On the Provable Security of Two-Round Multi-Signatures

Manu Drijvers
IBM Research – Zurich and ETH

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 highly efficient two-round 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). Previous two-round Schnorr-based schemes by Bagherzandi et al. (CCS 2008) and Ma et al. (DCC 2010) are less efficient in terms of signature size, signing time, or verification time. In this work, we prove that none of these 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 each of the above schemes (except CoSi, which was not proved secure) to explain the contradiction between our result and the existing proofs.

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 [2, 7, 9, 14, 17, 18, 20, 24], they have recently received renewed interest because of the rise of distributed applications that aim to decentralize trust such as Bitcoin [22] 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. [32] 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 [31], improving the scalability of Bitcoin [31], and introducing a decentralized software update framework [23]. CoSi is also considered for standardization by the IETF [13].

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 team of Bitcoin core developers published the MuSig scheme [19] 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.

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 needs to be programmed depends on the output of the adversarial signers. Bellare and Neven [7] 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 [2]'s BCJ scheme eliminated the need for this extra round by using homomorphic trapdoor commitments, while Ma et al.'s MWLD scheme [18] simulates signatures by exploiting the witness-indistinguishability of Okamoto signatures [25].

Our result. This paper essentially shows that none of the two-round schemes mentioned above can be proved secure under standard assumptions. More precisely, we prove that if the one-more discrete log 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 log (DL) or OMDL assumption.

This is surprising, because all of these schemes, except 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 forking argument 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 (so-called *forking* [27]). Also, given that all of the covered schemes are derived from Schnorr signatures [29], it would be very surprising if its 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. Another way to circumvent our impossibility result is to assume the much stronger generic group model [30].

Versions of this paper. An earlier version of this paper [12] contained the impossibility proof for the CoSi and MuSig schemes, and incorrectly suggested a new scheme called DG-CoSi as a provably secure alternative. This version adds 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.

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 [6, 8]). 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 [10] 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 [26]:

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 [27] to analyze the security of Schnorr signatures [29]. 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 [7] 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 [2] 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 $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$. Let $f = (\rho, h_1, \ldots, h_{q_\mathsf{H}})$ 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 $\{1, \ldots, q_\mathsf{H}\}$ 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:

```
$\mathcal{GF}_\mathcal{A}(in)$:
    $f = (\rho, h_1, \ldots, h_{q_\mathsf{H}}) \xleftarrow{\$} \Omega$
    $(J, \{out_j\}_{j \in J}) \leftarrow \mathcal{A}(in, f)$
    If $J = \emptyset$ then output fail
    Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \leq \ldots \leq j_n$
    For $i = 1, \ldots, n$ do
        $succ_i \leftarrow 0$ ; $k_i \leftarrow 0$ ; $k_{\max} \leftarrow 8nq_\mathsf{H}/\epsilon \cdot \ln(8n/\epsilon)$
        Repeat until $succ_i = 1$ or $k_i > k_{\max}$
            $f'' \xleftarrow{\$} \Omega$ such that $f''|_{j_i} = f|_{j_i}$
            Let $f'' = (\rho, h_1, \ldots, h_{j_i - 1}, h''_{j_i}, \ldots, h''_{q_\mathsf{H}})$
            $(J'', \{out''_j\}_{j \in J''}) \leftarrow \mathcal{A}(in, f'')$
            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 [2]). *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* $q > 8nq_\mathsf{H}/\epsilon$, *then* $\mathcal{GF}_\mathcal{A}(in)$ *runs in time at most* $\tau \cdot 8n^2 q_\mathsf{H}/\epsilon \cdot \ln(8n/\epsilon)$ *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. [2], which follows the so-called key-verification model, as introduced by Bagherzandi and Jarecki [3], 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 $par \leftarrow$ 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 $par \leftarrow$ 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 MS = (Pg, Kg, Sign, KAg, Vf, KVf) is defined by a three-stage game.

Setup. The challenger generates the parameters $par \leftarrow$ 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.

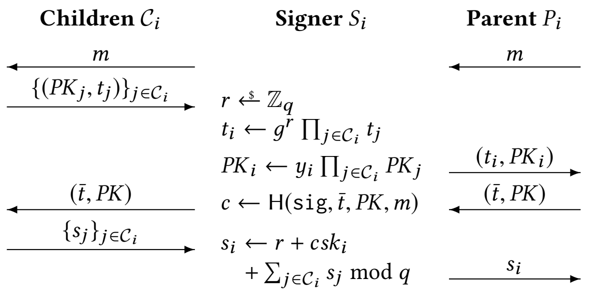
Children $\mathcal{C}_i$
Signer S_i
Parent P_i
m
m
$\{(PK_j, t_j)\}_{j \in \mathcal{C}_i}$
$r \xleftarrow{\$} \mathbb{Z}_q$
$t_i \leftarrow g^r \prod_{j \in \mathcal{C}_i} t_j$
$PK_i \leftarrow y_i \prod_{j \in \mathcal{C}_i} PK_j$
(t_i, PK_i)
$(\bar{t}, PK)$
$c \leftarrow \mathsf{H}(\mathtt{sig}, \bar{t}, PK, m)$
$(\bar{t}, PK)$
$\{s_j\}_{j \in \mathcal{C}_i}$
$s_i \leftarrow r + c sk_i$
$+ \sum_{j \in \mathcal{C}_i} s_j \bmod q$
s_i

Figure 1: The 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.

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 $(\tau, q_S, q_C, q_H, \epsilon)$-forger for multisignature scheme MS = (Pg, Kg, Sign, 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 $(\tau, q_S, q_C, q_H, \epsilon)$-unforgeable if no $(\tau, q_S, q_C, q_H, \epsilon)$-forger exists.

2.5 The CoSi Multisignature Scheme

CoSi is a multisignature scheme introduced by Syta et al. [32] that follows a long line of work on Schnorr-based multisignatures [2, 7, 18, 20, 28]. 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 [11, 15, 16, 23, 31] and is considered for standardization by the IETF [13]. However,

[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.

the security of CoSi is not formally analyzed, as Syta et al. [32] do not formally prove security.

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

2.5.2 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 [21], 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.

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

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

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

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

2.6 The BCJ Multisignature Schemes

Whereas Bellare and Neven [7] 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. [2] 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.

2.6.1 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. Output $par \leftarrow (\mathbb{G}, g_1, h_1, g_2, h_2, q)$.

[3]We make a small variation of the original CoSi scheme [32], 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.

2.6.2 Key generation. The key generation algorithm $\mathsf{Kg}(par)$ takes $sk \xleftarrow{\$} \mathbb{Z}_q$ and sets $y \leftarrow g^{sk}$. 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).

2.6.3 Signing. Signing is the four-step protocol. A signer S_i on input $\mathsf{Sign}(par, (sk_i, y_i, \pi_i), m, \mathcal{T})$ 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 $s \leftarrow s_i$, $\gamma_1 \leftarrow s_{i,1}$, $\gamma_2 \leftarrow s_{i,2}$, and outputs $\sigma \leftarrow (\bar{t}_1, \bar{t}_2, s, \gamma_1, \gamma_2)$.

2.6.4 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 $PK \leftarrow \prod y_i$.

2.6.5 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 Multisignature Scheme

The MWLD scheme due to Ma et al. [18] addresses the signature simulation problem by using a witness-indistinguishable proof based on Okamoto signatures [25], yielding shorter signatures and more efficient signing than the BCJ scheme.

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

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

2.7.3 Signing. Signing is the four-step protocol. A signer S_i on input $\mathsf{Sign}(par, ((sk_1, sk_2), pk_i), m, \mathcal{T})$ 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, 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 $s_{i,b} \leftarrow r_{i,b} + v_i \cdot sk_{i,b} + \sum_{j \in \mathcal{C}_i} s_{j,b}$ 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 $\sigma \leftarrow (c, s_1, s_2)$.

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

2.7.5 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

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

3 THE SECURITY OF TWO-ROUND MULTISIGNATURES USING REWINDING

In this section, we analyze the security of existing two-round multisignature 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 [27], 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 [8, 19], 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. [19] 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 [8] 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 [4] 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 [4]. 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 $((2\ell+1)\tau_{\text{exp}} + O(\ell), \ell, \ell, 3, 1 - 1/q)$-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.*

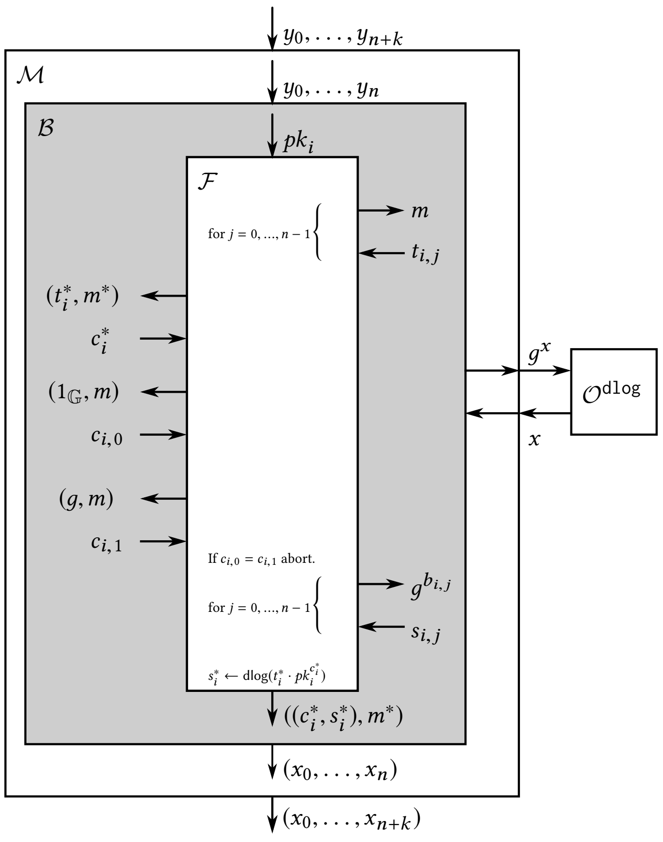

Figure 2: Our metareduction $\mathcal{M}$ in the proof of Theorem 3.1, which simulates forger $\mathcal{F}$ towards any reduction $\mathcal{B}$ that would prove the security of CoSi **under the OMDL assumption, and uses $\mathcal{B}$ to break the OMDL problem.**

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, 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 [5], 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 [30]. 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 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 $\bar{t}_i^* \leftarrow \mathrm{target}(pk_i, t_{i,1}, \ldots, t_{i,\ell})$.
- $\mathcal{F}$ 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 $b_{i,1} \| \ldots \| b_{i,\ell} \leftarrow \mathrm{rand}((pk_i, t_{i,1}, \ldots, t_{i,\ell}), (c_i^*, c_{i,0}, c_{i,1}))$ 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 $s_i^* \leftarrow \mathrm{forge}(\bar{t}_i^*, pk_i, c_i^*)$.

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 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 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 $s_i^* \leftarrow \mathcal{O}^{\mathrm{dlog}}(y_{j_C} \cdot pk_i^{c_i^*})$ 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 $\alpha_j \leftarrow \mathcal{O}^{\mathrm{dlog}}(y_j)$ 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 $\mathcal{O}^{\mathrm{dlog}}$ query for each of the k equations and at most n $\mathcal{O}^{\mathrm{dlog}}$ queries to respond to $\mathcal{B}$'s $\mathcal{O}^{\mathrm{dlog}}$ 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 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 multisignature scheme MuSig as recently introduced by Maxwell et al. [19]. 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 demonstrated 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.

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 [18] is flawed.[4] 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 random-oracle 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 $(s_{i,1}^*, s_{i,1}^*) \leftarrow \mathsf{forge}(\bar{t}_i^*, pk_i, v_i^*)$.

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

[4] To be precise, Claim 4 of the work is incorrect: while the view of the forger is independent of (sk_1, sk_2), rewinding can cause the reduction to leak information, and event E1 may therefore occur with a non-negligible probability. Our metareduction will cause E1 to occur with probability 1.

$(v_{i',b_{i',1}},\ldots,v_{i',b_{i',\ell}}) = (v_{i,b_{i,1}},\ldots,v_{i,b_{i,\ell}})$. 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 $\mathcal{O}^{\mathrm{dlog}}$ to compute $s^*_{i,1} \leftarrow \mathcal{O}^{\mathrm{dlog}}(\bar{t}^*_i \cdot pk_i^{v^*_i} \cdot h^{-s^*_{i,2}})$ 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 [2] is flawed.[5]

- 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 $\gamma^*i, 1 \leftarrow \gamma^*_{i',1} + (c^*_i - c^*_{i'})\delta_{\gamma_1}$, $\gamma^*i, 2 \leftarrow \gamma^*_{i',2} + (c^*_i - c^*_{i'})\delta_{\gamma_2}$, $s^*_i \leftarrow s^*_{i'} + (c^*_i - c^*_{i'})\delta_s$.
- 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 $\mathcal{O}^{\mathrm{dlog}}$ to compute $s^*_i \leftarrow \mathcal{O}^{\mathrm{dlog}}(\bar{t}^*_{i,2} g_2^{-\alpha i,1} h_2^{\alpha i,2} y_i^{c^*_i})$ and returns $(\gamma_{i,1}, \gamma_{i,2}, s_i)$.

4 CONCLUSION

Our work provides substantial evidence that none of the currently known two-round Schnorr-based multi-signature schemes (BCJ, MWLD, CoSi, and MuSig) can be proven secure under standard 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).

REFERENCES

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

[2] 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.

[3] 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.

[4] 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

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

[6] 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.

[5] The security proof distinguishes two forgery events: In E1, the forger broke the binding property of the commitment scheme, and in E2, 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 E1 occurs with $\mathcal{B}_1$ or E2 occurs with $\mathcal{B}_0$. The proof argues that the forger cannot distinguish $\mathcal{B}_0$ and $\mathcal{B}_1$, and therefore, each event is equally likely to occur with either of the reduction. However, the reductions are distinguishable, as rewinding causes the reduction to leak either the commitment key backdoor or the honest signer secret key. Our metareduction will always forge through E1 with $\mathcal{B}_0$ and through E2 with $\mathcal{B}_1$.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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

[12] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, and Gregory Neven. 2018. 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.

[13] 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.

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

[15] 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

[16] 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.

[17] 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.

[18] 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.

[19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2018. Simple Schnorr Multi-Signatures with Applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068, Version 20180118:124757. (2018).

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

[21] 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.

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

[23] 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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

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

[31] 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

[32] 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