $\mathcal{M}$ can make an additional query $\alpha_j \leftarrow \mathcal{O}^{\mathtt{dlog}}(y_j)$ to obtain an equation

$$x_j \;=\; \alpha_j \;. \tag{4}$$

The metareduction $\mathcal{M}$ created a system of $|P_0|$ equations of the form (1), $|P_1|$ equations of the form (2), $|P_{2+}|$ equations of the form (3), and $k - |P|$ equations of the form (4), so that overall it has a system of k linear equations in k unknowns. The equations of the form (1), (3), and (4) are clearly linearly independent, as each of these equations affects a single and different unknown x_j. Equations of the form (2) are independent as well, because at the time that $\mathcal{B}$ produces pk_i, its view is independent of $y_{j_{i'}}$ for $i' > i$. One can therefore order the equations of the form (2) such that each contains one unknown x_{j_C} that does not occur in any of the preceding equations.

Solving this linearly independent system of k equations in k unknowns yields all the values for $x_{n+1}, \ldots, x_k$. $\mathcal{M}$ can therefore output $(x_0, \ldots, x_{n+k})$ after having made exactly one $\mathcal{O}^{\mathtt{dlog}}$ query for each of the k equations and at most n $\mathcal{O}^{\mathtt{dlog}}$ queries to respond to $\mathcal{B}$'s $\mathcal{O}^{\mathtt{dlog}}$ queries, meaning at most $n + k$ queries in total.

The metareduction $\mathcal{M}$ runs in time $\tau + \tau_{\text{ext}} + O(n + k\ell)$ and wins the $(n + k)$-OMDL game whenever $\mathcal{B}$ wins the n-OMDL game, unless $\mathcal{M}$ outputs failure. The latter happens when in the i-th execution of $\mathcal{F}$, there exists a previous execution $i' < i$ with $(pk_{i'}, t_{i',1}, \ldots, t_{i',\ell}) = (pk_i, t_{i,1}, \ldots, t_{i,\ell})$ and $(c_{i',b_{i',1}}, \ldots, c_{i',b_{i',\ell}}) =$

$(c_{i,b_{i,1}}, \ldots, c_{i,b_{i,\ell}})$. We know that $c_{i,0} \neq c_{i,1}$, because otherwise $\mathcal{F}$ would have aborted earlier, meaning that at most one choice for $b_{i,j}$ will cause $c_{i',b_{i',j}} = c_{i,b_{i,j}}$. Therefore, at the moment that $b_{i,1} \| \ldots \| b_{i,\ell}$ is chosen at random from $\{0,1\}^\ell$ in a call to the rand subroutine, for each execution $i' \neq i$ there is at most one bad choice for $b_{i,1} \| \ldots \| b_{i,\ell}$ that causes $\mathcal{M}$ to output failure, meaning that there are at most k bad choices overall. (Note that the output of rand is fresh because it takes the full transcript of the protocol so far as an argument. If the arguments of rand are equal in the i-th and i'-th execution, then the executions are simply identical. Also note that $\mathcal{B}$ learns $\mathcal{F}$'s choice for $b_{i,1} \| \ldots \| b_{i,\ell}$ before $\mathcal{F}$ calls the forge subroutine, so that it could keep many candidate executions i' open at the same time.) The probability that the choice of $b_{i,1} \| \ldots \| b_{i,\ell}$ hits any of these k bad choices causing $\mathcal{M}$ to output failure in any of the k executions is at most $k^2/2^\ell$. The success probability in solving the $(n+k)$-OMDL game is therefore $\epsilon - k^2/2^\ell$.

3.2 Applicability to MuSig

While our metareduction is written for CoSi, the same technique can be applied to the similar multi-signature scheme MuSig as recently introduced by Maxwell et al. [MPSW18a]. The main difference between CoSi and MuSig is in how they avoid rogue-key attacks. While CoSi uses the key-verification model to avoid these attacks, MuSig works in the plain public key model by using a more involved key aggregation procedure. Rather than simply multiplying the individual keys together, they raise the individual keys to a hash function output, and present a security proof under the OMDL assumption. However, the problem in proving CoSi secure is not related to rogue-key attacks, as demonstracted by the fact that our metareduction holds in the KOSK setting, but due to the fact that many signing queries can be made in parallel, and rewinding may force the reduction to know the signer's secret key. Indeed, the same metareduction (with some minor changes in bookkeeping and including the more involved key aggregation) is applicable to MuSig, proving that their security proof overlooked this case and that it is very unlikely that MuSig can be proven secure under standard assumptions. In response to this work, the authors updated MuSig to use a 3-round signing algorithm [MPSW18b].

3.3 Applicability to MWLD

Our metareduction can be applied to the MWLD scheme with small modifications. This means that the security proof under the DL assumption [MWLD10] is flawed.[8] While the metareduction is mostly unchanged, the forger and the forge-routine slightly change to account for the double generator and the double hashing. The modified forger $\mathcal{F}$ works as follows:

- On input pk_i, $\mathcal{F}$ initiates ℓ signing queries on the same message m and for the same tree $\mathcal{T}$ consisting of two signers: a leader with public key $pk = 1_{\mathbb{G}}$ and a child that is the target signer with public key pk_i.
- After receiving the results of the first round $t_{i,1}, \ldots, t_{i,\ell}$, $\mathcal{F}$ sets $\bar{t}_i^* \leftarrow \mathsf{target}(pk_i, t_{i,1}, \ldots, t_{i,\ell})$.
- $\mathcal{F}$ makes random-oracle query $\mathsf{H}_0(\bar{t}_i^*, \{pk_i\}, m^*)$ for a fixed message $m^* \neq m$, yielding a response c_i^* and random-oracle query $\mathsf{H}_1(c_i^*, pk_i)$ yielding v_i^*.
- $\mathcal{F}$ makes four additional queries on $\mathsf{H}_1(\mathsf{H}_0(1_{\mathbb{G}}, \{pk, pk_i\}, m), pk_i)$ and $\mathsf{H}_1(\mathsf{H}_0(g, \{pk, pk_i\}, m), pk_i)$, yielding responses $v_{i,0}$ and $v_{i,1}$, respectively.
- If $v_{i,0} = v_{i,1}$, then $\mathcal{F}$ aborts. Otherwise, it continues the ℓ open signing sessions by generating random bits $b_{i,1} \| \ldots \| b_{i,\ell} \leftarrow \mathsf{rand}((pk_i, t_{i,1}, \ldots, t_{i,\ell}), (v_i^*, v_{i,0}, v_{i,1}))$ and sending the final $\bar{t}$-value for the j-th signing session as $\bar{t}_{i,j} \leftarrow g^{b_{i,j}}$ for $j = 1, \ldots, \ell$.
- When $\mathcal{F}$ receives the values $(s_{i,1,1}, s_{i,1,2}), \ldots, (s_{i,\ell,1}, s_{i,\ell,2})$ in the ℓ signing protocols, it verifies that $g^{s_{i,j,1}} h^{s_{i,j,2}} = t_{i,j} \cdot pk_i^{v_{i,b_{i,j}}}$, aborting if an invalid signature is detected.
- $\mathcal{F}$ outputs a forgery $(c_i^*, s_{i,1}^*, s_{i,2}^*)$ on message m^* with public keys $\mathcal{PK} = \{pk_i\}$ by taking $(s_{i,1}^*, s_{i,1}^*) \leftarrow \mathsf{forge}(\bar{t}_i^*, pk_i, v_i^*)$.

[8] Claim 4 of [MWLD10] is incorrect: while the view of the forger is independent of (sk_1, sk_2) in a single run of the forger, the joint input of different rewindings of the forger are not independent of (sk_1, sk_2), and hence the keys that the reduction extracts from these different rewindings are not independent of (sk_1, sk_2) either. Their event E_1 may therefore occur with a non-negligible probability.

The metareduction $\mathcal{M}$ implements forge as follows:

- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}_i^*, pk_i, v_i^*)$ and there exists a previous execution $i' \neq i$ that already computed the secret key $(sk_{i,1}, sk_{i,2})$ corresponding to pk_i, then the subroutine computes and return the requested s-values as $s_{i,b}^* \leftarrow s_{i',b}^* + (v_i^* - v_{i'}^*) \cdot sk_{i,b} \bmod q$. for $b \in \{1,2\}$.
- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}_i^*, pk_i, v_i^*)$ and there exists a previous execution $i' \neq i$ with $(pk_{i'}, t_{i',1}, \ldots, t_{i',\ell}) = (pk_i, t_{i,1}, \ldots, t_{i,\ell})$, then it checks whether $(v_{i',b_{i',1}}, \ldots, v_{i',b_{i',\ell}}) = (v_{i,b_{i,1}}, \ldots, v_{i,b_{i,\ell}})$. If so, then $\mathcal{M}$ halts and outputs failure. If not, then there exists at least one index j such that $v_{i',b_{i',j}} \neq v_{i,b_{i,j}}$, so that $\mathcal{M}$ can compute a secret key $(sk_{i,1}, sk_{i,2})$ corresponding to pk_i as $sk_{i,b} \leftarrow \frac{s_{i,j,b} - s_{i',j,b}}{v_{i,b_{i,j}} - v_{i',b_{i',j}}} \bmod q$ for $b \in \{0,1\}$. It can then compute and return the requested s-value as $s_{i,b}^* \leftarrow s_{i',b}^* + (v_i^* - v_{i'}^*) \cdot sk_{i,b} \bmod q$ for $b \in \{0,1\}$.
- Else, $\mathcal{M}$ picks $s_{i,2}^* \xleftarrow{\$} \mathbb{Z}_q$ uses $\mathcal{O}^{\mathtt{dlog}}$ to compute $s_{i,1}^* \leftarrow \mathcal{O}^{\mathtt{dlog}}(\bar{t}_i^* \cdot pk_i^{v_i^*} \cdot h^{-s_{i,2}^*})$ and outputs $(s_{i,1}^*, s_{i,2}^*)$.

3.4 Applicability to BCJ

Our metareduction can be applied to the BCJ key-verification model scheme with small modifications. This means that the security proof under the DL assumption [BCJ08] is flawed.[9]

The modified forger $\mathcal{F}$ works as follows:

- On input $pk_i = (y_i, \pi_i)$, $\mathcal{F}$ initiates ℓ signing queries on the same message m and for the same tree $\mathcal{T}$ consisting of two signers: a leader with public key $pk = (1_{\mathbb{G}}, \pi)$ (where proof-of-possession π can be honestly constructed for $sk = 0$) and a child that is the target signer with public key pk_i.
- After receiving the results of the first round $(t_{i,1,1}, t_{i,1,2}), \ldots, (t_{i,\ell,1}, t_{i,\ell,2})$, $\mathcal{F}$ takes $(\alpha_{i,1}, \alpha_{i,2}) \xleftarrow{\$} \mathbb{Z}_q^2$ and sets $\bar{t}_{i,1}^* \leftarrow g_1^{\alpha_{i,1}} h_2^{\alpha_{i,2}}$. It sets $\bar{t}_{i,2}^* \leftarrow \mathsf{target}(y_i, (t_{i,1,1}, t_{i,1,2}), \ldots, (t_{i,\ell,1}, t_{i,\ell,2}))$.
- $\mathcal{F}$ makes random-oracle query $\mathsf{H}_0(y_i, (\bar{t}_{i,1}^*, \bar{t}_{i,2}^*), m^*)$ for a fixed message $m^* \neq m$, yielding a response c_i^*.
- $\mathcal{F}$ makes two additional random-oracle queries on $\mathsf{H}_0(y_1, 1_{\mathbb{G}}, 1_{\mathbb{G}}, m)$ and $\mathsf{H}_0(y_1, g_1, g_1, m)$, yielding responses $c_{i,0}$ and $c_{i,1}$, respectively.
- If $c_{i,0} = c_{i,1}$, then $\mathcal{F}$ aborts. Otherwise, it continues the ℓ open signing sessions by generating random bits $b_{i,1} \| \ldots \| b_{i,\ell} \leftarrow \mathsf{rand}((y_i, t_{i,1}, \ldots, t_{i,\ell}), (c_i^*, c_{i,0}, c_{i,1}))$ and sending the $(\bar{t}_1, \bar{t}_2, PK)$-values for the j-th signing session as $\bar{t}_{i,j,1} \leftarrow g_1^{b_{i,j}}$ and $\bar{t}_{i,j,2} \leftarrow g_1^{b_{i,j}}$ for $j = 1, \ldots, \ell$.
- When $\mathcal{F}$ receives the values $(s_{i,1}, \gamma_{i,1,1}, \gamma_{i,1,2}), \ldots, (s_{i,\ell}, \gamma_{i,\ell,1}, \gamma_{i,\ell,2})$ in the ℓ signing protocols, it verifies that $t_{i,j,1} = g_1^{\gamma_{i,j,1}} h_1^{\gamma_{i,j,2}}$ and that $t_{i,j,2} = g_2^{\gamma_{i,j,1}} h_2^{\gamma_{i,j,2}} g_1^{s_{i,j}} y_1^{-c_{i,b_{i,j}}}$, aborting if an invalid signature is detected.
- $\mathcal{F}$ outputs a forgery $(\bar{t}_{i,1}^*, \bar{t}_{i,2}^*, s_i^*, \gamma_{i,1}^*, \gamma_{i,2}^*)$ on message m^* with public keys $\mathcal{PK} = \{pk_i\}$ by taking $(\gamma_{i,1}^*, \gamma_{i,2}^*, s_i^*) \leftarrow \mathsf{forge}(\bar{t}_{i,1}^*, \bar{t}_{i,2}^*, \alpha_{i,1}, \alpha_{i,2}, y_i, c_i^*)$.

The metareduction $\mathcal{M}$ implements forge as follows:

- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}_{i,1}^*, \bar{t}_{i,2}^*, \alpha_{i,1}, \alpha_{i,2}, y_i, c_i^*)$ and there exists a previous execution $i' \neq i$ that already computed representation $(\delta_{\gamma_1}, \delta_{\gamma_2}, \delta_s)$ for y_i, then the subroutine computes the requested forgery as $\gamma^* i, 1 \leftarrow \gamma_{i',1}^* + (c_i^* - c_{i'}^*)\delta_{\gamma_1}$, $\gamma^* i, 2 \leftarrow \gamma_{i',2}^* + (c_i^* - c_{i'}^*)\delta_{\gamma_2}$, $s_i^* \leftarrow s_{i'}^* + (c_i^* - c_{i'}^*)\delta_s$.

[9] The security proof distinguishes two forgery events: In E_1, the forger broke the binding property of the commitment scheme, and in E_2, the reduction can extract the secret key of the honest signer. It considers two different reductions, $\mathcal{B}_0$ which embeds the DL challenge in the commitment parameters and simulates signing queries by knowing the honest signer secret key, and $\mathcal{B}_1$ which embeds the DL challenge as the honest signer public key and simulates signing queries by knowing the backdoor to the commitment scheme. The DL challenge is solved if E_1 occurs with $\mathcal{B}_1$ or if E_2 occurs with $\mathcal{B}_0$. The proof argues that, because the simulation strategies of $\mathcal{B}_0$ and $\mathcal{B}_1$ are indistinguishable, for each event it holds that the difference between the probabilities that it occurs in reduction $\mathcal{B}_0$ and $\mathcal{B}_1$ is negligible. However, while it is true that the forger cannot distinguish between the oracle simulation strategies of $\mathcal{B}_0$ and $\mathcal{B}_1$ in a *single* execution, the events E_1 and E_2 are defined based on the joint outputs of *multiple* rewindings of the forger, which do depend on the simulation strategy. The events may therefore occur with non-negligibly different probability in both reductions, invalidating the proof.

– If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{forge}(\bar{t}^*_{i,1}, \bar{t}^*_{i,2}, \alpha_{i,1}, \alpha_{i,2}, y_i, c^*_i)$ and there exists a previous execution $i' \neq i$ with $(y_i, (t_{i,1,1}, t_{i,1,2}), \ldots, (t_{i,\ell,1}, t_{i,\ell,2})) = (y_i, (t_{i,1,1}, t_{i,1,2}), \ldots, (t_{i,\ell,1}, t_{i,\ell,2}))$, then it checks whether $(c_{i',b_{i',1}}, \ldots, c_{i',b_{i',\ell}}) = (c_{i,b_{i,1}}, \ldots, c_{i,b_{i,\ell}})$. If so, then $\mathcal{M}$ halts and outputs failure. If not, then there exists at least one index j such that $c_{i',b_{i',j}} \neq c_{i,b_{i,j}}$, so that $\mathcal{M}$ extracts a representation of y_i by setting

$$\delta_{\gamma_1} \leftarrow \frac{\gamma_{i,j,1} - \gamma_{i',j,2}}{c_{i,b_{i,j}} - c_{i',b_{i',j}}} \bmod q$$

$$\delta_{\gamma_2} \leftarrow \frac{\gamma_{i,j,2} - \gamma_{i',j,2}}{c_{i,b_{i,j}} - c_{i',b_{i',j}}} \bmod q$$

$$\delta_s \leftarrow \frac{s_{i,j} - s_{i',j}}{c_{i,b_{i,j}} - c_{i',b_{i',j}}} \bmod q$$

for which we have

$$y_i = g_2^{\delta_{\gamma_1}} h_2^{\delta_{\gamma_2}} g_1^{\delta_s} \tag{5}$$

and

$$g_1^{\delta_{\gamma_1}} h_1^{\delta_{\gamma_2}} = 1_{\mathbb{G}}. \tag{6}$$

It can then compute and return the requested forgery as $\gamma^*_{i,1} \leftarrow \gamma^*_{i',1} + (c^*_i - c^*_{i'})\delta_{\gamma_1}$, $\gamma^*_{i,2} \leftarrow \gamma^*_{i',2} + (c^*_i - c^*_{i'})\delta_{\gamma_2}$, $s^*_i \leftarrow s^*_{i'} + (c^*_i - c^*_{i'})\delta_s$.

– Else, $\mathcal{M}$ sets $\gamma_{i,1} \leftarrow \alpha_{i,1}$, $\gamma_{i,2} \leftarrow \alpha_{i,2}$, and uses $\mathcal{O}^{\mathtt{dlog}}$ to compute $s^*_i \leftarrow \mathcal{O}^{\mathtt{dlog}}(\bar{t}^*_{i,2} g_2^{-\alpha i,1} h_2^{\alpha i,2} y_i^{c^*_i})$ and returns $(\gamma_{i,1}, \gamma_{i,2}, s_i)$.

4 Sub-Exponential Attacks on Multi-Signature Schemes

In this section we state the k-sum problem [Wag02], which is a k-dimensional generalization of the birthday problem. We show that the security of CoSi, MuSig, MWLD, and BCJ multi-signature schemes can be reduced to this problem. Wagner [Wag02] proposed an algorithm that solves the k-sum problem in sub-exponential time. Together these results allow us to break the above multisignature schemes. Finally, we point out that the public-randomness generator RandHound [SJKK+17] is insecure, which stems from its use of the CoSi scheme.

Our attacks are related to the ROS-based attack on blind signatures [Sch01], which can also be reduced to the k-sum problem [Wag02]. However, for subtle reasons our attacks on multi-signatures need to use the k-sum problem directly.

4.1 The k-Sum Problem

We now define the k-sum problem and provide the complexity estimates for solving it.

Definition 5 (k-Sum Problem). *The k-sum problem in group $(\mathbb{Z}_q, +)$ for an arbitrary q provides k lists $L_1, \ldots, L_k$ of equal size, each containing elements sampled uniformly and independently from $\mathbb{Z}_q$, and requires to find $x_1 \in L_1, \ldots, x_k \in L_k$ such that $x_1 + \ldots + x_k \equiv 0 \mod q$. We say that adversary $\mathcal{A}$ $(\tau, s_{\mathrm{L}}, \epsilon)$-breaks the k-sum problem if it runs in time at most τ, uses lists $L_1, \ldots, L_k$ each containing s_{L} elements, and solves the k-sum problem with probability $\mathsf{Adv}^{\mathsf{k\text{-}sum}}_{\mathbb{Z}_q}(\mathcal{A}) \geq \epsilon$.*

For any parameters k and n, Wagner [Wag02] defines the k-tree algorithm that $(\tau, s_{\mathrm{L}}, \epsilon)$-breaks the k-sum problem on n-bit long list elements for $\tau \in O(k \cdot 2^{n/(1+\lg k)})$, $s_{\mathrm{L}} = 2^{n/(1+\lg k)}$ and ϵ some non-negligible function. The k-tree algorithm roughly requires to merge $k-1$ pairs of s_{L}-element lists. This means that the 4-sum problem with lists of size $2^{n/3}$ can be solved with a non-negligible probability in time $O(2^{n/3})$. Furthermore, if $k = 2^{\sqrt{n}}$ then the k-sum problem with lists of size $2^{\sqrt{n}}$ can be solved in sub-exponential time $O(2^{2\sqrt{n}})$. The analysis of the k-tree algorithm by Minder and Sinclair [MS09] suggests that $\epsilon \approx 1/4$. Some further improvements to the k-tree algorithm are provided by Nikolić and Sasaki [NS15].

4.2 Attack on CoSi

We state the reduction from the unforgeability of CoSi to the k-sum problem, and discuss the implications of our result to the choice of CoSi parameters.

Reduction to the k-Sum Problem The attack uses a 2-node tree $\mathcal{T}$ where the challenger controls the leaf node with some secret key sk^*, and the adversary controls the root node with a secret key sk. The adversary chooses an arbitrary message m and initiates $k-1$ concurrent calls to the signing oracle, using $m, \mathcal{T}$ as inputs to each of the oracle queries. At the beginning of the challenge phase of each concurrent oracle call $i \in \{1, \ldots, k-1\}$ the adversary chooses a malicious $\bar{t}_i$ (we show how to choose it below). Note that there are no restrictions on the choice of $\bar{t}_i$, because it fully depends on adversary's commitment in the root node at the end of the commitment phase. During the response phase the adversary obtains from $\mathcal{T}$'s leaf node $s_i = r_i + c_i \cdot sk^*$ for all $i \in \{1, \ldots, k-1\}$, one for each concurrent call to the signing oracle. Here r_i is uniformly random (and unknown to the adversary) and $c_i = \mathsf{H}_0(\bar{t}_i, m)$. The adversary then constructs a signature $\sigma = (c^*, s^*)$ for some message $m^* \neq m$ such that $c^* = \sum_{i=1}^{k-1} c_i$ and $s^* = \sum_{i=1}^{k-1} s_i + c^* \cdot sk$. We now show how to choose the $\bar{t}_i$'s such that σ is a valid signature for m^*.

By expanding the above, we get $s^* = \sum_{i=1}^{k-1} r_i + \sum_{i=1}^{k-1} c_i \cdot (sk^* + sk) = \sum_{i=1}^{k-1} r_i + c^* \cdot (sk^* + sk)$. Let $h = g^{\sum_{i=1}^{k-1} r_i}$, which is computable from the challenger's commitment values. Note that σ is a valid signature if $c^* = \mathsf{H}_0(g^{\sum_{i=1}^{k-1} r_i}, m^*) = \mathsf{H}_0(h, m^*)$. But from the above we only know that $c^* = \sum_{i=1}^{k-1} c_i = \sum_{i=1}^{k-1} \mathsf{H}_0(\bar{t}_i, m)$. So in order to produce a forgery, the adversary needs to find values m, m^* and $\bar{t}_i$ for all $i \in \{1, \ldots, k-1\}$ such that $\mathsf{H}_0(h, m^*) = \sum_{i=1}^{k-1} \mathsf{H}_0(\bar{t}_i, m)$ modulo the group order.

We reduce this to the k-sum problem where list L_i for each $i \in \{1, \ldots, k-1\}$ will contain s_L outputs of $\mathsf{H}_0(\bar{t}_i, m)$, filled by evaluating the hash function on distinct values of $\bar{t}_i$. List L_k will contain s_L outputs of $\mathsf{H}_0(h, m^*)$, each for some distinct value of m^*. This requires that the message space is large enough to produce a sufficient number of list elements for list L_k. In Section 4.2 we show how to extend this attack in case if the message space is limited.

Note that the unforgeability of CoSi can be broken in any tree $\mathcal{T}$ where the challenger controls a non-root node (recall that our security notion allows only a single non-adversarial tree node). Even though the values of PK_j, t_j and s_j propagate and accumulate throughout the tree, the adversary can cancel out the contributed shares from all adversarial nodes regardless of the tree structure, obtaining the shares that belong specifically to the challenger's node.

Furthermore, our attack can be extended to forge signatures in any tree $\mathcal{T}$ where the attacker controls only the root node (note that our security notion does not model this case). In order to do this, the attacker should treat all non-root nodes as a single challenger node, meaning it should multiply together all commitments and add together all responses acquired from its children. Running the above attack would then produce a forgery with respect to the sum of all secret keys that belong to the non-root nodes in the tree. The extended attack requires the same amount of resources as the basic attack against a single challenger node.

Theorem 2. *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order q. Let $\mathcal{A}$ be an adversary that $(\tau, s_\mathrm{L}, \epsilon)$-breaks the k-sum problem in group $(\mathbb{Z}_q, +)$. Then there exists an adversary $\mathcal{B}$ that is a $(\tau', q_\mathrm{S}, q_\mathrm{C}, q_\mathrm{H}, \epsilon)$-forger for CoSi in group $\mathbb{G}$, where $\tau' = \tau + O(k \cdot s_\mathrm{L} \cdot \lg q)$, $q_\mathrm{S} = q_\mathrm{C} = k - 1$ and $q_\mathrm{H} = k \cdot s_\mathrm{L}$.*

Proof (Proof of Theorem 2). We build a CoSi forger $\mathcal{B}$. Recall that adversary $\mathcal{B}$ playing against the unforgeability of CoSi takes par, pk^* as input, and is provided with an access to the random oracle $\mathsf{H}_0 \colon \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_q$ and to the signing oracle $\mathcal{O}^{\mathsf{Sign}(par, sk^*, \cdot, \cdot)}$. Here par are parameters and (pk^*, sk^*) is a challenge key-pair generated by the security game. Upon halting, the adversary has to return $\sigma, m, \mathcal{PK}$, where σ is a forgery for message m with respect to set of public keys $\mathcal{PK}$. Let $\mathcal{B}$ perform the following steps:

- Choose an arbitrary message $m \in \{0,1\}^*$. Set $\mathcal{T}$ equal to a 2-node tree of depth 1 where the leaf node corresponds to the challenge key pair pk^*, sk^*, and the root node is controlled by the adversary. Sample an adversarial key pair $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(par)$ that will be used for the root node. Parse pk^* as (y^*, π^*) and pk as (y, π), then set $PK = y^* \cdot y$. According to the key generation algorithm, we have $PK = g^{sk^* + sk}$.

– Simultaneously initiate $k-1$ queries to the signing oracle $\mathcal{O}^{\mathsf{Sign}(par,sk^*,\cdot,\cdot)}$ with m and $\mathcal{T}$ as input. For each $i \in \{1,\ldots,k-1\}$, pause the i-th concurrent interaction with the signing oracle after the commitment phase, having acquired the commitment value pair (t_i, pk^*) where $t_i = g^{r_i}$ for $r_i \in \mathbb{Z}_q$ that was chosen uniformly at random and is not known to $\mathcal{B}$. Set $h = \prod_{i=1}^{k-1} t_i$.
– Create empty tables $T\colon \{1,\ldots,k-1\} \times \mathbb{Z}_q \to \mathbb{G}$ and $T_k\colon \mathbb{Z}_q \to \mathbb{Z}_q$. For each $i \in \{1,\ldots,k-1\}$ create an empty list L_i and fill it with $x_{i,j} = \mathsf{H}_0(\bar{t}_j, m)$ for all $j \in \{1,\ldots,s_{\mathrm{L}}\}$, such that $\{\bar{t}_1,\ldots,\bar{t}_{s_{\mathrm{L}}}\}$ are distinct elements from $\mathbb{G}$. Set $T[i, x_{i,j}] = \bar{t}_j$ accordingly. Create an empty list L_k and fill it with $x_{k,j} = q - \mathsf{H}_0(h, m_j^*) \bmod q$ for $j \in \{1,\ldots,s_{\mathrm{L}}\}$, such that $\{m_1^*,\ldots,m_{s_{\mathrm{L}}}^*\}$ are distinct elements from $\{0,1\}^* \setminus \{m\}$. Set $T_k[x_{k,j}] = m_j^*$ accordingly.
– Run adversary $\mathcal{A}$ on lists $L_1,\ldots,L_k$ to get a solution to the k-list problem, meaning $\mathcal{A}$ returns to $\mathcal{B}$ a list of elements $c_1,\ldots c_k \in \mathbb{Z}_q$ such that $c_1 + \ldots + c_k \equiv 0 \mod q$, and $c_i \in L_i$ for all $i \in \{1,\ldots,k\}$. Set $m^* = T_k[c_k]$ and $c^* = \mathsf{H}_0(h, m^*)$. Note that $c_1 + \ldots + c_{k-1} \equiv c^* \mod q$ if adversary $\mathcal{A}$ returned a correct solution to the k-sum problem.
– For each $i \in \{1,\ldots,k-1\}$ set $\bar{t}_i = T[i, c_i]$ and pass $(\bar{t}_i, PK)$ as the challenge value pair to the i-th concurrent interaction with the signing oracle to get back the response $s_i = r_i + c_i \cdot sk^*$.
– Compute $s^* = \sum_{i=1}^{k-1} s_i + c^* \cdot sk \bmod q$. Let $\sigma = (c^*, s^*)$, and let $\mathcal{PK} = \{pk^*, pk\}$. Return $\sigma, m^*, \mathcal{PK}$.

Assume that adversary $\mathcal{A}$ returned a valid solution to the k-sum problem. Denote $r^* = \sum_{j=1}^{k-1} r_i$. Then we have $s^* = \sum_{j=1}^{k-1} r_i + \sum_{j=1}^{k-1} c_j \cdot sk^* + c^* \cdot sk = r^* + c^* \cdot (sk^* + sk)$. It follows that $c^* = \mathsf{H}_0(h, m^*) = \mathsf{H}_0(g^{r^*}, m^*) = \mathsf{H}_0(g^{s^*} \cdot PK^{-c^*}, m^*)$, meaning σ is a valid signature for message m^* with respect to the aggregated public key PK.

Parameter Choices When using the k-tree algorithm in the attack against CoSi, we have $n = \lg q$. If the number of parallel signing queries is 3 then the adversary solves the 4-sum problem using $4 \cdot q^{1/3}$ random oracle queries in time $O(q^{1/3})$. Furthermore, if $k = 2^{\sqrt{\lg q}}$ and the adversary uses $(k-1)$ parallel signing oracle queries, then it solves the k-sum problem using $2^{2\sqrt{\lg q}}$ random oracle queries in time $O(2^{2\sqrt{\lg q}})$. More generally, if the adversary uses $(k-1)$ parallel signing queries for any k, then it has to solve the k-sum problem using $q_{\mathrm{H}} = k \cdot 2^{\lg q/(1+\lg k)}$ random oracle queries (to generate k lists of size $s_{\mathrm{L}} = 2^{\lg q/(1+\lg k)}$) and runtime $\tau \in O(k \cdot 2^{\lg q/(1+\lg k)})$.

Producing a forgery for a fixed message In the attack on CoSi described above, the list L_k is filled using distinct values of m^*. This requires that the message space is large enough to produce a sufficient number of list elements for L_k. It also prohibits from producing a forgery for some a priori fixed message m^*. We now extend the attack to remove these restrictions. We will show how to produce a forgery for an arbitrary message m^*.

Compared to the initial attack, the adversary will now construct a signature $\sigma = (c^*, s^*)$ for some message m^* such that $c^* = a \cdot \sum_{i=1}^{k-1} c_i$ and $s^* = a \cdot \sum_{i=1}^{k-1} s_i + c^* \cdot sk$ for an arbitrary group exponent a. The purpose of value a is to re-randomize possible values of c^*, allowing to add a new element to list L_k for each distinct a. We now again show how to choose the $\bar{t}_i$'s such that σ is a valid signature for m^*.

From the above, we get $s^* = a \cdot \sum_{i=1}^{k-1} r_i + c^* \cdot (sk^* + sk)$. Let $h = g^{\sum_{i=1}^{k-1} r_i}$. Note that σ is a valid signature if $c^* = \mathsf{H}_0(g^{a \cdot \sum_{i=1}^{k-1} r_i}, m^*) = \mathsf{H}_0(h^a, m^*)$. But we only know that $c^* = a \cdot \sum_{i=1}^{k-1} c_i = a \cdot \sum_{i=1}^{k-1} \mathsf{H}_0(\bar{t}_i, m)$. So in order to produce a forgery, the adversary needs to find values a, m, m^* and $\bar{t}_i$ for all $i \in \{1,\ldots,k-1\}$ such that $\mathsf{H}_0(h^a, m^*) \cdot a^{-1} = \sum_{i=1}^{k-1} \mathsf{H}_0(\bar{t}_i, m)$ modulo the group order.

We reduce this to the k-sum problem where list L_i for each $i \in \{1,\ldots,k-1\}$ will contain s_{L} outputs of $\mathsf{H}_0(\bar{t}_i, m)$, filled by evaluating the hash function on distinct values of $\bar{t}_i$. List L_k will contain s_{L} outputs of $\mathsf{H}_0(h^a, m^*) \cdot a^{-1}$, each for some distinct value a.

The following steps change in the reduction to the k-sum problem:

– In the first step of the attack, adversary $\mathcal{B}$ now chooses two arbitrary distinct messages $m, m^* \in \{0,1\}^*$.
– In the third step of the attack, the list L_k is now populated with $x_{k,j} = q - \mathsf{H}_0(h^{a_j}, m^*) \cdot a_j^{-1} \bmod q$ for $j \in \{1,\ldots,s_{\mathrm{L}}\}$, such that $\{a_1,\ldots,a_{s_{\mathrm{L}}}\}$ are distinct elements from $\mathbb{Z}_q$. Set $T_k[x_{k,j}] = a_j$ accordingly.

– In the fourth step of the attack, $\mathcal{B}$ runs $\mathcal{A}$ to acquire a list of elements $c_1, \ldots c_k \in \mathbb{Z}_q$ as before. It sets $a = T_k[c_k]$ and $c^* = \mathsf{H}_0(h^a, m^*)$. Note that $a(c_1 + \ldots + c_{k-1}) \equiv c^* \mod q$ if adversary $\mathcal{A}$ returned a correct solution to the k-sum problem.
– In the final step of the attack, compute $s^* = a \cdot \sum_{i=1}^{k-1} s_i + c^* \cdot sk \bmod q$. Let $\sigma = (c^*, s^*)$, and let $\mathcal{PK} = \{pk^*, pk\}$. Return $\sigma, m^*, \mathcal{PK}$.

Assume that adversary $\mathcal{A}$ returned a valid solution to the k-sum problem. Denote $r^* = a \cdot \sum_{j=1}^{k-1} r_i$. Then we have $s^* = a \cdot \sum_{j=1}^{k-1} r_i + a \cdot \sum_{j=1}^{k-1} c_j \cdot sk^* + c^* \cdot sk = r^* + c^* \cdot (sk^* + sk)$. It follows that $c^* = \mathsf{H}_0(h^a, m^*) = \mathsf{H}_0(g^{r^*}, m^*) = \mathsf{H}_0(g^{s^*} \cdot PK^{-c^*}, m^*)$, meaning σ is a valid signature for message m^* with respect to the aggregated public key PK.

Note that if $h = 1$ then L_k will contain elements that are not sampled uniformly and independently. This happens with probability $1/q$. Therefore, the success probability of $\mathcal{B}$ is $(1 - 1/q)$ times that of $\mathcal{A}$.

4.3 Applicability to MuSig

The two-round version of MuSig [MPSW18a] can be attacked in the same way as CoSi. Compared to CoSi, the main change in MuSig is their use of a more complicated key aggregation. This affects the above attack only syntactically, so we omit the details. Our attack does not affect the updated three-round version of MuSig [MPSW18b].

4.4 Applicability to BCJ

The attack against BCJ closely follows the attack against CoSi. We consider a 2-node tree $\mathcal{T}$, where the challenger controls the leaf node. Let PK be the aggregated public key of $\mathcal{T}$, and let m be an arbitrary message. Let sk be the secret key of the root node (controlled by the adversary). The adversary starts $k-1$ concurrent signing oracle queries with the challenger, obtaining values $t_{1,i}$ and $t_{2,i}$ for all $i \in \{1, \ldots, k-1\}$, where i denotes an index of the signing oracle query. Next, the adversary creates lists $L_1, \ldots, L_k$. For each $i \in \{1, \ldots, k-1\}$ the list L_i is filled with $\mathsf{H}_0(t_{1,i} \cdot g_1^{\alpha_1} h_1^{\alpha_2}, t_{2,i} \cdot g_2^{\alpha_1} h_2^{\alpha_2} g_1^r, PK, m)$ for arbitrary distinct tuples $(\alpha_1, \alpha_2, r) \in \mathbb{Z}_q^3$. Let $\bar{t}_1^* = \prod_{i=1}^{k-1} t_{1,i}$ and $\bar{t}_2^* = \prod_{i=1}^{k-1} t_{2,i}$. The list L_k is filled with $-\mathsf{H}_0(\bar{t}_1^*, \bar{t}_2^*, PK, m^*)$ for distinct messages $m^* \in \{0,1\}^*$ such that $m^* \neq m$. The adversary chooses the values for the challenge phase based on the solution to the k-sum problem, and gets back responses s_i for all $i \in \{1, \ldots, k-1\}$. Let m^* be the message that corresponds to the element of L_k that is included in the solution to the k-sum problem, and let $c^* = \mathsf{H}_0(\bar{t}_1^*, \bar{t}_2^*, PK, m^*)$. Then the adversary returns a forged signature $\sigma^* = (\bar{t}_1^*, \bar{t}_2^*, s^*, \gamma_1^*, \gamma_2^*)$ for m^*, where $s^* = \sum_{i=1}^{k-1} s_i + c^* \cdot sk$ and γ_1^*, γ_2^* are the sums of the corresponding γ values returned by the challenger at the end of the signing oracle queries.

4.5 Applicability to MWLD

The attack is again similar to the attack on CoSi. We consider a 2-node tree $\mathcal{T}$, where the challenger controls the leaf node with public key pk^*, and the adversary controls the root node with keys pk, sk_1, sk_2. Let $L = \{pk^*, pk\}$, and let m be an arbitrary message. The adversary starts $k-1$ concurrent signing oracle queries with the challenger, obtaining a commitment t_i for each $i \in \{1, \ldots, k-1\}$, where i denotes an index of the signing oracle query. Next, the adversary creates lists $L_1, \ldots, L_k$. For each $i \in \{1, \ldots, k-1\}$ the list L_i is filled with $\mathsf{H}_1(\mathsf{H}_0(\bar{t}_i, L, m), pk^*)$ for distinct values of $\bar{t}_i \in \mathbb{G}$. The list L_k is filled with $-\mathsf{H}_1(\mathsf{H}_0(\prod_{i=1}^{k-1} t_i, L, m^*), pk^*)$ for distinct messages $m^* \in \{0,1\}^*$ such that $m^* \neq m$. The adversary chooses the values for the challenge phase based on the solution to the k-sum problem, and gets back responses $s_{1,i}$ and $s_{2,i}$ for all $i \in \{1, \ldots, k-1\}$, where i denotes the number of a concurrent oracle query. Let m^* be the message that corresponds to the element of L_k that is included in the solution to the k-sum problem. Let $c^* = \mathsf{H}_0(\prod_{i=1}^{k-1} t_i, L, m^*)$ and let $v^* = \mathsf{H}_1(c^*, pk^*)$. Then adversary produces a forgery $\sigma = (c^*, s_1^*, s_2^*)$ for message m^*, where $s_b^* = \sum_{i=1}^{k-1} s_{b,i} + v^* \cdot sk_b$ for each $b \in \{1, 2\}$.

4.6 Attack on RandHound

RandHound is a public-randomness generator introduced in [SJKK+17]. The protocol allows a client to interact with a set of RandHound servers to generate a random string. It uses CoSi to ensure that the result is publicly verifiable. Our attack against CoSi allows the client to forge a transcript that serves as a third-party verifiable proof of the produced randomness. Hence the client can maliciously claim to have acquired randomness that is beneficial to them.

In the RandHound protocol, the client communicates with each of the servers directly. Let n denote the number of the servers participating in the protocol. Then the protocol runs the CoSi scheme for an $(n+1)$-node tree $\mathcal{T}$ of depth 1 where the client is the leader (i.e. is represented by the root node), and each of the n servers is represented by a leaf node directly connected to the root. Our attack against the RandHound protocol will essentially run the CoSi attack against each of the n servers.

In Section 4.2 we discussed that the CoSi attack can be extended to the case when the adversary controls only the root node. This attack would immediately work against RandHound if the protocol always used the same set of servers. However, the set of servers that participate in RandHound may change in each separate execution of the protocol. To circumvent this limitation, we now show that it is possible to produce a forgery with respect to an arbitrary set of servers in RandHound, even if they never produced a joint signature. The attacker will run the CoSi attack against each of the servers separately, and then construct a signature with respect to all servers simultaneously. This will require the adversary to open $k-1$ concurrent signing queries to each of the n servers, for a total of $n \cdot (k-1)$ concurrent queries. The adversary will then run a variant of the earlier CoSi attack n times, each involving the $k-1$ concurrent signing queries to a single targeted server at a time.

The RandHound protocol uses the CoSi scheme to sign messages that contain a unique session identifier, along with information about the subset of servers whose secret shares will be used to compute shared randomness. In order to attack RandHound, an adversary has to choose some fixed message m^* and forge a valid signature $\sigma = (c^*, s^*)$ for this message with respect to the set $\mathcal{PK} = \{pk_1^*, \ldots, pk_n^*\}$ containing a public key for each of the n servers.

We now describe the attack. For each $z \in \{1, \ldots, n\}$ let $\mathcal{T}_z$ be a 2-node tree where the adversary controls the root node with a secret key sk, and the challenger is the z-th of n servers controlling the leaf node with some secret key sk_z^* (the client does not participate in the RandHound signature, but for notational consistency here we assume otherwise). Adversary chooses an arbitrary message m^*, and for each $z \in \{1, \ldots, n\}$ concurrently runs a variant of the earlier CoSi attack in tree $\mathcal{T}_z$ as follows:

- The adversary chooses mutually distinct messages $m_{z,1}, \ldots, m_{z,k-1} \in \{0,1\}^* \setminus \{m^*\}$ and initiates $k-1$ concurrent calls to the signing oracle, using $m_{z,i}, \mathcal{T}_z$ as an input to the oracle query, one for each $i \in \{1, \ldots, k-1\}$.
- At the end of the commitment phase, adversary receives a commitment $t_{z,i} = g^{r_{z,i}}$ for each concurrent oracle call $i \in \{1, \ldots, k-1\}$, where $r_{z,i}$ is not known to the adversary.
- At the beginning of the challenge phase the adversary chooses and distributes a malicious $\bar{t}_{z,i}$ that is used to compute the challenge $c_{z,i} = \mathsf{H}_0(\bar{t}_{z,i}, m_{z,i})$ for each $i \in \{1, \ldots, k-1\}$ (we show how to choose $\bar{t}_{z,i}$'s below).
- During the response phase the adversary obtains from $\mathcal{T}_z$'s leaf node $s_{z,i} = r_{z,i} + c_{z,i} \cdot sk_z^*$ for each $i \in \{1, \ldots, k-1\}$.

The adversary constructs the signature $\sigma = (c^*, s^*)$ for message m^* where $c^* = \mathsf{H}_0(\prod_{z=1}^{n} \prod_{i=1}^{k-1} t_{z,i}, m^*)$ and $s^* = \sum_{z=1}^{n} \sum_{i=1}^{k-1} s_{z,i} + c^* \cdot sk$. We now show how to choose the $\bar{t}_{z,i}$'s such that σ is a valid signature for m^*.

Let $r^* = \sum_{z=1}^{n} \sum_{i=1}^{k-1} r_{z,i}$. By expanding the above, we get $c^* = \mathsf{H}_0(g^{r^*}, m^*)$ and $s^* = r^* + \sum_{z=1}^{n} \sum_{i=1}^{k-1} c_{z,i} \cdot sk_z^* + c^* \cdot sk$. Note that σ is a valid signature if $\sum_{i=1}^{k-1} c_{z,i} = c^*$ for each $z \in \{1, \ldots, n\}$, which would give us $s^* = r^* + c^* \cdot (\sum_{z=1}^{n} sk_z^* + sk)$. So in order to produce a forgery, for each $z \in \{1, \ldots, n\}$ the adversary needs to find values $\bar{t}_{z,i}$ for all $i \in \{1, \ldots, k-1\}$ such that $c^* = \sum_{i=1}^{k-1} \mathsf{H}_0(\bar{t}_{z,i}, m_{z,i})$ modulo the group order.

This can be reduced to a variant of the k-sum problem that given a constant c^* and $k-1$ lists $L_1, \ldots, L_{k-1}$ of uniformly random elements, requires to find $x_1 \in L_1, \ldots, x_{k-1} \in L_{k-1}$ such that $x_1 + \ldots + x_{k-1} \equiv c^*$

mod q. Here list L_i for each $i \in \{1, \ldots, k-1\}$ will contain outputs of $\mathsf{H}_0(\bar{t}_{z,i}, m_{z,i})$ for some distinct values of $\bar{t}_{z,i}$. This problem can be solved using the k-tree algorithm [Wag02] just as before, but finding the solution requires more resources because we now lost one degree of freedom (the choice of elements from the k-th list), so each of the remaining lists will have to be populated with a larger number of elements.

5 A secure Two-Round Multi-Signature from BCJ

The metareduction in Section 3 shows that none of the existing two-round multi-signature schemes (without pairings) can be proved secure, raising the question whether such schemes can exist at all. In this section we answer that question in the positive by presenting a modification of the BCJ scheme that avoids the pitfalls exploited by the metareduction. Namely, the metareduction exploits the fact that, when the reduction rewinds the forger, the forger can extract the signing key from the reduction, or some other information that can be used to forge signatures. In the following scheme, we ensure that the extracted information can only be used to create new signatures on the same message as the one being signed, but cannot be used to sign different messages.

5.1 The Modified Scheme

The mBCJ scheme uses the same homomorphic commitment scheme to let signers commit to their first rounds as the BCJ scheme, but lets the parameters of the commitment scheme be given by the message through a random oracle. Namely, the scheme uses three hash functions $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{G}^3$

Parameters generation. Pg sets up a group $\mathbb{G}$ of order q with generator g_1, where q is a κ-bit prime. Output $par \leftarrow (\mathbb{G}, g_1, q)$.

Key generation. $\mathsf{Kg}(par)$ chooses $sk \xleftarrow{\$} \mathbb{Z}_q$ and sets $y \leftarrow g_1^{sk}$. The proof-of-possession $\pi = (c, s)$ is computed by choosing $r \xleftarrow{\$} \mathbb{Z}_q$ and computing $c \leftarrow \mathsf{H}_1(y, g_1^r)$ and $s \leftarrow r + c \cdot sk \bmod q$. Let $pk \leftarrow (y, \pi)$ and output (pk, sk).

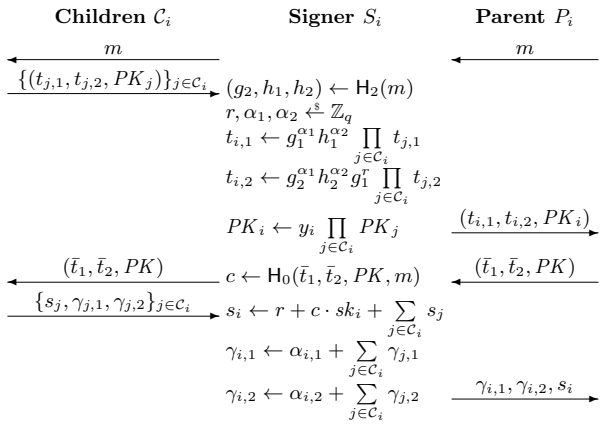

Fig. 3. The mBCJ signing protocol for signer S_i with secret key sk_i and public key $pk = (y_i, \pi_i)$. If S_i is the leader then, instead of sending $(t_{i,1}, t_{i,2}, PK_i)$ to its parent, it sends $(\bar{t}_1, \bar{t}_2, PK) = (t_{i,1}, t_{i,2}, PK_i)$ to its children, and instead of sending $(s_i, \gamma_{i,1}, \gamma_{i,2})$ to its parent, it outputs $(\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2) = (\bar{t}_1, \bar{t}_2, s_i, \gamma_{i,1}, \gamma_{i,2})$ as the signature.

Signing. The two-round signing protocol is depicted in Figure 3. More precisely, a signer S_i running $\mathsf{Sign}(par, (sk_i, y_i, \pi_i), m, \mathcal{T})$ behaves as follows.
Announcement. If S_i is the leader, then it sends a unique session identifier *ssid* to its children. Otherwise, it waits for this message and forwards it to its set of children $\mathcal{C}_i$ in $\mathcal{T}$.

Commitment. S_i waits to receive all values $(t_{j,1}, t_{j,2}, PK_j)$ for $j \in \mathcal{C}_i$. It then generates the commitment parameters $(g_2, h_1, h_2) \leftarrow \mathsf{H}_2(m)$, chooses $(r_i, \alpha_{i,1}, \alpha_{i,2}) \xleftarrow{\$} \mathbb{Z}_q^3$, and computes $t_{i,1} \leftarrow g_1^{\alpha_{i,1}} h_1^{\alpha_{i,2}} \cdot \prod_{j \in \mathcal{C}_i} t_{j,1}$ and $t_{i,2} \leftarrow g_2^{\alpha_{i,1}} h_2^{\alpha_{i,2}} \cdot g_1^{r_i} \cdot \prod_{j \in \mathcal{C}_i} t_{j,2}$, and $PK_i \leftarrow y_i \cdot \prod_{j \in \mathcal{C}_i} PK_j$. If S_i is not the leader, it sends $(t_{i,1}, t_{i,2}, PK_i)$ to its parent, otherwise it proceeds to the next phase.
Challenge. If S_i is the leader, it sets $\bar{t}_1 \leftarrow t_{i,1}$, $\bar{t}_2 \leftarrow t_{i,2}$, and $PK \leftarrow PK_i$. It computes $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, and sends $(\bar{t}_1, \bar{t}_2, PK)$ to its children. If S_i is not the leader, it waits to receive a message $(\bar{t}_1, \bar{t}_2, PK)$, computes $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, and sends $(\bar{t}_1, \bar{t}_2, PK)$ to its children.
Response. S_i waits to receive all values $(s_j, \gamma_{j,1}, \gamma_{j,2})$ for $j \in \mathcal{C}_i$ (note that if S_i is a leaf it will proceed immediately), and then computes $s_i \leftarrow r_i + c \cdot sk_i + \sum_{j \in C_i} s_j$, $\gamma_{i,b} \leftarrow \alpha_{i,b} + \sum_{j \in C_i} \gamma_{j,b}$ for $b \in \{1, 2\}$. It sends $(s_i, \gamma_{i,1}, \gamma_{i,2})$ to its parent, unless S_i is the root, then S_i sets $s \leftarrow s_i$, $\gamma_1 \leftarrow \gamma_{i,1}$, $\gamma_2 \leftarrow \gamma_{i,2}$, and outputs $\sigma \leftarrow (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$.

Key Aggregation. KAg on input a set of public keys $\mathcal{PK}$ parses every $pk_i \in \mathcal{PK}$ as $(y_i, (c_i, s_i))$, and if this public key has not been validated before, check that $c_i = \mathsf{H}_1(y_i, g_1^{s_i} y_i^{-c_i})$. Output aggregate public key $PK \leftarrow \prod y_i$.

Verification. Vf on input aggregate public key PK, a signature $\sigma = (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$, and a message m, compute $(g_2, h_1, h_2) \leftarrow \mathsf{H}_2(m)$, $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$ and check that $\bar{t}_1 \stackrel{?}{=} g_1^{\gamma_1} h_1^{\gamma_2}$ and $\bar{t}_2 \stackrel{?}{=} g_2^{\gamma_1} h_2^{\gamma_2} g_1^{s} PK^{-c}$.

5.2 Security

To prove security of the mBCJ scheme, we construct an algorithm $\mathcal{A}$ around the forger $\mathcal{F}$ that simulates the random oracle $\mathsf{H}_3(m)$ either with commitment parameters with known trapdoors, or with binding commitment parameters that are a re-randomization of the challenge public key. It then hopes that all signing queries are on messages with commitment parameters of the first type, so that it can use the trapdoor to equivocate commitments and simulate signatures. If moreover the forgery is on a message of the second type, then we show that $\mathcal{A}$ can be used in a forking lemma to compute discrete logarithms.

Theorem 3. mBCJ *is a secure multi-signature scheme in the key-verification setting under the discrete-logarithm assumption in the random-oracle model. More precisely,* mBCJ *is* $(\tau, q_S, q_C, q_H, \epsilon)$*-unforgeable in the random-oracle model if* $q > 64e(N+1)(q_H + q_S + 1)(q_S + 1)/\epsilon$ *and if the discrete-logarithm problem is* $(\tau \cdot 64e(N+1)^2(q_H + q_S + 1)(q_S + 1)/\epsilon \cdot \ln(8e(N+1)(q_S+1)/\epsilon), \epsilon/(8e(q_S+1)))$*-hard, where* N *is the maximum number of signers involved in a single multi-signature,* e *is the base of the natural logarithm, and* τ_{exp} *is the time of a multi-exponentiation in* $\mathbb{G}$*.*

Proof. Given a $(\tau, q_S, q_C, q_H, \epsilon)$ forger $\mathcal{F}$ against the mBCJ multi-signature scheme, consider an input generator IG that outputs $y^* \xleftarrow{\$} \mathbb{G}$ and an algorithm $\mathcal{A}$ that on input y^* and randomness $f = (\rho, h_1, \ldots, h_{q_H})$ proceeds as follows.

Algorithm $\mathcal{A}$ simulates the self-signed certificate $\pi^* = (d^*, w^*)$ by picking $d^*, w^* \xleftarrow{\$} \mathbb{Z}_q$ and computing $v^* \leftarrow g_1^{d^*} y^{*-d^*}$. It then runs the forger $\mathcal{F}$ on input $pk^* = (y^*, \pi^*)$ with random tape ρ. It responds to $\mathcal{F}$'s k-th random-oracle query to H_0 or H_1 with h_k, except when $\mathcal{F}$ makes a query $\mathsf{H}_1(y^*, v^*)$ it responds with d^*. Moreover, when $\mathcal{F}$ makes a query $\mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, then $\mathcal{A}$ internally simulates a query $\mathsf{H}_3(m)$. We assume that $\mathcal{A}$ never makes the same random-oracle query twice.

For each query $\mathsf{H}_3(m)$, tosses a biased coin that comes out heads with probability δ and tails with probability $1 - \delta$. If the coin comes out heads, then it chooses $\omega_1, \omega_2, \omega_3 \xleftarrow{\$} \mathbb{Z}_q$, computes $h_1 \leftarrow g_1^{\omega_1}$, $g_2 \leftarrow g_1^{\omega_2}$, and $h_2 \leftarrow y^{*\omega_3}$, stores $(m, 0, (\omega_1, \omega_2, \omega_3))$, and returns (h_1, g_2, h_2). If it comes out tails, then $\mathcal{A}$ chooses $\omega \xleftarrow{\$} \mathbb{Z}_q$ and $g_2, h_2 \xleftarrow{\$} \mathbb{G}$, computes $h_1 \leftarrow y^{*\omega}$, stores $(m, 1, \omega)$, and returns (h_1, g_2, h_2).

To answer signing queries for a message m as signer S_i in a tree $\mathcal{T}$, $\mathcal{A}$ proceeds as follows. If $\mathcal{F}$ did not yet query $\mathsf{H}_3(m)$, then $\mathcal{A}$ internally simulates such a query. If $\mathcal{A}$ does not have a record $(m, 0, \omega_1, \omega_2, \omega_3)$, then it aborts by outputting $(\emptyset, \emptyset)$. Otherwise, it receives all values $(t_{j,1}, t_{j,2}, PK_j)$ from its children $j \in \mathcal{C}_i$ in $\mathcal{T}$, chooses $\hat{r}_i, \hat{s}_i, \hat{c} \xleftarrow{\$} \mathbb{Z}_q$ and computes $t_{i,1} \leftarrow g_1^{\hat{r}_i} \cdot \prod_{j \in \mathcal{C}_i} t_{j,1}$ and $t_{i,2} \leftarrow g_1^{\hat{s}_i} y^{*\hat{c}} \cdot \prod_{j \in \mathcal{C}_i} t_{j,2}$.

When the forger sends $(\bar{t}_1, \bar{t}_2, PK)$, $\mathcal{A}$ simulates a query $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$, then $\mathcal{A}$ waits for its children $j \in \mathcal{C}_i$ to send $(s_j, \gamma_{j,1}, \gamma_{j,2})$ and computes $\gamma'_{i,2} \leftarrow (c + \hat{c})/\omega_3 \bmod q$, $\gamma'_{i,1} \leftarrow \hat{r}_i - \omega_1 \gamma'_{i,2} \bmod q$, and

$s'_i \leftarrow \hat{s}_i - \omega_2\gamma'_{i,1} \bmod q$. It then sends $\gamma_{i,1} \leftarrow \gamma'_{i,1} + \sum_{j\in\mathcal{C}_i}\gamma_{j,1} \bmod q$, $\gamma_{i,2} \leftarrow \gamma'_{i,2} + \sum_{j\in\mathcal{C}_i}\gamma_{j,2} \bmod q$, and $s_i \leftarrow s'_i + \sum_{j\in\mathcal{C}_i} s_j \bmod q$ to its parent. One can see that, if the co-signers behave honestly, these values yield a valid signature because

$$\begin{aligned} g_1^{\gamma_{i,1}-\sum_{j\in\mathcal{C}_i}\gamma_{j,1}} h_1^{\gamma_{i,2}-\sum_{j\in\mathcal{C}_i}\gamma_{j,2}} &= g_1^{\gamma'_{i,1}} h_1^{\gamma'_{i,2}} \\ &= g_1^{\hat{r}_i-\omega_1\gamma'_{i,2}} g_1^{\omega_1\gamma'_{i,2}} \\ &= g_1^{\hat{r}_i} \\ &= t_{i,1}/\prod_{j\in\mathcal{C}_i} t_{j,1} \end{aligned}$$

and

$$\begin{aligned} g_2^{\gamma_{i,1}-\sum_{j\in\mathcal{C}_i}\gamma_{j,1}} h_2^{\gamma_{i,2}-\sum_{j\in\mathcal{C}_i}\gamma_{j,2}} g_1^{s_i-\sum_{j\in\mathcal{C}_i} s_j} y^{*-c} &= g_2^{\gamma'_{i,1}} h_2^{\gamma'_{i,2}} g_1^{s'_i} y^{*-c} \\ &= g_1^{\omega_2\gamma'_{i,1}} y^{*c+\hat{c}} g_1^{\hat{s}_i-\omega_2\gamma'_{i,1}} y^{*-c} \\ &= g_1^{\hat{s}_i} y^{*\hat{c}} \\ &= t_{i,2}/\prod_{j\in\mathcal{C}_i} t_{j,2} \end{aligned}$$

as in the real signing protocol. Moreover, the responses are correctly distributed due to the random choices of $\hat{r}_i$, $\hat{s}_i$, and $\hat{c}$.

When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ outputs `fail`. Otherwise, let $\mathcal{F}$'s forgery is $\sigma = (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$ on message m^* for a set of public keys $\mathcal{PK}^*$. Without loss of generality, we assume that $\mathcal{F}$ makes all hash queries involved in verifying the forgery and the proofs of possession in $\mathcal{PK}^*$. If there doesn't exist a record $(m^*, 1, \omega)$, then $\mathcal{A}$ aborts by outputting $(\emptyset, \emptyset)$. Otherwise, let j_f be the index of $\mathcal{A}$'s random-oracle query $\mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m^*) = h_{j_\mathrm{f}} = c$ where $PK^* = \prod_{(y,\pi)\in\mathcal{PK}^*} y$, and let $out_{j_\mathrm{f}} = (\bar{t}_1, \bar{t}_2, c, s, \gamma_1, \gamma_2, \mathcal{PK}, \omega)$. For each $pk = (y, (d, w)) \in \mathcal{PK}^* \setminus \{pk^*\}$, let j_y be the index of $\mathcal{A}$'s random-oracle query $\mathsf{H}_1(y, v) = d$ where $v = g_1^w y^{-d}$, and let $out_{j_y} = (y, v, d, w)$. Algorithm $\mathcal{A}$ outputs $(J, \{out_j\}_{j\in J}$, where $J = \{j_y\}_{(y,\pi)\in\mathcal{PK}^*} \cup \{j_\mathrm{f}\}$.

Algorithm $\mathcal{A}$ runs in time at most $\tau + (3q_\mathrm{H} + 5q_\mathrm{S} + N + 6)\cdot\tau_\mathrm{exp} + O(Nq_\mathrm{S} + N)$ and succeeds whenever $\mathcal{F}$ succeeds and $\mathcal{A}$ doesn't abort prematurely, meaning that all $\mathcal{F}$'s signing queries were on messages m where the biased coin in $\mathsf{H}_3(m)$ came out heads, and the one in $\mathsf{H}_3(m^*)$ came out tails. The success probability of $\mathcal{A}$ is therefore $\epsilon_\mathcal{A} \geq \delta^{q_\mathrm{S}}(1-\delta)\cdot\epsilon$. Using $\delta = q_\mathrm{S}/(q_\mathrm{S}+1)$, we have that

$$\epsilon_\mathcal{A} \;\geq\; \left(\frac{1}{1+1/q_\mathrm{S}}\right)^{q_\mathrm{S}} \cdot \frac{1}{q_\mathrm{S}+1}\cdot\epsilon \;\geq\; \frac{\epsilon}{e(q_\mathrm{S}+1)}\ ,$$

where we use the fact that $(1/(1+1/q_\mathrm{S}))^{q_\mathrm{S}} \geq e$ for $q_\mathrm{S} > 0$.

We prove the theorem by constructing an algorithm $\mathcal{B}$ that, on input a group element y^* and given a forger $\mathcal{F}$, solves the discrete logarithm problem in $\mathbb{G}$. Namely, $\mathcal{B}$ runs the generalized forking algorithm $\mathcal{GF}_\mathcal{A}$ from Lemma 1 on input y^* with the algorithm $\mathcal{A}$ described above. If $\mathcal{GF}_\mathcal{A}$ outputs `fail`, then $\mathcal{B}$ also outputs `fail`. If $\mathcal{GF}_\mathcal{A}$ outputs $(J, \{out_j\}_{j\in J}, \{out'_j\}_{j\in J})$, then $\mathcal{B}$ proceeds as follows.

Let $out_{j_\mathrm{f}} = (\bar{t}_1, \bar{t}_2, c, s, \gamma_1, \gamma_2, \mathcal{PK}^*, \omega)$ and $out'_{j_\mathrm{f}} = (\bar{t}'_1, \bar{t}'_2, c', s, \gamma'_1, \gamma'_2, \mathcal{PK}^{*\prime}, \omega')$ be the two outputs of $\mathcal{A}$ related to the forgery. For every $(y, \pi) \in \mathcal{PK}^* \setminus \{pk^*\}$, there are two outputs $out_{j_y} = (y, v, d, w)$ and $out'_{j_y} = (y', v', d', w')$ such that $g_1^w = vy^d$ and $g_1^{w'} = v'y'^{d'}$. From the construction of $\mathcal{GF}_\mathcal{A}$, we know that $y = y'$ and $v = v'$ because they were part of the arguments to $\mathcal{F}$'s j_y-th random-oracle query. Dividing the two verification equations yields $g_1^{w-w'} = y^{d-d'}$, from which we can compute the secret key $sk_y = (w-w')/(d-d') \bmod q$ such that $y = g_1^{sk_y}$, because by the construction of $\mathcal{GF}_\mathcal{A}$ we have that $d \neq d'$.

Let $out_{j_\mathrm{f}} = (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2, \mathcal{PK}^*, \omega)$ and $out'_{j_\mathrm{f}} = (\bar{t}'_1, \bar{t}'_2, s', \gamma'_1, \gamma'_2, \mathcal{PK}^*, \omega')$. From the validity of the forgeries, we have that $\bar{t}_1 = g_1^{\gamma_1} y^{*\omega\gamma_2}$ and $\bar{t}'_1 = g_1^{\gamma'_1} y^{*\omega'\gamma'_2}$. From the construction of $\mathcal{GF}_\mathcal{A}$, we know that $\bar{t}_1 = \bar{t}'_1$,

and because $\mathcal{A}$ internally simulates a query $\mathsf{H}_3(m)$ as soon as a query $\mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$ is made, we also have that $\omega = \omega'$. We therefore have that $g_1^{\gamma_1 - \gamma_1'} = y^{*\omega(\gamma_2' - \gamma_2)}$. If $\gamma_2 \neq \gamma_2'$, then $\mathcal{B}$ outputs the discrete logarithm of y^* as

$$\frac{\gamma_1 - \gamma_1'}{\omega(\gamma_2' - \gamma_2)} \bmod q \ .$$

If $\gamma_2 = \gamma_2'$, then by the fact that $g_1^{\gamma_1 - \gamma_1'} = y^{*\omega(\gamma_2' - \gamma_2)}$ it must also hold that $\gamma_1 = \gamma_1'$. The validity of the signatures gives us that

$$\bar{t}_2 = g_2^{\gamma_1} h_2^{\gamma_2} g_1^s PK^{-c} \text{ and } \bar{t}_2' = {g'}_2^{\gamma_1'} {h'}_2^{\gamma_2'} g_1^s {PK'}^{-c'} \ ,$$

where g_2, h_2, g_2', h_2' are determined by the output of $\mathsf{H}_3(m^*)$ latest during the simulation of $\mathcal{F}$'s j_{f}-th random-oracle query. By the construction of $\mathcal{GF}_{\mathcal{A}}$, we therefore have that $(g_2, h_2, \bar{t}_2, PK) = (g_2', h_2', \bar{t}_2', PK')$ and $c \neq c'$ so that dividing the two equations above yields

$$g_1^{s-s'} = PK^{c-c'} \ .$$

We also have that $PK = \prod_{(y,\pi)\in\mathcal{PK}^*} y = g_1^{\sum_{(y,\pi)\in\mathcal{PK}^*} sk_y}$, where $\mathcal{B}$ already extracted all sk_y except sk_{y^*}. It can therefore compute the discrete logarithm of y^* as

$$\frac{s - s'}{c - c'} - \sum_{(y,\pi)\in\mathcal{PK}^*\setminus\{(y^*,\pi^*)\}} sk_y \bmod q \ .$$

The running time of $\mathcal{B}$ is at most that of $\mathcal{GF}_{\mathcal{A}}$ plus $O(N)$ operations, so that the bounds in the theorem follow from Lemma 1.

5.3 Variants and Caveats

Obtaining security in the plain public-key model. The mBCJ scheme as described above thwarts rogue-key attacks in the key-verification model by letting signers add self-signed certificates to their public keys. Alternatively, one could prevent such attacks in the plain public-key model (i.e., without requiring certificates) by using a different hash values as exponents for each public key in the verification equation [BN06], or by using a product of hash values as exponents [MWLD10,MPSW18a]. However, these schemes would be less efficient in terms of verification and key aggregation time, respectively, because they would require a number of exponentiations that is linear in the group size for large or frequently changing groups.

Simplifications for the KOSK model. Because security in the key-verification setting implies security in the KOSK setting, mBCJ can readily be used in the KOSK model, and we can even simplify the scheme a bit. Most importantly, the self-signed credentials preventing rogue-key attacks are no longer neccessary, as these are avoided by the KOSK setting. Also, the aggregate public key PK no longer needs to be included in the hash and setting $c \leftarrow \mathsf{H}_0(\bar{t}_1, \bar{t}_2, m)$ is sufficient. In the key-verification setting we needed PK to be included in the hash to be able to "divide out" the signatures of cosigners and extract a solution to the DL problem. In the KOSK setting this is much simpler, as we know the secret key of every corrupt signer, and PK can be omitted. This saves some bandwith as PK no longer has to be propagated down the tree of signers. We stress that this simplified scheme should only be used in a setting where one is assured that every key is honestly generated.

Extension to multi-sets. It is also easy to extend the mBCJ scheme to multi-sets of signers, where each signer can participate multiple times in the same signing protocol. In a highly distributed setting, this could offer the advantage that signers do not have to keep track in which signing protocol they already participated. The key aggregation algorithm would simply have to be modified to compute $PK \leftarrow \prod_{pk\in\mathcal{PK}} y^{n_y}$ as the aggregate public key, where n_y is the multiplicity of public key y in the multi-set $\mathcal{PK}$.

Note that this extension is only secure because mBCJ includes the aggregate public key PK in the hash $\mathsf{H}_0(\bar{t}_1, \bar{t}_2, PK, m)$. Without including PK, as was done for example in the CoSi scheme, the extension to multi-sets becomes insecure, because a signature on message m and public key y is easily transformed into a valid signature on a different message m' for public key y with multiplicity $c/c' = \mathsf{H}_0(\bar{t}_1, \bar{t}_2, m)/\mathsf{H}_0(\bar{t}_1, \bar{t}_2, m') \bmod q$.

Collision attacks. Bagherzandi et al.'s forking lemma [BCJ08] imposes that the random oracle H maps into the full exponent set of $\mathbb{Z}_q$, where q is typically a 256-bit prime, rather than a subset $\mathbb{Z}_{2^\ell}$ for $\ell < |q|$. Standard Schnorr signatures are well known to remain secure for much shorter hash outputs, around 128 bits [Sch91,NSW09] because their security does not rely on the collision resistance of the hash function. It is worth noting that the same is *not* true for the case of multi-signatures, because unlike standard signatures, collisions in the hash function actually do lead to forgeries on the multi-signature scheme.

Namely, consider a forger $\mathcal{F}$ that performs a signing query for a message m and a tree of signers where $\mathcal{F}$ is the leader with an honestly generated public key $y = g_1^{sk}$ and the honest signer with public key y^* the only child. On input m, the honest signer returns $t_{1,1}, t_{1,2}$. The forger then repetitively generates random values $\alpha_{2,1}, \alpha_{2,2}, r_2 \xleftarrow{\$} \mathbb{Z}_q$ and computes $\bar{t}_1 \leftarrow g_1^{\alpha_{2,1}} h_1^{\alpha_{2,2}} \cdot t_{1,1}$ and $\bar{t}_2 \leftarrow g_2^{\alpha_{2,1}} h_2^{\alpha_{2,2}} \cdot g_1^{r_2} \cdot t_{1,2}$ and hash values $\mathsf{H}(\bar{t}_1, \bar{t}_2, PK, m)$ and $\mathsf{H}(\bar{t}_1, \bar{t}_2, PK, m')$ for $PK = y \cdot y^*$ and $m' \neq m$ until it finds two pairs $(\bar{t}_1, \bar{t}_2)$ and $(\bar{t}'_1, \bar{t}'_2)$ such that $\mathsf{H}(\bar{t}_1, \bar{t}_2, PK, m) = \mathsf{H}(\bar{t}'_1, \bar{t}'_2, PK, m')$, which for an ℓ-bit hash function is expected to happen after $O(2^{\ell/2})$ tries. It then sends $(\bar{t}_1, \bar{t}_2, PK)$ to the honest signer, who responds with $s_1, \gamma_{1,1}, \gamma_{1,2}$. If $\alpha_{2,1}, \alpha_{2,2}, r_2, \alpha'_{2,1}, \alpha'_{2,2}, r'_2$ are the random values that $\mathcal{F}$ used to generate $(\bar{t}_1, \bar{t}_2)$ and $(\bar{t}'_1, \bar{t}'_2)$, then $\mathcal{F}$ outputs $(\bar{t}'_1, \bar{t}'_2, s' = s - r_2 + r'_2, \gamma'_1 = \gamma_1 + -\alpha_{2,1} + \alpha'_{2,1}, \gamma'_2 = \gamma_2 - \alpha_{2,2} + \alpha'_{2,2})$ as a valid forgery on m'.

6 Restricted Security of CoSi

In the previous sections, we showed that CoSi cannot be proved secure as a multi-signature scheme and presented actual sub-exponential attacks. Both the impossibility proof and the attacks, however, crucially rely on the adversary being able to engage in many concurrent queries with the signing oracle. So a natural question to ask is whether CoSi can be proved secure against an adversary that is limited to sequential attacks.

Below, we answer this question in the positive, but with some important caveats. We show that, in the knowledge of secret key (KOSK) model, CoSi can be proved secure against sequential attacks (or even logarithmically many concurrent attacks) in the knowledge of secret key (KOSK) model under the one-more discrete-logarithm assumption. The idea of the proof is that the reduction answers most of its signing queries using its discrete-logarithm oracle, but simulates the open queries at the moment of the fork by programming a simulated transcript into the random oracle. In order to do so, it guesses the indices of all open signing queries at the moment of the fork, as well as the indices of the random-oracle queries involved in those queries, resulting in a tightness loss of $q_{\mathrm{S}}^{q_{\mathrm{C}}}$, where q_{S} is the number of signing queries and q_{C} is the maximum number of concurrently open queries.

6.1 Security Proof

Theorem 4. CoSi *is a secure multi-signature scheme in the knowledge of secret key (KOSK) setting under the one-more discrete-logarithm assumption in the random-oracle model for logarithmic number of concurrent signing queries. More precisely,* CoSi *is* $(\tau, q_{\mathrm{S}}, q_{\mathrm{C}}, q_{\mathrm{H}}, \epsilon)$*-unforgeable in the KOSK setting in the random-oracle model if* $q > 8q_{\mathrm{H}}/\epsilon$ *and if the one-more discrete-logarithm problem is*

$$\left(k_{\max} \cdot \tau + k_{\max}(q_{\mathrm{S}} + 1) \cdot \tau_{\exp}) \ , \ \frac{\epsilon}{8q_{\mathrm{H}}q_{\mathrm{S}}^{q_{\mathrm{C}}}} - \frac{q_{\mathrm{S}}^2 k_{\max}^2}{q}\right)\text{-hard,}$$

where $k_{\max} = 8q_{\mathrm{H}}/\epsilon \cdot \ln(8n/\epsilon)$ *and* $\tau_{\exp}$ *is the time of a multi-exponentiation in* $\mathbb{G}$.

Proof. Since we analyze security in the KOSK setting, we consider CoSi without proofs of possession in the public keys. To prove the theorem, we assume for contradiction that a CoSi-forger $\mathcal{F}$ exists. Our reduction will simulate the CoSi unforgeability game towards $\mathcal{F}$, and when $\mathcal{F}$ successfully forges, we apply the forking lemma to extract the solution to the OMDL problem instance. To this end, we define algorithm $\mathcal{A}$ that simulates the CoSi unforgeability game towards $\mathcal{F}$ and is "compatible" with the forking lemma (Lemma 1). Having defined $\mathcal{A}$, we can define reduction $\mathcal{B}$, which will apply the forking lemma to $\mathcal{A}$ (meaning it will execute $\mathcal{GF}_{\mathcal{A}}$) to solve the OMDL problem.

Suppose we have a $(\tau, q_{\mathrm{S}}, q_{\mathrm{C}}, q_{\mathrm{H}}, \epsilon)$ forger $\mathcal{F}$ against the CoSi multi-signature scheme. Then consider an input generator IG that generates a random element $y \xleftarrow{\$} \mathbb{G}$ and an algorithm $\mathcal{A}$ that has access to a discrete-logarithm oracle $\mathcal{O}^{\mathtt{dlog}}$ and a challenge oracle $\mathcal{O}^{\mathtt{chall}}$ that, when queried on a previously unseen input h, chooses a random element $y \xleftarrow{\$} \mathbb{G}$ and returns it as the output of $\mathcal{O}^{\mathtt{chall}}(h)$. When queried again on $\mathcal{O}^{\mathtt{chall}}(h)$, it returns the same output as before. On input (y_0) and randomness $f = (\rho, h_1, \ldots, h_{q_{\mathrm{H}}})$, and given access to oracles $\mathcal{O}^{\mathtt{dlog}}$ and $\mathcal{O}^{\mathtt{chall}}$, $\mathcal{A}$ proceeds as follows.

For every index $k \in \{1, \ldots, q_{\mathrm{S}}\}$, $\mathcal{A}$ tosses a biased coin $coin_k \xleftarrow{\$} \{0, 1\}$ that turns out 1 with probability $\delta = (N+1)q_{\mathrm{C}}/q_{\mathrm{S}}$. If $coin_k = 1$, then $\mathcal{A}$ additionally guesses an index $\hat{q}_k \xleftarrow{\$} \{1, \ldots, q_{\mathrm{H}}\}$ and replaces the value of $h_{\hat{q}_k}$ in its input with a random element of $\mathbb{Z}_q$, chosen uniformly based on $\mathcal{A}$'s random tape.

Algorithm $\mathcal{A}$ then runs the forger $\mathcal{F}$ on input $pk^* = y_0$ with random tape ρ. If $\mathcal{F}$'s k-th oracle query is a random-oracle query $\mathsf{H}(x)$, then it returns h_k. (We assume without loss of generality that $\mathcal{F}$ does not make any duplicate queries.) If $\mathcal{F}$'s k-th oracle query is a signing query for a message m_k and a tree where the challenge signer is in position i with children $\mathcal{C}_i$, then it proceeds as follows.

If $coin_k = 0$, then $\mathcal{A}$ uses its $\mathcal{O}^{\mathtt{dlog}}$ oracle to answer the query while embedding a challenge point from $y_1, \ldots, y_n$ that it obtains from its oracle $\mathcal{O}^{\mathtt{chall}}$. More precisely, it queries $t'_i \leftarrow \mathcal{O}^{\mathtt{chall}}(h_k)$ and computes the value t_i in its first round (t_i, PK_i) as $t_i \leftarrow t'_i \cdot \prod_{j \in \mathcal{C}} t_j$. When $\mathcal{F}$ sends $(\bar{t}, PK)$, then $\mathcal{A}$ simulates an internal random-oracle query $c \leftarrow \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m_k)$ and makes a query to its discrete-logarithm oracle to obtain $s'_i \leftarrow \mathcal{O}^{\mathtt{dlog}}(t'_i \cdot pk^{*c})$. It then adds (t'_i, c, s'_i) to the initially empty set aux and returns its second round $s_i \leftarrow s'_i + \sum_{j \in \mathcal{C}_i} s_j \bmod q$.

If $coin_k = 1$, then $\mathcal{A}$ simulates the signing protocol by programming the random oracle at the index $\hat{q}_k$ for the index that it guessed to be the random-oracle query $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) = h_{\hat{q}_k}$ that will be used in this signing protocol. Namely, after receiving all values (t_j, PK_j) from its children $j \in \mathcal{C}_i$, $\mathcal{A}$ computes its first round (t_i, PK_i) by choosing $s'_i \xleftarrow{\$} \mathbb{Z}_q$ and computing $t_i \leftarrow g^{s'_i} pk^{*-h_{\hat{q}_k}} \cdot \prod_{j \in \mathcal{C}_i} t_j$. When the adversary sends $(\bar{t}, PK)$ and the challenge hash $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) \neq h_{\hat{q}_k}$, then $\mathcal{A}$ aborts by outputting $(\emptyset, \emptyset, aux)$. Otherwise, it waits for its children $j \in \mathcal{C}_i$ to send s_j and then outputs $s_i \leftarrow s'_i + \sum_{j \in \mathcal{C}_i} s_j$.

When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ outputs $(\emptyset, \emptyset, aux)$. Otherwise, let $\mathcal{F}$'s forgery be $\sigma = (c, s)$ on message m for a set of public keys $\mathcal{PK} = \{pk^*, pk_2, \ldots, pk_n\}$, which it outputs together with the secret keys $sk_2, \ldots, sk_n$. Let j_{f} be the index of $\mathcal{A}$'s random-oracle query $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) = c = h_{j_{\mathrm{f}}}$ where $\bar{t} = g^s PK^{-c}$ and $PK = \prod_{pk \in \mathcal{PK}} pk$. (Without loss of generality, we assume that $\mathcal{F}$ makes all hash queries involved in verifying the forgery.) If we let $s^* \leftarrow s - c \cdot \sum_{i=2}^{n} sk_i \bmod q$, then one can verify that $g^{s^*} = \bar{t} \cdot pk^{*c}$.

Let K be the set of "open" signing queries at the moment that $\mathcal{F}$ made its j_{f}-th random-oracle query, i.e., the set of indices k of $\mathcal{F}$'s signing queries where, at the moment of $\mathcal{F}$'s j_{f}-th random-oracle query, $\mathcal{A}$ had already output its first round (t_i, PK_i) but not yet its second round s_i. Since at most q_{C} signing queries can be open at any given time, we have that $|K| \leq q_{\mathrm{C}}$. If $coin_k = 0$ for some $k \in K$, then $\mathcal{A}$ aborts by outputting $(\emptyset, \emptyset, aux)$. Otherwise, it outputs $(\{j_{\mathrm{f}}\}, \{(\bar{t}, c, s^*)\}, aux)$.

Algorithm $\mathcal{A}$ runs in time less than $\tau + (q_{\mathrm{S}}+1)\tau_{\mathrm{exp}}$ and succeeds whenever $\mathcal{F}$ succeeds and $\mathcal{A}$ doesn't abort prematurely. This means that all of at most q_{C} open signing queries at the time of the j_{f}-th random-oracle query have $coin_k = 1$, which happens with probability $\delta^{q_{\mathrm{C}}}$, and that all of $\mathcal{A}$'s guesses $\hat{q}_k$ for all signing queries with $coin_k = 1$ were correct, which happens with probability $1/q_{\mathrm{H}}^{\delta \cdot q_{\mathrm{S}}}$. Setting $\delta = 1/q_{\mathrm{S}}$, we have that

the success probability of $\mathcal{A}$ is given by

$$\begin{aligned}\epsilon_{\mathcal{A}} &= \epsilon \cdot \frac{\delta^{q_{\mathrm{C}}}}{q_{\mathrm{H}}^{\delta \cdot q_{\mathrm{S}}}} \\ &= \frac{\epsilon}{q_{\mathrm{H}} q_{\mathrm{S}}^{q_{\mathrm{C}}}} \ .\end{aligned}$$

We prove the theorem by constructing an algorithm $\mathcal{B}$ that, on input random group elements $y_0, \ldots, y_n$ and given access to a discrete-logarithm oracle $\mathcal{O}^{\mathtt{dlog}}$ and a forger $\mathcal{F}$, solves the one-more discrete logarithm problem in $\mathbb{G}$. Namely, $\mathcal{B}$ runs the generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ from Lemma 1 on input y_0 with the algorithm $\mathcal{A}$ described above. It responds to $\mathcal{A}$'s $\mathcal{O}^{\mathtt{dlog}}$ oracle by relaying to and from its own $\mathcal{O}^{\mathtt{dlog}}$ oracle, except that when $\mathcal{A}$ makes a query $\mathcal{O}^{\mathtt{dlog}}(y)$ that it made before, then $\mathcal{B}$ returns the same response as before. It simulates the $\mathcal{O}^{\mathtt{chall}}(h)$ oracle by taking the next unused element from $y_1, \ldots, y_n$ on each new value h. If $\mathcal{GF}_{\mathcal{A}}$ outputs `fail`, then $\mathcal{B}$ also outputs `fail`. If $\mathcal{GF}_{\mathcal{A}}$ outputs $(\{j_{\mathrm{f}}\}, \{(\bar{t}, c, s^*)\}, \{(\bar{t}', c', s'^*)\}, Aux)$, then $\mathcal{B}$ proceeds as follows.

Algorithm $\mathcal{B}$ uses the output of $\mathcal{GF}_{\mathcal{A}}$ to solve the OMDL problem as follows. From the construction of $\mathcal{A}$ and $\mathcal{GF}_{\mathcal{A}}$, we know that all executions of $\mathcal{A}$ are identical up to the j_{f}-th random-oracle query, and therefore that $\bar{t} = \bar{t}'$. Since $c = h_{j_{\mathrm{f}}}$ and $c' = h'_{j_{\mathrm{f}}}$, we also have that $c \neq c'$. Given that $pk^* = y_0$, $g^{s^*} = \bar{t} \cdot pk^{*c}$, and $g^{s'^*} = \bar{t} \cdot pk^{*c'}$, we can compute the discrete logarithm of y_0 as $x_0 \leftarrow (s^* - s'^*)/(c - c') \bmod q$.

In each run, $\mathcal{A}$ makes at most q_{H} queries $\mathcal{O}^{\mathtt{chall}}(h_k)$ to its challenge oracle. If there are no collisions on any of the h_k values generated by $\mathcal{GF}_A$, then $\mathcal{O}^{\mathtt{chall}}$ always returns the same challenge points from $y_1, \ldots, y_n$ before the forking point at the j_{f}-th random-oracle query, and returns fresh challenge points after the fork. Because $\mathcal{B}$ doesn't repeat $\mathcal{O}^{\mathtt{dlog}}$ queries, and because each execution of $\mathcal{A}$ is identical up to the forking point, each of these challenge values y_i leads to at most one $\mathcal{O}^{\mathtt{dlog}}$ query, and when it does, it adds a tuple (t_i, c_i, s_i) to aux with $t_i = y_i$ and $g^{s_i} = t_i y_0^c$. Algorithm $\mathcal{B}$ can therefore compute the discrete logarithm x_i of y_i for each of these as $x_i \leftarrow s_i - x_0 \cdot c \bmod q$. For all unused challenge values y_i, it simply queries $x_i \leftarrow \mathcal{O}^{\mathtt{dlog}}(y_i)$, to that it can output $(x_0, \ldots, x_n)$ after a total of n queries to $\mathcal{O}^{\mathtt{dlog}}$.

The probability that $\mathcal{B}$ succeeds is the probability that $\mathcal{GF}_{\mathcal{A}}$ and no collisions occur among any of the q_{S} values h_i that $\mathcal{A}$ submits to its $\mathcal{O}^{\mathtt{chall}}$ oracle in any of its $k_{\max}$ runs. We therefore have that

$$\begin{aligned}\epsilon_{\mathcal{B}} &\geq \epsilon_{\mathcal{GF}_{\mathcal{A}}} - \frac{q_{\mathrm{S}}^2 k_{\max}^2}{q} \\ &\geq \frac{\epsilon_{\mathcal{A}}}{8} - \frac{q_{\mathrm{S}}^2 k_{\max}^2}{q} \\ &\geq \frac{\epsilon}{8 q_{\mathrm{H}} q_{\mathrm{S}}^{q_{\mathrm{C}}}} - \frac{q_{\mathrm{S}}^2 k_{\max}^2}{q} \ .\end{aligned}$$

Its running time of that of $\mathcal{GF}_{\mathcal{A}}$ plus $O(n)$ low-cost operations, so that the bounds in the theorem follow.

6.2 Security in Stronger Models

It is worth noting that the above proof does *not* extend to the key verification model with proofs of possession. The reason is that the reduction would in that case have to guess the open queries at *each* of the N forking points corresponding to the N proofs of possession for the public keys in the forgery, resulting in a tightness loss of $(q_{\mathrm{S}} q_{\mathrm{H}})^{N q_{\mathrm{C}}}$. This would mean that the scheme is secure for at most a handful of signers, i.e., barely useful as a multi-signature.

The proof technique does work for a polynomial number of signers in a restricted model where the adversary has to commit to all potential signers' public keys before being granted access the signing oracle. Alternatively, one could use a scheme in the plain public-key model (i.e., without requiring certificates) by using a different hash value as exponents for each public key in the verification equation [BN06], or by using a product of hash values as exponents [MWLD10,MPSW18a].

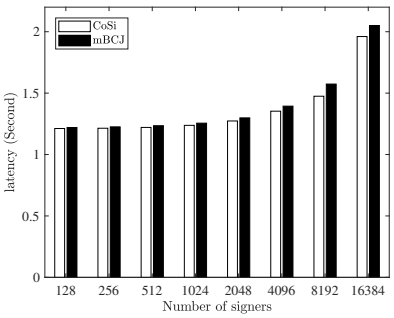

Fig. 4. Comparing end-to-end latency of CoSi and mBCJ signing with varying amounts of signers.

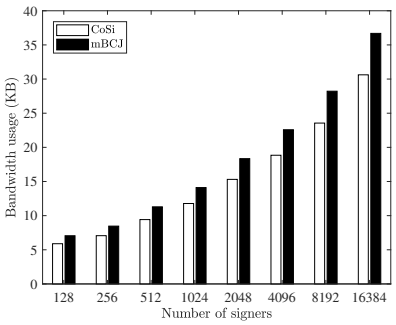

Fig. 5. Bandwidth consumption (sent and received combined) of CoSi and mBCJ with varying amounts of signers.

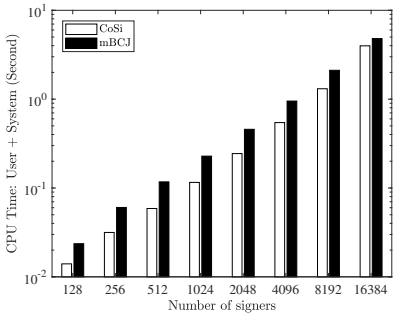

Fig. 6. CPU time (User + System) of CoSi and mBCJ with varying amounts of signers.

7 Evaluation

We presented mBCJ as an alternative to CoSi, where the provable security comes at the price of an increased signature size and a slightly increased computational cost. In this section, we evaluate the performance of mBCJ and show that the increased size and computational cost will not have a significant impact on the efficiency of the system.

7.1 Experiment Setup

Prototype. We implemented mBCJ in the Go programming language as an extension to the Collective Authority project (Cothority). We used the Cothority [Ded18b] and Onet [Ded18a] libraries to provide support for the tree-based collective signing as used in all three CoSi and mBCJ. This experiment compares the latest version of CoSi to an implementation of mBCJ. Note that CoSi has been further developed since its original publication [STV+16], which explains the small differences in performance measurements between their work and our results.

Physical configuration. A DeterLab [Det18] testbed was used to evaluate our system. The testbed consists of 28 physical machines, each containing an Intel Xeon E3-1260L processor and 16GB of RAM. Every physical machine simulated up to 586 signers for a total of 16,384 signers. A round-trip delay of 200 milliseconds between the machines is enforced to simulate an international connection, and all the signers that communicate with each other are deployed to different physical machines to correctly simulate the network delay.

Tree Configuration. mBCJ requires a tree structure between the different signers. For a given amount of signers, we can choose either a tree with a lower depth but a higher branching factor, or accept a higher depth but a lower branching factor. The overall network delay is linear in the depth of the tree, and the computation cost and network usage in each node scale linearly in the branching factor. We find that a depth of 3 (excluding the root of the tree) yields low network delays while keeping the computation cost and network usage manageable, and therefore use this setting for our following experiments, choosing a branching factor according to the number of signers.

Experiment. We simulate the signing process of CoSi and mBCJ to evaluate the system. In each experiment, the leader initiates the signing protocol for an arbitrary message, and the resulting signature is verified against the aggregate public key. Every experiment is repeated 10 times, taking the average of the individual runs.

7.2 Results

Signing Latency. To evaluate the scalability of mBCJ, we measured the end-to-end latency of the signing process, meaning the time between the moment that the root initiates the signing protocol and that it outputs the signature, from 128 up to 16,384 signers. Fig. 4 depicts the results, showing that mBCJ can easily scale to 16,384 signers, yielding a signature in about 2 seconds. It can readily be seen that the network delay dominates the overall latency, as the 1.2 seconds is exactly two rounds of three round trips over the depth of the tree. The results confirm our prediction that mBCJ scales as well as CoSi does, only marginally increasing the overall latency compared to them.

Bandwidth. Our second experiment measures the amount of data that every signer sends and receives. While leaf-signers (signers without children in the tree) send and receive less data, we here look at the data sent and received by the root signer, who always has the maximum amount of children. Fig. 5 shows that the bandwidth consumption of CoSi[10] and mBCJ are independent of the number of signers. mBCJ uses two points $t_{i,1}$ and $t_{i,2}$ in the commitment instead of one and it has three values s, γ_1 and γ_2 in the response which creates the additional cost with respect to CoSi. One may expect that these changes result in a more significant difference in bandwidth usage, however, the overhead of the connection and communicating the tree structure reduced the gap between the schemes. We believe a 20% increase in the bandwidth is a very acceptable overhead to gain provable security and will not hinder the system's scalability.

Computation Cost. Our final experiment compares the computational cost between CoSi and mBCJ, by measuring the total CPU time used to run all the signers (that is, the total time should be divided by the number of signers to obtain the average time spent per signer). We gathered both user time and system time of running processes to compute the CPU time, Fig. 6 shows the results. We observe a 75% average increase from CoSi to mBCJ. The major factor in CoSi time increase are the two multi-exponentiations required in mBCJ to compute $t_{i,1}$ and $t_{i,2}$.

Overall, mBCJ is still extremely efficient, as even with 16,384 signers, the average CPU time per signer is under 0.29 milliseconds.

8 Conclusion

Our work provides evidence that all currently known two-round Schnorr-based multi-signature schemes (BCJ, MWLD, CoSi, and MuSig) are insecure. On the one hand, we show that they cannot be proved asymptotically secure under standard assumptions, and on the other hand, we give sub-exponential attacks that can actually be mounted in practice for reasonable parameter choices. We then propose the mBCJ scheme as a provably secure yet highly efficient alternative. Compared to the original CoSi scheme, our experiments yield a 73% increase in CPU time and no noticeable difference in signing latency, showing that mBCJ is just as scalable as CoSi or any of the other schemes and is a viable alternative for use in large-scale decentralized systems. Alternative secure multi-signature schemes include non-interactive schemes based on pairings [BLS01,Bol03,BDN18] or three-move schemes based on discrete logarithms without pairings [BN06,BDN18,MPSW18b]. The construction of quantum-safe multi-signatures or efficient multi-signatures in the plain public-key model without random oracles are interesting open problems.

Our results demonstrate the usefulness as well as the limitations of provable security. Schemes without security proofs clearly have no place in modern cryptographic design, especially if efficient provably secure alternatives exist. Apparent resistance against obvious attacks says nothing about the security of a scheme because, as the sub-exponential attacks in this paper have shown, subtler attacks may always be hiding beneath the surface.

[10] We observed a tenfold improvement in the bandwidth of the current version of CoSi over the original one. After further investigation, we found out that the original CoSi aggregated the bandwidth cost over the ten rounds instead of the average.

But a security proof does not provide an absolute guarantee either. Security proofs can contain subtle flaws that, as some of the flaws that we pointed out in this work, may go unnoticed for years. Rewinding arguments, e.g., using the forking lemma, are particularly delicate, but are often brushed aside because of the complexity of the analysis. On the one hand, our results show the importance of writing out detailed proofs and carefully reviewing these proofs, but on the other hand, they also show the need for better tools, manual or automated, to create and verify proofs. Provable security may not be a perfect tool, but it's certainly the best tool that we have at the moment.

Acknowledgements

We would like to thank Stephen Schwab and the entire DeterLab team for their support for our experiments. We would also like to thank the anonymous referee for pointing out a flaw in an earlier version of this paper that helped us to significantly improve our results. Drijvers and Neven were supported by the EU under CHIST-ERA project USE-IT.

References

Bar04. Boaz Barak. *Non-Black-Box Techniques in Cryptography*. PhD thesis, 2004.

BCJ08. Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 449–458. ACM Press, October 2008.

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

bit17. Technology roadmap – Schnorr signatures and signature aggregation. `https://bitcoincore.org/en/2017/03/23/schnorr-signature-aggregation`, 2017.

BJ08. Ali Bagherzandi and Stanislaw Jarecki. Multisignatures using proofs of secret key possession, as secure as the Diffie-Hellman problem. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 218–235. Springer, Heidelberg, September 2008.

BKJ+17. Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Proof-of-personhood: Redemocratizing permissionless cryptocurrencies. In *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*, pages 23–26. IEEE, 2017.

BL13. Foteini Baldimtsi and Anna Lysyanskaya. On the security of one-witness blind signature schemes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 82–99. Springer, Heidelberg, December 2013.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 390–399. ACM Press, October / November 2006.

BNPS03. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

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

BP02. Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, August 2002.

BV98. Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg, May / June 1998.

Ded18a. Dedis. Cothority overlay network library. `https://github.com/dedis/onet`, 2018. [Online; accessed February-2018].

Ded18b. Dedis. Scalable collective authority prototype. `https://github.com/dedis/cothority`, 2018. [Online; accessed February-2018].

DEFN18a. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, and Gregory Neven. Okamoto beats Schnorr: On the provable security of multi-signatures. Cryptology ePrint Archive, Report 2018/417, Version 20180510:204458, 2018. Earlier version of this work.

DEFN18b. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, and Gregory Neven. On the provable security of two-round multi-signatures. Cryptology ePrint Archive, Report 2018/417, Version 20180809:082623, 2018. Earlier version of this work.

Det18. DeterLab: Cyber-Defense Technology Experimental Research Laboratory. `https://www.isi.deterlab.net`, 2018. [Online; accessed February-2018].

IN83. K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71:1–8, 1983.

KJG+16. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 279–296. USENIX Association, 2016.

KKJG+18. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy*, pages 583–598. IEEE Computer Society Press, May 2018.

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

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 01*, pages 245–254. ACM Press, November 2001.

MPSW18a. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068, Version 20180118:124757, 2018.

MPSW18b. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068, Version 20180520:191909, 2018.

MS09. Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *20th SODA*, pages 586–595. ACM-SIAM, January 2009.

MWLD10. Changshe Ma, Jian Weng, Yingjiu Li, and Robert H. Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Des. Codes Cryptography*, 54(2):121–133, 2010.

Nak08. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

NKJ+17. Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: proactive software-update transparency via collectively signed skipchains and verified builds. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 1271–1287. USENIX Association, 2017.

NS15. Ivica Nikolic and Yu Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 683–703. Springer, Heidelberg, November / December 2015.

NSW09. Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *J. Mathematical Cryptology*, 3(1):69–87, 2009.

Oka93. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993.

OO93. Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 139–148. Springer, Heidelberg, November 1993.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

PV05. Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2005.

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

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

Sch01. Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Heidelberg, November 2001.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

SJKK+17. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy*, pages 444–460. IEEE Computer Society Press, May 2017.

STV+16. Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities"honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy*, pages 526–545. IEEE Computer Society Press, May 2016.

Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

Wat05. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.