Okamoto Beats Schnorr: On the Provable Security of Multi-Signatures

Manu Drijvers
IBM Research – Zurich and ETH Zurich

Kasra Edalatnejad
EPFL

Bryan Ford
EPFL

Gregory Neven
IBM Research – Zurich

ABSTRACT

A multisignature 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 two highly efficient Schnorr-based multisignature schemes designed to scale up to thousands of signers, namely CoSi by Syta et al. (S&P 2016) and MuSig by Maxwell et al. (ePrint 2018). The MuSig scheme was presented with a proof under the one-more discrete-logarithm assumption, while the provable security of CoSi has so far remained an open question. In this work, we prove that CoSi and MuSig cannot be proved secure without radically departing from currently known techniques (and point out a flaw in the proof of MuSig). We then present DG-CoSi, a double-generator variant of CoSi based on the Okamoto (multi)signature scheme, and prove it secure under the discrete-logarithm assumption in the random-oracle model. Our experiments show that the second generator in DG-CoSi barely affects scalability compared to CoSi, allowing 8192 signers to collaboratively sign a message in under 1.5 seconds, making it a highly practical and provably secure alternative for large-scale deployments.

1 INTRODUCTION

A multisignature 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 multisignature schemes have been studied since decades [3, 8, 10, 16, 19, 20, 22, 27], they have recently received renewed interest because of the rise of distributed applications that aim to decentralize trust such as Bitcoin [24] and more generally blockchain. Such applications typically involve many users or nodes that need to approve particular actions, which naturally matches the multisignature setting where many signers must collaborate in order to create a joint multisignature.

Motivated by such applications, Syta et al. [35] presented the CoSi multisignature scheme, a highly scalable multisignature 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 [34], improving the scalability of Bitcoin [34], and introducing a decentralized software update framework [26]. CoSi is also considered for standardization by the IETF [15].

More recently, the Bitcoin community is actively looking into integrating Schnorr signatures as these could support multisignatures 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 [1]. To this end, a number of Bitcoin core developers published the MuSig scheme [21] 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. This is unfortunate, because the interactive nature of multisignature schemes tends to make their security proofs considerably more delicate than for standard signatures, as the results in this paper confirm.

Negative results. Our first result essentially shows that the CoSi and MuSig schemes cannot be proved secure. (This obviously contradicts the security proof of MuSig [21], but we point out that the proof is flawed.) More precisely, we prove that if the OMDL problem is hard, then there cannot exist an algebraic black-box reduction that proves CoSi or MuSig secure under the DL or OMDL assumption. 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 (so-called *forking* [30]). Also, given that CoSi and MuSig are derived from Schnorr signatures [32], it would be very surprising if its security could be proved under an assumption that is not implied by DL or OMDL (and that doesn't trivially assume the security of the scheme). 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 is extremely unlikely, as it would have to depart radically from any currently known techniques.

Intuitively, the problem of proving CoSi and MuSig secure is common among Schnorr-based multisignature schemes [3, 8, 22]. Namely, 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 needs to be programmed depends on the output of the other, possibly adversarial signers. Previous schemes [3, 8] therefore make signers commit to their output, so that the reduction can extract the commitment and program the random oracle. The CoSi and MuSig schemes do not have such a commitment step, however. Correctly simulating signing queries becomes especially difficult when used in combination with a forking argument, because the forger may be forked at a point where it has an "open" signing query. In that case, the reduction has to come up with a second response for the same first round of the signing protocol, which implies that it must already know the signing key that it was hoping to extract from the forger. The actual impossibility proof is a bit more involved, but it exploits this exact difficulty in simulating signing queries.

Scheme	KVf	KAg	Sign	Vf	Rounds	*pk* size	Signature size	*PK* size	Security proof
BN [8]			$1\mathbb{G}$	$1\mathbb{G}^{n+1}$	3	$\mathbb{G}$	$\mathbb{G} \times \mathbb{Z}_q$	$\mathbb{G}^n$	DL, ROM
B-Pop [10, 31]	$2P$		$1\mathbb{G}_1$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$	co-CDH, ROM
WM-Pop [19, 31]	$2P$		$1\mathbb{G}_1 + 1\mathbb{G}_2$	$2P$	1	$\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_T$	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_T$	co-CDH
BCJ1 [3]	$1\mathbb{G}^2$		$1\mathbb{G} + 2\mathbb{G}^2$	$3\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{G}^2 \times \mathbb{Z}_q^4$	$\mathbb{G}$	DL, ROM
BCJ2 [3]			$1\mathbb{G} + 2\mathbb{G}^2$	$1\mathbb{G}^{n+1} + 2\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{G}^3 \times \mathbb{Z}_q^3$	$\mathbb{G}$	DL, ROM
MWLD [20]			$1\mathbb{G}^2$	$1\mathbb{G}^{n+2}$	2	$\mathbb{G}$	$\mathbb{Z}_q^3$	$\mathbb{G}^n$	DL, ROM
CoSi [35]	$1\mathbb{G}^2$		$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G} \times \mathbb{Z}_q^2$	$\mathbb{Z}_q^2$	$\mathbb{G}$	N/A (Thm 3.1)
MuSig [21]		$1\mathbb{G}^n$	$1\mathbb{G}$	$1\mathbb{G}^2$	2	$\mathbb{G}$	$\mathbb{Z}_q^2$	$\mathbb{G}$	N/A (Thm 3.1)
DG-CoSi (this work)	$1\mathbb{G}^3$		$1\mathbb{G}^2$	$1\mathbb{G}^3$	2	$\mathbb{G} \times \mathbb{Z}_q^3$	$\mathbb{Z}_q^3$	$\mathbb{G}$	DL, ROM

Table 1: Efficiency of multisignatures in the key verification model. 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 $\mathbb{G}$, "$\mathbb{G}^n$" denotes an n-multiexponentiation in group $\mathbb{G}$ where n is the number of signers, and "P" denotes a pairing operation. 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. Column 10 shows the assumptions under which the scheme is proved secure, if any, where "ROM" indicates a proof in the random-oracle model.

Positive results. The main reason to deviate from existing provably secure schemes is that they are less efficient than CoSi and MuSig in terms of the number of rounds in the signing protocol [8], in terms of signature size [3], or in terms of signature verification [3, 8, 21]. We therefore present the DG-CoSi multisignature scheme, a double-generator variant of CoSi that is based on Okamoto signatures [28] instead of Schnorr signatures. As already observed by Ma et al. [20], the witness indistinguishability of Okamoto signatures solves the problem of simulating the honest signer, because the simulator can use one witness to simulate the signer and hope to extract a second witness from the forger. We thereby provide a rigorous security proof of the DG-CoSi scheme under the DL assumption.

Table 1 gives an overview of the efficiency of existing multisignature schemes and DG-CoSi. For large numbers of signers, it is crucial that a constant-size aggregate public key can be computed from the set of individual public keys, and that the signature size and the cost of signature verification is independent of the number of signers. This will allow a verifier to compute the aggregate public key once, and have (amortized) constant-time verification of multisignatures.

Prior to DG-CoSi, only three provably secure schemes offered these features. The scheme due to Bagherzandi et al. [3] has a signature size that is roughly double that of DG-CoSi, while the B-Pop [10, 31] and WM-Pop [19, 31] schemes have more expensive verification due to the use of pairings.

One may wonder what price one pays for the provable security of the double-generator DG-CoSi scheme, when compared to the highly efficient single-generator 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 roundtrip delay of 200 miliseconds, we found that a group as large as 8192 signers can collaboratively sign a message using DG-CoSi in less than 1.5 seconds, showing no significant difference with CoSi. In terms of computational efficiency, DG-CoSi on average needs 32% more CPU time than CoSi, but still requires far below 1 millisecond of CPU time per signer when signing with 8192 signers.

Our results show that DG-CoSi is only marginally less efficient than CoSi, so that any protocol based on the unprovable CoSi scheme could instead be built on the provably secure DG-CoSi scheme.

2 PRELIMINARIES

2.1 Discrete Logarithm Problems

Definition 2.1 (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

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

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 2.2 (n-One-more Discrete Log Problem [7, 9]). For a group $\mathbb{G} = \langle g \rangle$ of prime order q, let $\mathcal{O}^{\mathsf{dlog}}(\cdot)$ be a discrete logarithm oracle that can be called at most n times. We define $\mathsf{Adv}^{\mathsf{n-omdl}}_{\mathbb{G}}$ of an adversary $\mathcal{A}$ as

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

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 $\mathsf{Adv}^{\mathsf{n-omdl}}_{\mathbb{G}} \geq \epsilon$. n-one-more discrete log is (τ, ϵ)-hard if no such adversary exists.

2.2 Algebraic Algorithms

Boneh and Venkatesan [11] 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 [29]:

Definition 2.3. 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 [30] to analyze the security of Schnorr signatures [32]. The lemma rewinds a forger $\mathcal{A}$ against the Schnorr signature scheme in the random-oracle model 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 [8] 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 [3] generalized the lemma even further to multiple subsequent rewindings on multiple crucial queries. It is the latter generalization of the forking lemma that we recall here.

Let $\mathcal{A}$ be an algorithm that on input *in* interacts with a random oracle $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$. Let $f = (\rho, h_1, \ldots, h_{q_\mathrm{H}})$ be the randomness involved in an execution of $\mathcal{A}$, where ρ is $\mathcal{A}$'s random tape, h_i is the response to $\mathcal{A}$'s i-th query to H, and q_H is its maximal number of random-oracle queries. Let Ω be the space of all such vectors f and let $f|_i = (\rho, h_1, \ldots, h_{i-1})$. We consider an execution of $\mathcal{A}$ on input *in* and randomness f, denoted $\mathcal{A}(in, f)$, as *successful* if it outputs a pair $(J, \{out_j\}_{j \in J})$, where J is a multi-set that is a non-empty subset of $\{1, \ldots, q_\mathrm{H}\}$ and $\{out_j\}_{j \in J}$ is a multi-set of side 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:

$\mathcal{GF}_\mathcal{A}(in)$:
- $f = (\rho, h_1, \ldots, h_{q_\mathrm{H}}) \xleftarrow{\$} \Omega$
- $(J, \{out_j\}_{j \in J}) \leftarrow \mathcal{A}(in, f)$
- If $J = \emptyset$ then output `fail`
- 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 $f'' = (\rho, h_1, \ldots, h_{j_i - 1}, h''_{j_i}, \ldots, h''_{q_\mathrm{H}})$
    - $(J'', \{out''_j\}_{j \in J''}) \leftarrow \mathcal{A}(in, f'')$
    - If $h''_{j_i} \neq h_{j_i}$ and $J'' \neq \emptyset$ and $j_i \in J''$ then
      - $out'_{j_i} \leftarrow out''_{j_i}$; $succ_i \leftarrow 1$
- If $succ_i = 1$ for all $i = 1, \ldots, n$
- Then output $(J, \{out_j\}_{j \in J}, \{out'_j\}_{j \in J})$ else output `fail`

We say that $\mathcal{GF}_\mathcal{A}$ succeeds if it doesn't output `fail`. Bagherzandi et al. proved the following lemma for this forking algorithm.

Lemma 2.4 (Generalized Forking Lemma [3]). *Let* IG *be a randomized algorithm and $\mathcal{A}$ be a randomized algorithm running in time τ making at most q_H random-oracle queries 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 $\xleftarrow{\$}$ IG and over the coins of $\mathcal{GF}_\mathcal{A}$.*

2.4 Security of Multisignatures

We follow the syntax and security model due to Bagherzandi et al. [3], which follows the so-called key-verification model, as introduced by Bagherzandi and Jarecki [4], 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 multisignature schemes that do not have this feature: indeed, such schemes can simply use the identity function as key aggregation algorithm.

A multisignature 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 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 multisignature 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 multisignature scheme should satisfy unforgeability. Unforgeability of a multisignature 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 multisignature 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 $pk \neq pk^*$, $PK \leftarrow \mathsf{KAg}(\mathcal{PK})$, $\mathsf{Vf}(\mathcal{PK}, \sigma, m) = 1$, 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.[1] 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 2.5. We say $\mathcal{A}$ is a $(\tau, q_S, q_H, \epsilon)$-forger for multisignature scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ if it runs in time τ, makes q_S signing queries, makes q_H random oracle queries, and wins the above game with probability at least ϵ. MS is $(\tau, q_S, q_H, \epsilon)$-unforgeable if no $(\tau, q_S, q_H, \epsilon)$-forger exists.

3 SECURITY OF THE CoSi MULTISIGNATURE SCHEME

CoSi is a multisignature scheme introduced by Syta et al. [35] that follows a long line of work on Schnorr-based multisignatures [3, 8, 20, 22, 31]. It improves the efficiency of prior work: it is a two round protocol, verification of a signature is as efficient as verifying a single Schnorr signature, and due to employing a tree structure to compute the signature, thousands of signers can create a multisignature in seconds, as demonstrated by their open source implementation[2]. Since its recent introduction, CoSi has already led to a large body of follow-up work [12, 17, 18, 26, 34] and is considered for standardization by the IETF [15]. However, the security of CoSi is not formally analyzed, as Syta et al. [35] do not formally prove security, leaving the open question: is CoSi provably secure? In this section, we will show that it is very unlikely that CoSi can be proven secure, by proving that it is impossible to prove security even under the strong OMDL assumption.

[1] The distinction between the key-verification model and plain public key model is a bit informal, as they are in fact equivalent: any multisignature 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.

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

3.1 Description of CoSi

3.1.1 Parameters generation. Pg sets up a group $\mathbb{G} = \langle g \rangle$ of order q, where q is a κ-bit prime. Output $par \leftarrow (\mathbb{G}, g, q)$.

3.1.2 Key generation. The key generation algorithm $\mathsf{Kg}(par)$ takes $sk \xleftarrow{\$} \mathbb{Z}_q$ and sets $pk \leftarrow g^{sk}$. To prevent related-key attacks [23], the authors suggest that users prove knowledge of their secret key. We will omit those proofs here and study CoSi in the KOSK setting in which such attacks cannot occur.

3.1.3 Signing. Signing is the four-step protocol. A signer S_i on input $\mathsf{Sign}(par, (x_i, pk_i), m, \mathcal{T})$ 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 $PK_i \leftarrow pk_i \cdot \prod_{j \in \mathcal{C}_i} PK_j$. 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 $c \leftarrow \mathsf{H}(\bar{t}, m)$, and sends $\bar{t}$ to its children. If S_i is not the leader, it waits to receive a message $\bar{t}$, computes $c \leftarrow \mathsf{H}(\bar{t}, m)$, and sends $\bar{t}$ to its children[3].

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 $\sigma \leftarrow (c, s)$.

3.1.4 Key Aggregation. KAg on input a set of public keys $\mathcal{PK}$ outputs aggregate public key $PK \leftarrow \prod_{pk \in \mathcal{PK}} pk$.

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

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

3.2 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 random-oracle model under the OMDL assumption.

In the classical security proof of Schnorr signatures under the DL assumption [30], 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

[3] We make a small variation of the original CoSi scheme [35], in which the cosigners receive c from the leader instead of $\bar{t}$. Without knowing $\bar{t}$, cosigners cannot verify that $c = \mathsf{H}(\bar{t}, m)$ and are unable to check that they are signing the message they intend to sign, and one cannot hope to prove unforgeability in this setting.

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 [9, 21], 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. [21] 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 [9] 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 [5] 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 multisignature 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 [5]. 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 3.1. *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 $((2\ell+1)\tau_{\text{exp}} + O(\ell), \ell, 3, 1 - 1/q)$-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*, 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's 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's set $\ell = 250$, and let's say that we have a forger that breaks CoSi with overwhelming probability using just 500 exponentiations, 250 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. Nevertheless, even for such a strong forger, Theorem 3.1 says that there cannot exist a reduction $\mathcal{B}$ that uses this forger to obtain just an ϵ advantage in breaking the n-OMDL problem for any $n \geq 0$. More specifically, it says that if such a reduction would exist, then that reduction would immediately give rise to a solution for the $(n+k)$-OMDL problem *without* needing access to any forger,

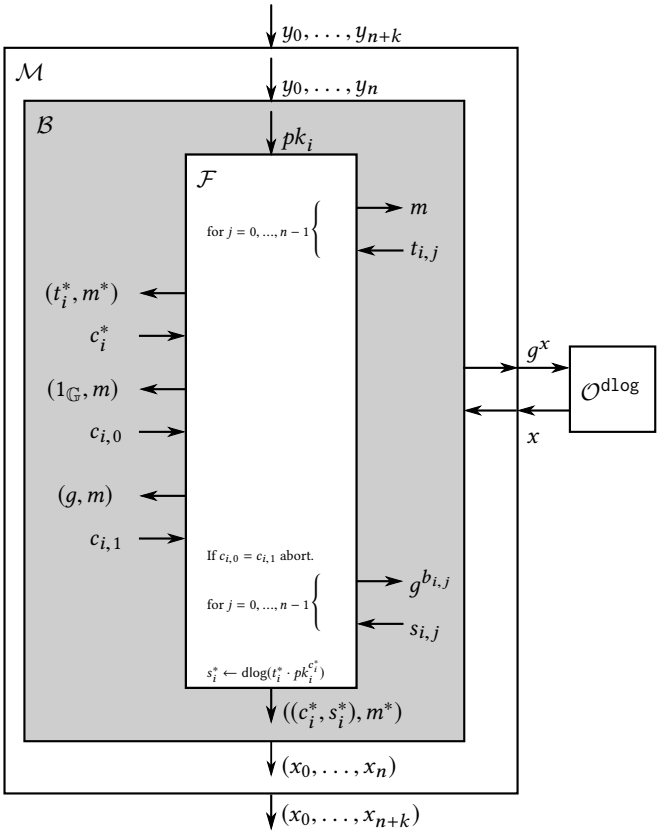

Figure 1: Our metareduction $\mathcal{M}$ in the proof of Theorem 3.1, which simulates forger $\mathcal{F}$ towards any reduction $\mathcal{B}$ that would prove the security of CoSi **under the OMDL assumption, and uses $\mathcal{B}$ to break the OMDL problem.**

essentially meaning that the OMDL assumption was false to begin with.

The only room left by Theorem 3.1 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 any of the other ones. 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 [6], 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 [33]. However, this would mean that much stronger assumptions are required than one would expect from a Schnorr-based protocol.

Proof of Theorem 3.1. 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 1 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 $\mathcal{O}^{\mathsf{dlog}}$ queries using its own $\mathcal{O}^{\mathsf{dlog}}$ 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 dlog 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 dlog, finally, returns the discrete logarithm of its argument; 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 $j = 1, \ldots, \ell$.
- 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 message m^* with public keys $\mathcal{PK} = \{pk_i\}$ by computing $s_i^* \leftarrow \mathsf{dlog}(\bar{t}_i^* \cdot pk_i^{c_i^*})$.

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 $(2\ell+1)\tau_{\mathrm{exp}} + O(\ell)$. It outputs a successful forgery unless $c_{i,0} = c_{i,1}$, which happens with probability $1/q$. Therefore, $\mathcal{F}$ is a $((2\ell+1)\tau_{\mathrm{exp}} + O(\ell), \ell, 3, 1 - 1/q)$-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 dlog subroutine as follows:

- If the i-th execution of $\mathcal{F}$ invokes the subroutine $\mathsf{dlog}(\bar{t}_i^* \cdot pk_i^{c_i^*})$ 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 discrete logarithm 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 $\mathsf{dlog}(\bar{t}_i^* \cdot pk_i^{c_i^*})$ 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 discrete logarithm 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 dlog subroutine in any execution $i \in C$, let P_1 contain $C \in P$ such that $\mathcal{F}$ invoked the dlog exactly once over all executions $i \in C$, and let P_{2+} contain the components $C \in P$ such that $\mathcal{F}$ invoked dlog 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 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 dlog, and extracted the value of sk_i during the second invocation of dlog. 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 $\tau + \tau_{\mathrm{ext}} + O(n + k\ell)$ 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 dlog 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 $\epsilon - k^2/2^\ell$. □

3.3 Applicability of metareduction to MuSig

While our metareduction is written for CoSi, the same technique can be applied to the similar multisignature scheme MuSig as recently introduced by Maxwell et al. [21]. 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 paralel, 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.

4 EFFICIENT MULTISIGNATURES IN THE KEY VERIFICATION MODEL

There are two main problems when designing provably secure Schnorr-type multisignatures. First, in the security proof, signing queries are usually simulated by using the zero-knowledge property to generate a valid transcript for a chosen challenge, and by programming the relevant random-oracle entry to return exactly that challenge. This approach works fine for standard signatures, because one argument of the random-oracle query (the so-called t-value) is generated fresh by the simulator, so that the probability that the relevant entry had already been queried by the adversary is negligible. For many efficient multisignature schemes, however, the final $\tilde{t}$-value is known first to the leader of or to the last contributor, which may not be the honest signer. The Bellare-Neven scheme [8] gets around this by letting signers first commit to their contribution, and only then open those commitments, thereby adding another round to the protocol. The Bagherzandi et al. scheme [3] uses homomorphic trapdoor commitments to eliminate the extra round, but at the cost of efficiency in terms of signature size and computational efficiency. The Ma et al. scheme [20] uses a double-generator technique due to Okamoto [28] that simulates signatures using a real signing key, rather than by programming the random oracle.

The other main problem to overcome is that, in the reduction, the signatures of co-signers typically need to be "divided out" of the forgery to obtain a solution to the underlying problem. This is usually easy in the KOSK model, where the adversary has to give up the secret keys of all cosigners involved in the forgery, but is more difficult in the plain public-key and key-verification models, where the adversary is allowed to use arbitrary keys, including rogue keys to which he doesn't know the secret key. Previous schemes in these models either required all signers to generate their keys jointly in a distributed protocol [22], which may not be particularly practical, or modified the verification equation to use different exponents for different public keys for each signature [3, 8, 20] or for each new group of signers [21], which may not be particularly efficient, e.g., for large or dynamic groups of signers, or when multi-exponentiations cannot be used.

Our DG-CoSi scheme is a double-generator variant of the CoSi scheme that overcomes both of the above problems. The protocol is largely due to Okamoto [28], who described a standard signature scheme that is a variant of Schnorr signatures where the secret key, rather than being the discrete logarithm of the public key y, is a representation (x_1, x_2) of y with respect to (g, h) such that $y = g^{x_1}h^{x_2}$. The extension to multisignatures was mentioned by Okamoto [28], but without details or proofs, and the same technique was used in Ma et al.'s multisignature scheme [20]. Rather than using the zero-knowledge property and programming the random oracle to simulate signatures, it relies on the witness indistinguishability property by simulating signatures using a real signing key (x_1, x_2). The simulator can therefore simply follow the honest signing protocol, avoiding the problem of having to program the random oracle on a free entry. The reduction then uses the forking lemma to extract a second representation (x_1', x_2') from the forger, from which with high probability it can derive the discrete logarithm of h with respect to g.

We solve the other problem of extracting the cosigners' secret keys by showing that the simple proofs of possession that Ristenpart and Yilek [31] proved for Boldyreva's [10] and Lu et al.'s [19] multisignature schemes also work for our scheme. Signers in our scheme merely have to include a self-signed certificate in their public keys, i.e., a simple signature on a dedicated certificate request. Verifiers have to check the certificates of new public keys, but can mark verified public keys as trusted and skip this check in subsequent verifications. Our scheme thereby supports true key aggregation, meaning that the resulting multisignature can be verified in the same time as a single signature under a public key that is simply the product of all cosigners' public keys.

We prove the security of our approach by using Bagherzandi et al.'s generalized forking lemma [3] to extract from the self-signed certificates the secret keys of all cosigners involved in the *first* forgery produced by the adversary. By including the aggregated public key in the signature hash, we know that the second forgery will use the same aggregated public key (even if possibly for a different set of cosigners), and this suffices to "divide out" the cosigners' contributions to the forgeries.

4.1 Description of DG-CoSi

4.1.1 Parameters generation. $\mathsf{Pg}(\kappa)$ sets up a group $\mathbb{G} = \langle g \rangle$ of order q, where q is a κ-bit prime. Take $h \xleftarrow{s} \mathbb{G}$ and output $par \leftarrow (\mathbb{G}, g, h, q)$.

4.1.2 Key generation. The key generation algorithm $\mathsf{Kg}(par)$ chooses $x_1, x_2 \xleftarrow{s} \mathbb{Z}_q$ and computes $y \leftarrow g^{x_1}h^{x_2}$. It then computes a standard Okamoto signature [28] $\pi = (d, w_1, w_2)$ using by choosing $u_1, u_2 \xleftarrow{s} \mathbb{Z}_q$ and computing $v \leftarrow g^{u_1}h^{u_2}$, $d \leftarrow \mathsf{H}(\texttt{cert}, y, v)$, $w_1 \leftarrow u_1 + dx_1 \bmod q$ and $w_2 \leftarrow u_2 + dx_2 \bmod q$. Here, `cert` is a string literal to perform domain separation on H, such that these individual "self-signed credentials" cannot be confused with multisignatures. It sets $sk \leftarrow (x_1, x_2, y)$ and $pk \leftarrow (y, \pi)$, and outputs (pk, sk).

4.1.3 Signing. On input $\mathsf{Sign}(par, (x_{i,1}, x_{i,2}, y_i), m, \mathcal{T})$, a signer S_i behaves as follows.

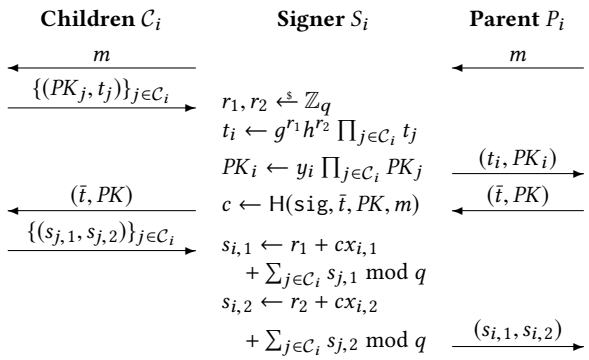

Figure 2: The DG-CoSi **signing protocol for signer S_i with secret key $sk_i = (x_{i,1}, x_{i,2})$ and public key $pk = (y_i, \pi_i)$. If S_i is the leader then, instead of sending (t_i, PK_i) to its parent, it sends $(\bar{t}, PK) = (t_i, PK_i)$ to its children, and instead of sending $(s_{i,1}, s_{i,2})$ to its parent, it outputs $(c, s_1, s_2) = (c, s_{i,1}, s_{i,2})$ as the signature.**

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 $r_{i,1}, r_{i,2} \xleftarrow{\$} \mathbb{Z}_q$ and computes $t_i \leftarrow g^{r_{i,1}}h^{r_{i,2}} \cdot \prod_{j\in\mathcal{C}_i} t_j$ 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 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 $c \leftarrow \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$, 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 $c \leftarrow \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$, and sends $(\bar{t}, PK)$ to its children. Here, sig denotes the string literal.

Response. S_i waits to receive all values $s_{j,b}$ for $j \in \mathcal{C}_i$ and $b \in \{1, 2\}$ (note that if S_i is a leaf it will proceed immediately), and then computes $s_{i,b} \leftarrow r_{i,b} + c \cdot x_{i,b} + \sum_{j\in C_i} s_{j,b}$ 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 $s_1 \leftarrow s_{i,1}$ and $s_2 \leftarrow s_{i,2}$ and outputs $\sigma \leftarrow (c, s_1, s_2)$.

4.1.4 *Key Aggregation.*

KAg on input a set of public keys $\mathcal{PK}$ checks the validity of the self-signed certificates included in the public keys by checking for all $pk = (y, (d, w_1, w_2)) \in \mathcal{PK}$ that $d \stackrel{?}{=} \mathsf{H}(\mathtt{cert}, y, g^{w_1}h^{w_2}y^{-d})$. If they are all valid, it outputs $PK \leftarrow \prod_{(y,\pi)\in\mathcal{PK}} y$.

4.1.5 *Verification.*

Vf on input an aggregate public key PK, a signature $\sigma = (c, s_1, s_2)$, and a message m, checks that

$$c \stackrel{?}{=} \mathsf{H}\left(\mathtt{sig}\ ,\ g^{s_1}h^{s_2} \cdot PK^{-c}\ ,\ PK\ ,\ m\right)\ .$$

It is worth noting that the message m is actually not required during the first round of the signing protocol, so it could alternatively be passed only later, together with the second-round message $(\bar{t}, PK)$. This allows some degree of pre-computation, as the computationally most intensive part of the protocol, the computation of t_i, can be performed before the message is known.

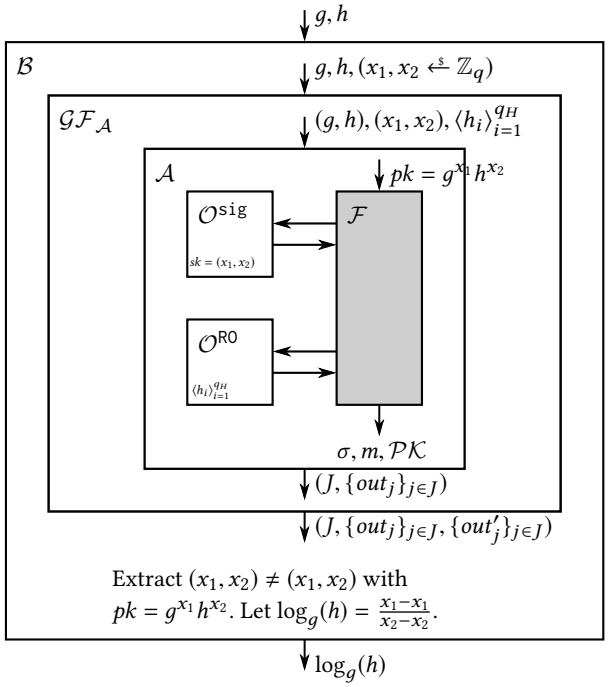

Figure 3: The construction of our reduction $\mathcal{B}$ breaking the DL problem given a DG-CoSi**-forger $\mathcal{F}$.**

Also, signers do not necessarily need to know the full tree structure $\mathcal{T}$: it suffices that they know how to reach their parent and their children. The tree may even be built dynamically by letting each node discover its responsive children during the first round of the signing protocol. All signers that participated in the first round by contributing a value t_i must also participate in the second round by contributing $(s_{i,1}, s_{i,2})$, however.

When verifying many signatures by (subsets of) the same group of signers, it is clear that the self-signed certificates π_i in the public keys $pk_i = (y_i, \pi_i)$ only need to be verified once, and that also the aggregate public key PK can be precomputed. Amortized over many signatures for the same or similar groups of signers, verification of a DG-CoSi signature therefore only takes a single multi-exponentiation in $\mathbb{G}$.

4.2 Security

Theorem 4.1. DG-CoSi *is a secure multisignature scheme in the key-verification setting under the discrete-logarithm assumption in the random-oracle model. More precisely,* DG-CoSi *is $(\tau, q_S, q_H, \epsilon)$-unforgeable in the random-oracle model if $q > 8Nq_H/\epsilon$ and if the discrete-logarithm problem is $((\tau + (q_S + N + 1)\tau_{\text{exp}}) \cdot 8N^2q_H/\epsilon \cdot \ln(8N/\epsilon), \epsilon/8 - 1/q)$-hard, where N is the maximum number of signers involved in a single multi-signature and τ_{exp} is the time of a multi-exponentiation in $\mathbb{G}$.*

Proof. To prove this theorem, we assume for contradiction a DG-CoSi-forger $\mathcal{F}$ exists. Our reduction will simulate the DG-CoSi

unforgeability game towards $\mathcal{F}$, and when $\mathcal{F}$ successfully forges, we apply the forking lemma to extract the solution to the DL problem instance. To this end, we define algorithm $\mathcal{A}$ that simulates the DG-CoSi unforgeability game towards $\mathcal{F}$ and is "compatible" with the forking lemma. That is, it outputs the signature and certificates from which our reduction needs to extract. Having defined $\mathcal{A}$, we can define reduction $\mathcal{B}$, which will apply the forking lemma on $\mathcal{A}$ (meaning it will execute $\mathcal{GF}_{\mathcal{A}}$) to solve the DL problem. The structure of this proof is depicted in Fig. 3.

Suppose we have a $(\tau, q_{\mathrm{S}}, q_{\mathrm{H}}, \epsilon)$ forger $\mathcal{F}$ against the DG-CoSi multisignature scheme. Then consider an input generator IG that generates random tuples $(h, x_1, x_2) \xleftarrow{\$} \mathbb{G} \times \mathbb{Z}_q \times \mathbb{Z}_q$ and an algorithm $\mathcal{A}$ that on input h, x_1, x_2 and randomness $f = (\rho, h_1, \ldots, h_{q_{\mathrm{S}}})$ proceeds as follows.

Algorithm $\mathcal{A}$ computes $y^* \leftarrow g^{x_1} h^{x_2}$ and simulates the corresponding self-signed certificate π^* by picking $(d^*, w_1^*, w_2^*) \xleftarrow{\$} \mathbb{Z}_q^3$ and computing $v^* \leftarrow g^{w_1^*} h^{w_2^*} y^{*-d^*}$. It then runs the forger $\mathcal{F}$ on input $pk^* = (y^*, \pi^*)$ with random tape ρ. It reponds to $\mathcal{F}$'s i-th random-oracle query with h_i, except when $\mathcal{F}$ makes a query $\mathsf{H}(\mathtt{cert}, y^*, v^*)$ it responds with d^*. (We assume without loss of generality that $\mathcal{F}$ does not make any duplicate queries.) It responds to $\mathcal{F}$'s signing queries by running the honest signing algorithm using $sk = (x_1, x_2)$.

When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ outputs `fail`. Otherwise, if $\mathcal{F}$'s forgery is $\sigma = (c, s_1, s_2)$ on message m for a set of public keys $\mathcal{PK}$, then let j_{f} be the index of $\mathcal{A}$'s random-oracle query $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) = c = h_j$ where $\bar{t} = g^{s_1} h^{s_2} PK^{-c}$ and $PK = \prod_{y \in \mathcal{PK}} y$, and let $out_{j_{\mathrm{f}}} = (\mathtt{sig}, \bar{t}, c, s_1, s_2, \mathcal{PK})$. (Also without loss of generality, we assume that $\mathcal{F}$ makes all hash queries involved in verifying the forgery.) For each $pk = (y, (d, w_1, w_2)) \in \mathcal{PK}^*$, where $\mathcal{PK}^* = \mathcal{PK} \setminus \{pk^*\}$, let j_y be the index of $\mathcal{A}$'s random-oracle query $\mathsf{H}(\mathtt{cert}, y, v)$ where $v = g^{w_1} h^{w_2} y^{-d}$, and let $out_{j_y} = (\mathtt{cert}, y, v, d, w_1, w_2)$. Algorithm $\mathcal{A}$ then outputs $(J = \{j_y\}_{(y,\pi) \in \mathcal{PK}^*} \cup \{j_{\mathrm{f}}\}, \{out_j\}_{j \in J})$. It runs in time less than $\tau + (q_{\mathrm{S}} + N + 1)\tau_{\exp}$ and succeeds with probability ϵ.

We prove the theorem by constructing an algorithm $\mathcal{B}$ that, on input a group element h and given a forger $\mathcal{F}$, solves the discrete logarithm problem in $\mathbb{G}$. Namely, $\mathcal{B}$ chooses $(x_1, x_2) \xleftarrow{\$} \mathbb{Z}_q^2$ and runs the generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ from Lemma 2.4 on input (h, x_1, x_2) 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.

Algorithm $\mathcal{B}$ then uses the output of $\mathcal{GF}_{\mathcal{A}}$ to extract a second pair (x'_1, x'_2) such that $g^{x'_1} h^{x'_1} = y^*$, as well as the secret keys of all signers involved in the first forgery $out_{j_{\mathrm{f}}}$. It does *not* extract the secret keys of the signers involved in the *second* forgery $out'_{j_{\mathrm{f}}}$, however, which may be a different set of signers. Those keys cannot be extracted with Lemma 2.4, as doing so would require rewinding $\mathcal{F}$ a number of times that is exponential in the number of signers involved in the forgery. We will see that we don't need to extract those keys, as long as the aggregated public keys PK and PK' are the same in both runs.

Let $out_{j_{\mathrm{f}}} = (\mathtt{sig}, \bar{t}, c, s_1, s_2, \mathcal{PK})$ and $out'_{j_{\mathrm{f}}} = (\mathtt{sig}, \bar{t}', c', s'_1, s'_2, \mathcal{PK}')$ be the two outputs of $\mathcal{A}$ related to the forgery. For every $(y, \pi) \in \mathcal{PK}^* = \mathcal{PK} \setminus \{pk^*\}$, there are two outputs $out_{j_y} = (\mathtt{cert}, y, v, d, w_1, w_2)$ and $out'_{j_y} = (\mathtt{cert}, y', v', d', w'_1, w'_2)$ such that $g^{w_1} h^{w_2} = v y^d$ and $g^{w'_1} h^{w'_2} = v' y'^{d'}$. From the construction of $\mathcal{GF}_{\mathcal{A}}$, we know that out_{j_y} and out'_{j_y} were obtained from two executions of $\mathcal{A}$ with randomness f and f' such that $f|_{j_y} = f'|_{j_y}$, meaning that these executions are identical up to the j_y-th random-oracle query. In particular, this means that the arguments of the j_y-th random-oracle query $\mathsf{H}(\mathtt{cert}, y, v)$ are equal in both runs, meaning that $y = y'$ and $v = v'$. Dividing the two verification equations yields

$$g^{w_1 - w'_1} h^{w_2 - w'_2} = y^{d - d'} ,$$

and since the construction of $\mathcal{GF}_{\mathcal{A}}$ guarantees that $d \neq d'$, we can extract a secret key $sk_y = (x_{y,1}, x_{y,2})$ for y as $x_{y,1} \leftarrow (w_1 - w'_1)/(d - d') \bmod q$ and $x_{y,2} \leftarrow (w_2 - w'_2)/(d - d') \bmod q$.

Algorithm $\mathcal{B}$ then extracts a second pair (x'_1, x'_2) such that $g^{x'_1} h^{x'_2} = y^*$ from the forgeries $(\mathtt{sig}, \bar{t}, c, s_1, s_2, \mathcal{PK})$ and $(\mathtt{sig}, \bar{t}', c', s'_1, s'_2, \mathcal{PK}')$ as follows. Since they are valid forgeries, they satisfy the verification equations $g^{s_1} h^{s_2} = \bar{t} \cdot PK^c$ and $g^{s'_1} h^{s'_2} = \bar{t}' \cdot PK'^{c'}$, where $PK = \prod_{(y,\pi) \in \mathcal{PK}} y$ and $PK' = \prod_{(y,\pi) \in \mathcal{PK}'} y$. We know that the two executions of $\mathcal{A}$ leading to those forgeries are identical up to the j_{f}-th random-oracle query, meaning that the arguments of these hash queries $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$ and $\mathsf{H}(\mathtt{sig}, \bar{t}', PK', m')$ must be identical. We therefore have that $\bar{t} = \bar{t}'$ and that, even though the set of signers $\mathcal{PK}$ and $\mathcal{PK}'$ may be different, their aggregated public keys $PK = PK'$ must be the same. Dividing the two signature verification equations therefore yields

$$g^{s_1 - s'_1} h^{s_2 - s'_2} = PK^{c - c'} = y^{*c - c'} \cdot \Big(\prod_{y \in \mathcal{PK}^*} g^{x_{y,1}} h^{x_{y,2}} \Big)^{c - c'} .$$

Let $SK_1^* = \sum_{(y,\pi) \in \mathcal{PK}^*} x_{y,1}$ and $SK_2^* = \sum_{(y,\pi) \in \mathcal{PK}^*} x_{y,2}$. By the construction of the forking algorithm $\mathcal{GF}_{\mathcal{A}}$, we also know that $c \neq c'$, so that $\mathcal{B}$ can compute

$$x'_1 \leftarrow (s_1 - s'_1)/(c - c') - SK_1^* \bmod q$$
$$x'_2 \leftarrow (s_2 - s'_2)/(c - c') - SK_2^* \bmod q$$

so that $g^{x'_1} h^{x'_2} = y^* = g^{x_1} h^{x_2}$. If $(x'_1, x'_2) = (x_1, x_2)$, then we say that event **bad** occurred and $\mathcal{B}$ outputs `fail`. Otherwise, $\mathcal{B}$ outputs $\log_g h = \frac{x_1 - x'_1}{x'_2 - x_2} \bmod q$.

Using the bounds of Lemma 2.4, we know that if $q > 8nq_{\mathrm{H}}/\epsilon$, where $n = |\mathcal{PK}|$, then $\mathcal{B}$ runs in time at most $\tau \cdot 8n^2 q_{\mathrm{H}}/\epsilon \cdot \ln(8n/\epsilon)$ and succeeds with probability

$$\epsilon' \geq \epsilon/8 - \Pr[\mathbf{bad}] .$$

We have left to bound the probability $\Pr[\mathbf{bad}]$. We will do so by arguing that in each individual run of $\mathcal{A}$, its view is information-theoretically independent of $\mathcal{B}$'s choice for (x_1, x_2). Namely, let $\mathcal{P} = \{(x'_1, x'_2) : y^* = g^{x'_1} h^{x'_2}\}$ be the set of q possible secret keys underlying $pk^* = (y^*, \pi^*)$. Based on just y^*, each of the q elements of $\mathcal{P}$ is equally likely to be (x_1, x_2), and since π^* is computed without using (x_1, x_2), this remains true for the entire pk^*. We now show that the signing oracle does not leak any further information about (x_1, x_2). Let the transcript of an interaction with the signing oracle in the role of signer S_i be given by $\big((\mathcal{T}, m, \{(t_j, PK_j)\}_{j \in \mathcal{C}_i}), (t_i, PK_i), (\bar{t}, PK, \{s_j\}_{j \in \mathcal{C}_i}), (s_{i,1}, s_{i,2})\big)$, where (t_i, PK_i) and $(s_{i,1}, s_{i,2})$ are outputs of the signing oracle and the other values are provided by $\mathcal{A}$. In an honest signing protocol, the oracle's outputs follow a distribution such that $s_{i,1}$ and $s_{i,2}$ are

uniformly distributed over $\mathbb{Z}_q$ (due to the random choice of r_1 and r_2), while t_i is the unique value such that

$$g^{s_{i,1}-\sum_{j\in C_i} s_{j,1}} h^{s_{i,2}-\sum_{j\in C_i} s_{j,2}} = \frac{t_i}{\prod_{j\in C_i} t_j} \cdot y^{*c} ,$$

where $c = \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$. We argue that for any $(x_1', x_2') \in \mathcal{P}$, there exists exactly one choice for $(r_1', r_2') \in \mathbb{Z}_q^2$ such that the honest signer would have produced the exact same outputs $(t_i, PK_i), (s_{i,1}, s_{i,2})$ if it would have used (x_1', x_2') as the secret key, namely,

$$r_1' = s_{i,1} - \sum_{j\in C} s_{j,1} - cx_1' \bmod q$$
$$r_2' = s_{i,2} - \sum_{j\in C} s_{j,2} - cx_2' \bmod q .$$

The signing oracle therefore doesn't leak any further information about the secret key (x_1, x_2).

As (x_1', x_2') is computed from $\mathcal{A}$'s output after two executions that are information-theoretically independent of (x_1, x_2), the extracted pair (x_1', x_2') must also be independent of (x_1, x_2), so we have that

$$\Pr[\mathbf{bad}] = 1/q .$$

from which the bounds in the theorem follow. □

4.3 Variants and Caveats

Obtaining security in the plain public-key model. The DG-CoSi scheme as described in Section 4.1 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 value as exponents for each public key in the verification equation [8], or by using a product of hash values as exponents [20, 21]. However, these schemes would be less efficient in terms of verification or 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. As security in the key-verification setting implies security in the KOSK setting, DG-CoSi 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. Consequently, the domain separation on the hash function can be ommitted. More interestingly, the aggregate public key *PK* no longer needs to be included in the hash and setting $c \leftarrow \mathsf{H}(\bar{t}, m)$ 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 ommitted. 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 DG-CoSi 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 DG-CoSi includes the aggregate public key *PK* in the hash $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$. Without including *PK*, as was done for example in the CoSi scheme, the extension to multisets becomes insecure, because a signature (c, s_1, s_2) 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 $c/c' \bmod q$, where $c' = \mathsf{H}(\bar{t}, m')$.

Collision attacks. Bagherzandi et al.'s forking lemma [3] 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 [25, 32] 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 $y = g^{x_1} h^{x_2}$ and the honest signer with public key y^* the only child. On input m, the honest signer returns t_1. The forger then repetitively generates random values $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$ and computes $\bar{t} \leftarrow g^{r_1} h^{r_2} \cdot t_1$ and hash values $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$ and $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m')$ for $PK = y \cdot y^*$ and $m' \neq m$ until it finds two values $\bar{t}, \bar{t}'$ such that $\mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) = \mathsf{H}(\mathtt{sig}, \bar{t}', PK, m')$, which for an ℓ-bit hash function is expected to happen after $O(2^{\ell/2})$ tries. It then sends $(\bar{t}, PK)$ to the honest signer, who responds with $s_{1,1}, s_{1,2}$ such that $g^{s_{1,1}} h^{s_{1,2}} = t_1 y^{*c}$ for $c = \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m) = \mathsf{H}(\mathtt{sig}, \bar{t}', PK, m')$. If r_1', r_2' are the random values that $\mathcal{F}$ used to generate $\bar{t}' = g^{r_1'} h^{r_2'}$, then $\mathcal{F}$ outputs $(c, s_1 = r_1' + cx_1 + s_{1,1} \bmod q, s_2 = r_2' + cx_2 + s_{1,2} \bmod q)$ as a valid forgery on m'.

5 EVALUATION

We presented DG-CoSi 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 DG-CoSi and show that the burden of using a double generator will not have a significant impact on the efficiency of the system.

5.1 Experiment Setup

Prototype. We implemented DG-CoSi in the Go programming language as an extension to the Collective Authority project (Cothority). We used the Cothority [14] and Onet [13] libraries to provide support for the tree-based collective signing as used in CoSi and DG-CoSi. This experiment compares the latest version of CoSi to an implementation of DG-CoSi. Note that CoSi has been further developed since its original publication [35], which explains the small differences in performance measurements between their work and our results.

Physical configuration. A DeterLab [2] testbed was used to evaluate our system. The testbed consists of 32 physical machines, each

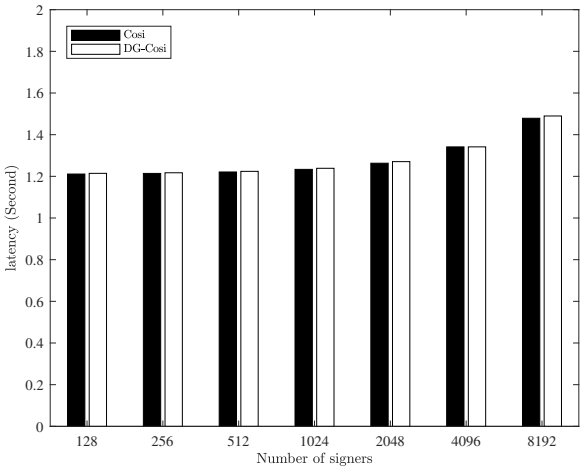

Figure 4: Comparing end-to-end latency of CoSi and DG-CoSi signing with varying amounts of signers.

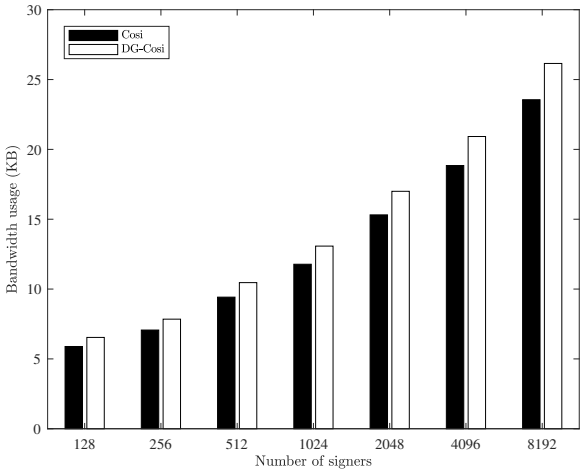

Figure 5: Bandwidth consumption (sent and received combined) of CoSi and DG-CoSi with varying amounts of signers.

containing an Intel Xeon E3-1260L processor and 16GB of RAM. Every physical machine simulated up to 256 signers for a total of 8192 signers. A round-trip delay of 200 miliseconds 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. DG-CoSi 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. Experimenting showed that a low tree depths (and high branching factors) provide good results, 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 managable, 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 DG-CoSi 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 indiviual runs.

5.2 Results

Signing Latency. To evaluate the scalability of DG-CoSi, 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 8192 signers. Fig. 4 depicts the results, showing that DG-CoSi can easily scale to 8192 signers, yielding a signature in less than 1.5 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 DG-CoSi scales as well as CoSi does, only marginally increasing the overall latency compared to CoSi.

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 the bandwidth consumption of CoSi and DG-CoSi. We observe a fixed increase of 11% in the bandwidth cost, independent of the number of signers. The main differences concerning network usage between CoSi and DG-CoSi are that the latter computes the aggregate public key PK_i in the tree, and that DG-CoSi has two s-values in the response rather than a single s-value for CoSi. One may expect that these changes result in a more significant difference in bandwidth usage, but the overhead of the connection and communicating the tree structure reduced the gap between the two schemes. We believe an 11% increase in the bandwidth is a very acceptable overhead to gain provable security and will not hinder the system's scalability.

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 averaging.

Computation Cost. Our final experiment compares the computational cost between CoSi and DG-CoSi, 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 an 32% average increase from CoSi to DG-CoSi, which we expect is due to the multi-exponentiation required to compute t_i. Overall, DG-CoSi is still extremely efficient, as a total CPU time of 1.47 seconds for 8192 signers means every individual signer spends 0.18 milliseconds on average.

6 CONCLUSION

Our work provides substantial evidence that the CoSi and MuSig multisignature schemes cannot be proven secure under standard

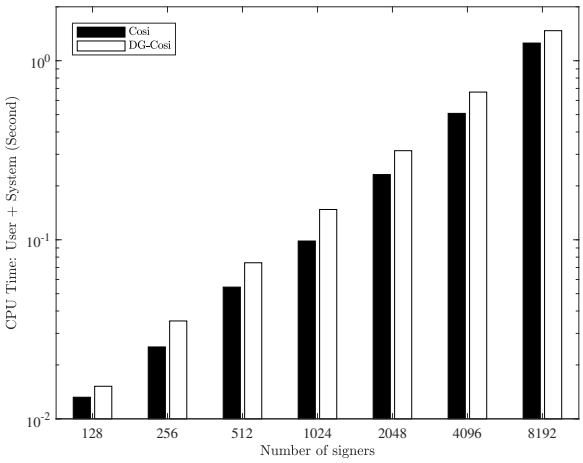

Figure 6: CPU time (User + System) of CoSi **and** DG-CoSi **with varying amounts of signers.**

assumptions, as any such proof would have to be non-algebraic, non-black-box, or under an assumption that is not implied by the one-more discrete logarithm assumption (unless the one-more discrete logarithm assumption turns out to be false). Given the status of provable security as a *sine qua non* in modern cryptographic protocol design and given its importance in the selection of industry standards, we surface the DG-CoSi scheme as a provably secure yet highly efficient alternative. Compared to the original CoSi scheme, our experiments yield a 32% increase in CPU time and no noticeable difference in signing latency, showing that DG-CoSi is essentially just as scalable as CoSi and is a viable alternative for use in large-scale decentralized systems.

REFERENCES

[1] 2017. Technology roadmap - Schnorr signatures and signature aggregation. https://bitcoincore.org/en/2017/03/23/schnorr-signature-aggregation. (2017).

[2] 2018. DeterLab: Cyber-Defense Technology Experimental Research Laboratory. https://www.isi.deterlab.net. (2018). [Online; accessed February-2018].

[3] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. 2008. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS 08*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, 449–458.

[4] Ali Bagherzandi and Stanislaw Jarecki. 2008. Multisignatures Using Proofs of Secret Key Possession, as Secure as the Diffie-Hellman Problem. In *SCN 08 (LNCS)*, Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti (Eds.), Vol. 5229. Springer, Heidelberg, 218–235.

[5] Foteini Baldimtsi and Anna Lysyanskaya. 2013. On the Security of One-Witness Blind Signature Schemes. In *ASIACRYPT 2013, Part II (LNCS)*, Kazue Sako and Palash Sarkar (Eds.), Vol. 8270. Springer, Heidelberg, 82–99. https://doi.org/10.1007/978-3-642-42045-0_5

[6] Boaz Barak. 2004. *Non-Black-Box Techniques in Cryptography*. Ph.D. Dissertation.

[7] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. 2003. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *Journal of Cryptology* 16, 3 (June 2003), 185–215.

[8] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS 06*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, 390–399.

[9] Mihir Bellare and Adriana Palacio. 2002. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *CRYPTO 2002 (LNCS)*, Moti Yung (Ed.), Vol. 2442. Springer, Heidelberg, 162–177.

[10] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC 2003 (LNCS)*, Yvo Desmedt (Ed.), Vol. 2567. Springer, Heidelberg, 31–46.

[11] Dan Boneh and Ramarathnam Venkatesan. 1998. Breaking RSA May Not Be Equivalent to Factoring. In *EUROCRYPT'98 (LNCS)*, Kaisa Nyberg (Ed.), Vol. 1403. Springer, Heidelberg, 59–71.

[12] Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. 2017. 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*. IEEE, 23–26. https://doi.org/10.1109/EuroSPW.2017.46

[13] Dedis. 2018. Cothority overlay network library. https://github.com/dedis/onet. (2018). [Online; accessed February-2018].

[14] Dedis. 2018. Scalable collective authority prototype. https://github.com/dedis/cothority. (2018). [Online; accessed February-2018].

[15] Bryan Ford, Nicolas Gailly, Linus Gasser, and Phillipp Jovanovic. 2017. *Collective Edwards-Curve Digital Signature Algorithm*. Internet-Draft draft-ford-cfrg-cosi-00.txt. IETF Secretariat.

[16] K. Itakura and K. Nakamura. 1983. A Public-Key Cryptosystem suitable for Digital Multisignatures. *NEC Research & Development* 71 (1983), 1–8.

[17] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 279–296. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias

[18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. 2017. OmniLedger: A Secure, Scale-Out, Decentralized Ledger. Cryptology ePrint Archive, Report 2017/406. (2017). http://eprint.iacr.org/2017/406.

[19] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. 2006. Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In *EUROCRYPT 2006 (LNCS)*, Serge Vaudenay (Ed.), Vol. 4004. Springer, Heidelberg, 465–485.

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

[21] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2018. Simple Schnorr Multi-Signatures with Applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068. (2018). https://eprint.iacr.org/2018/068.

[22] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. 2001. Accountable-Subgroup Multisignatures: Extended Abstract. In *ACM CCS 01*. ACM Press, 245–254.

[23] Markus Michels and Patrick Horster. 1996. On the Risk of Disruption in Several Multiparty Signature Schemes. In *ASIACRYPT'96 (LNCS)*, Kwangjo Kim and Tsutomu Matsumoto (Eds.), Vol. 1163. Springer, Heidelberg, 334–345.

[24] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

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

[26] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. 2017. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 1271–1287. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/nikitin

[27] Kazuo Ohta and Tatsuaki Okamoto. 1993. A Digital Multisignature Scheme Based on the Fiat-Shamir Scheme. In *ASIACRYPT'91 (LNCS)*, Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto (Eds.), Vol. 739. Springer, Heidelberg, 139–148.

[28] Tatsuaki Okamoto. 1993. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *CRYPTO'92 (LNCS)*, Ernest F. Brickell (Ed.), Vol. 740. Springer, Heidelberg, 31–53.

[29] Pascal Paillier and Damien Vergnaud. 2005. Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log. In *ASIACRYPT 2005 (LNCS)*, Bimal K. Roy (Ed.), Vol. 3788. Springer, Heidelberg, 1–20.

[30] David Pointcheval and Jacques Stern. 2000. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology* 13, 3 (2000), 361–396.

[31] Thomas Ristenpart and Scott Yilek. 2007. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In *EUROCRYPT 2007 (LNCS)*, Moni Naor (Ed.), Vol. 4515. Springer, Heidelberg, 228–245.

[32] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991), 161–174.

[33] Victor Shoup. 1997. Lower Bounds for Discrete Logarithms and Related Problems. In *EUROCRYPT'97 (LNCS)*, Walter Fumy (Ed.), Vol. 1233. Springer, Heidelberg, 256–266.

[34] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. 2017. Scalable Bias-Resistant Distributed Randomness. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 444–460. https://doi.org/10.1109/SP.2017.45

[35] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping Authorities"Honest or Bust" with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 526–545.

https://doi.org/10.1109/SP.2016.38