

One-More Unforgeability for Multi- and Threshold Signatures

Sela Navot and Stefano Tessaro

Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, WA, USA
{senavot,tessaro}@cs.washington.edu

Abstract. This paper initiates the study of one-more unforgeability for multi-signatures and threshold signatures as a stronger security goal, ensuring that ℓ executions of a signing protocol cannot result in more than ℓ signatures. This notion is widely used in the context of blind signatures, but we argue that it is a convenient way to model strong unforgeability for other types of distributed signing protocols. We provide formal security definitions for one-more unforgeability (OMUF) and show that the HBMS multi-signature scheme does not satisfy this definition, whereas MuSig and MuSig2 do. We also show that mBCJ multi-signatures do not satisfy OMUF, as well as expose a subtle issue with their existential unforgeability (which does not contradict their original security proof). For threshold signatures, we show that FROST satisfies OMUF, but ROAST does not.

Table of Contents

1	Introduction	3
2	Preliminaries	4
3	Specifications and Usage	5
3.1	Multi-Signatures	5
3.2	Threshold Signatures	6
4	Existential and Strong Unforgeability	7
4.1	Extending Strong Unforgeability to Multi-Signatures.....	8
4.2	Strong Unforgeability of Threshold Signatures	10
5	Multi-Signature Schemes	11
5.1	Analysis of HBMS	11
5.2	Analysis of MuSig	14
5.3	Analysis of MuSig2	18
5.4	Analysis of mBCJ	20
6	Threshold Signatures Schemes	23
6.1	Comparison to Previous SUF-TS Definition, and FROST	23
6.2	Analysis of ROAST	25
7	Acknowledgements	26
	References	27
A	Toy Strongly Unforgeable Multi-Signature Scheme	30
B	SUF of Underlying Plain Signature Schemes	30
B.1	Underlying Scheme of HBMS	31
B.2	Underlying Scheme of mBCJ.....	32
C	Proof of Lemma 5	34
D	Statement and Proof of Lemma 10	40

1 Introduction

There has been growing interest in protocols for distributed generation of signatures, in the form of *threshold signatures* (TS) [22, 23] and *multi-signatures* (MS) [29]. While these primitives have been studied for decades, their recent widespread use has been driven by applications in blockchain ecosystems, such as digital wallets [27], and to enforce the need for multiple signatures to authorize a transaction. Threshold signatures are also at the center of standardization efforts by NIST [38] and IETF [18].

Recall that in a t -out-of- n threshold signature, a secret signing key, associated with a public verification key, is secret shared amongst a set of n signers (often as the result of running a distributed key generation protocol). Any subset of at least t signers should be able to sign a message, whereas an adversary corrupting fewer than t signers should not be able to come up with a signature on their own. In multi-signatures, in contrast, parties generate their own keys, independently. Then, any group of signers can come together to generate signature shares and aggregate them into a multi-signature, which can be verified using a verification key obtained by aggregating the verification keys of all involved signers.

SECURITY DEFINITIONS. Security definitions for distributed signing are far more challenging than definitions for signatures in isolation. A key point is that issuance of signatures generally involves an *interactive* protocol (this is the case for all pairing-free schemes, such as FROST [31, 6] and MuSig/MuSig2 [37, 42]), and executions are subject to adversarial corruptions. Often, the adversary can not only corrupt a subset of the signers, but also control communication between signers—this is the case for a common model where inter-signer communication is mediated by a proxy.

This makes it hard to define *when* a signature on a message has been issued, and, in turn, to formalize a notion of unforgeability. A number of works (for example, [9, 37, 14, 7, 20]) sidestep this question by considering a message signed as long as a signing session *started* on it. In other cases the definition is tailored to the specific structure of the scheme (for example, in the analysis of MuSig2 [42]) or a very limited class of schemes, as in [6, 12], where Bellare et al. put forward a hierarchy of security notions for partially non-interactive threshold signatures.

STRONG UNFORGEABILITY. This paper considers a further challenge in the study of security definitions for TS/MS, namely the definition of strong unforgeability. This standard notion of security for plain signatures ensures that, in addition to achieving regular unforgeability, an adversary cannot come up with a different signature for a message for which it has already seen valid signatures. It is natural to expect that a distributed signing protocol for a strongly unforgeable signature scheme, like Schnorr signatures [47, 44], should also ensure strong unforgeability. However, somewhat jumping ahead, we will show that in general this is *not true*: there are strongly unforgeable signature schemes with distributed signing protocols that are not strongly unforgeable.

Beyond theoretical interest, this may be of practical interest in the context of blockchain ecosystems, where multi-signatures are used to generate strongly unforgeable plain signatures [41, 52]. Lack of strong unforgeability of plain signatures has attracted attention [51, 33, 52] and has historically been associated with costly transaction malleability attacks [21, 2], warranting the study and prevention of similar weaknesses that may be introduced by the usage of multi-signatures. We point out that such a weakness is, to some extent, inherent in probabilistic threshold signature schemes, since not all signers are required to participate in each signing session. When corrupting enough signers, an adversary can obtain multiple signatures for a message even if each honest signer only signs the message once, allowing execution of a malleability attack. Moreover, regardless of the number of corrupt signers, an adversary can obtain more signatures for some message than the maximum number of signature shares that any single signer provided.

Our work on threshold signatures, however, is motivated by NIST’s interest in strong unforgeability, which is part of the requirements for submissions to its threshold signatures call [15], and we seek to provide a formalization candidate candidate schemes could use. Furthermore, system designers often assume strong unforgeability in unpredictable ways, especially when the underlying scheme is known to be strongly unforgeable, and it is prudent not to break this guarantee when the signatures are issued in a threshold setting.

It turns out that a rigorous definition of strong unforgeability for distributed signing is challenging, as the winning condition requires defining which signatures have been generated by interactive signing protocols subject to adversarial behavior, but it is not always clear how to do this. The security notions of Bellare et al. [6, 12], for example, give definitions of strong unforgeability for a limited class of semi non-interactive

threshold signatures where the signature is uniquely defined by the input to one of the signing rounds, but this is not a property we expect a protocol to have, and no general definition is known.

THIS PAPER: ONE-MORE UNFORGEABILITY. In order to give a generic definition of strong unforgeability, this paper proposes the notion of *one-more unforgeability* (OMUF) as the better approach to model strong unforgeability. OMUF requires that after a certain number ℓ of executions of the signing protocol for a message m , the adversary can generate no more than ℓ signatures for that message. A similar notion is widely used for blind signatures and was introduced by Poitcheval and Stern [43, 45], and we argue that it is natural for distributed signing. In particular, for non-distributed plain digital signatures, OMUF and the classical definition of strong unforgeability are in fact equivalent.

OUR CONTRIBUTIONS. Concretely, we make the following contributions.

- **New Definitions.** We formalize the notion of one-more unforgeability for multi-signature and threshold signature schemes.
- **Attacks.** We show that the HBMS multi-signature scheme [7] does not satisfy one-more unforgeability using a polynomial time attack based on the algorithm of Benhamouda et al. [13] to solve the ROS problem [48]. We also show that mBCJ multi-signatures [24] do not satisfy OMUF using a similar attack, and reveal a subtle issue with their existential unforgeability: an adversary can use signatures for a message m and a signing set S to forge a signature for the same message that is valid for a different arbitrary signing set.¹ Lastly, we point out that the ROAST threshold signature scheme [46] is not strongly unforgeable using a trivial attack. This is despite the fact that the underlying plain signatures that HBMS and mBCJ generate are strongly unforgeable, which we prove, and that the standard instantiation of ROAST produces ordinary Schnorr signatures.
- **Proofs of security.** We prove that the MuSig [37] and MuSig2 [42] multi-signature schemes satisfy our one-more unforgeability definition. We also show that a previous security proof [6, 12] implies that FROST [32, 6, 19] satisfies one-more unforgeability, assuming idealized key generation.

ON UC SECURITY. We stress that in this paper we target game-based definitions of security for distributed signing primitives. An alternative approach is to consider schemes with UC Security [16], as done in [35], for example. It is not hard to see that a UC-secure threshold signature which implements signatures for a strongly unforgeable signature scheme has to achieve one-more unforgeability. However, many practical threshold and multi-signatures are not UC secure, and hence our game-based approach is meant to capture strong unforgeability for a broader class of schemes.

2 Preliminaries

GAMES FRAMEWORK. We use the game playing framework of [11] for all security definitions and hardness assumptions, with minor simplifications.

A game consists of an initialization algorithm (INIT), finalization algorithm (FIN), and any number of algorithms that can be queried as oracles. When a randomized algorithm \mathcal{A} (usually called an adversary) plays a game Gm, which we denote by $\text{Gm}(\mathcal{A})$, \mathcal{A} is executed with the output of INIT as its input. \mathcal{A} may query the oracles repeatedly at the cost of a single time unit per query. When \mathcal{A} terminates, FIN is executed with the output of \mathcal{A} and outputs true or false, which is the output of the game. We use $\text{Pr}[\text{Gm}(\mathcal{A})]$ as a shorthand for $\text{Pr}[\text{Gm}(\mathcal{A}) = \text{true}]$ where the probability is taken over the randomness of \mathcal{A} and Gm. A game may have parameters params , such as a group used by the game or the number of permitted queries to some oracle.

All schemes and hardness assumptions in this paper are parameterized by an underlying group \mathbb{G} of publicly known prime order p , and their security parameter is $\log(p)$.

Definition 1. We define the advantage of an adversary \mathcal{A} against an assumption ASMP defined by the game $\text{Gm}_{\text{params}}^{\text{asmp}}$ as

$$\text{Adv}_{\text{params}}^{\text{asmp}}(\mathcal{A}) := \text{Pr}[\text{Gm}_{\text{params}}^{\text{asmp}}(\mathcal{A})].$$

The assumption ASMP holds if $\text{Adv}_{\text{params}}^{\text{asmp}}(\mathcal{A})$ is negligible for all polynomial time adversaries \mathcal{A} , where polynomial and negligible are in terms of the security parameter defined by params .

¹ Note that this attack does not contradict the security proof of [24], since their security definition does not cover such forgeries.

Definition 2. Let S be a cryptographic scheme with scheme parameters params and suppose DEFN is a security definition defined by the game $\mathbf{G}^{\text{defn}}[S]$. We define the advantage of an adversary \mathcal{A} against S as

$$\text{Adv}_S^{\text{defn}}(\mathcal{A}) := \Pr[\mathbf{G}^{\text{defn}}[S](\mathcal{A})].$$

The scheme S is DEFN -secure if $\text{Adv}_S^{\text{defn}}(\mathcal{A})$ is negligible for all polynomial time adversaries \mathcal{A} , where polynomial and negligible are in terms of the security parameter defined by params .

Definitions 1 and 2 convert a game definition to a concrete assumption or security definition. Thus, in the rest of the paper, we only write the game definitions.

All of our security proofs are in the random oracle model (ROM) [10], where hash functions are modeled as random oracles. Our security definitions do not rely on the ROM.

NOTATION. We use multiplicative notation for all groups except for \mathbb{Z}_p , which denotes the integers modulus p . Addition and multiplication operations of \mathbb{Z}_p elements are modular. Logarithms use base 2.

In pseudocode, we use \leftarrow for assignment and \leftarrow_s for randomized assignment. In particular, $x \leftarrow_s S$ denotes sampling an element uniformly at random from a finite set S and $x \leftarrow_s \mathcal{A}(x_1, \dots)$ denotes assigning the output of a randomized algorithm \mathcal{A} with uniformly random random tape and input x_1, \dots to x . We use \perp to denote an error value, and use subscripts for array indexing. All variables are assumed to be uninitialized until assigned a value. Arrays and lists are one-indexed.

3 Specifications and Usage

3.1 Multi-Signatures

A multi-signature scheme allows a group of signers to provide a succinct joint signature for an agreed upon message. More specifically, a valid multi-signature by a group of n signers intends to convince verifiers that each of the n signers have participated in the signing protocol in order to sign this message with this group of signers.

In this paper, we primarily consider multi-signature in the plain public key model [9], the setting where each signer has their own long-standing public key that they generate independently (as opposed to using a distributed key generation protocol). This allows signers to use the same public key with multiple signing groups.

KEY AGGREGATION. A multi-signature scheme supports key aggregation if a signature can be verified using a single short key, called the aggregate key of the group, as opposed to the public keys of all the signers. In particular, MuSig [37] and MuSig2 [42] produce ordinary Schnorr signatures that can be verified with respect to the aggregate key.

BROADCASTING VERSUS AN AGGREGATOR NODE. In our syntax, the signers sends the output of each signing round to every other signer. It is sometimes more efficient to use an aggregator node (may be one of the signers) whose role is to aggregate the output of each signing round and forward it to the signers, as well as output the final multi-signature. Some authors describe schemes this way (for example [28, 42]) and every scheme can be described in this manner. Since the aggregator is not trusted and all the information available to the aggregator is also available to the adversary in our security model, using an aggregator node does not affect the unforgeability of schemes.

FORMAL SYNTAX AND CORRECTNESS. A multi-signature scheme MS is a collection of algorithms MS.Kg , $(\text{MS.Sign}_r)_{r=1}^{\text{MS.nr}}$, and MS.Verify , where nr is the number of signing rounds specified by the scheme. A scheme also specifies the last interactive round MS.lir , after which it is possible to construct a multi-signature without knowledge of the signers secret information. If a scheme supports key aggregation, it also has an MS.KeyAgg algorithm accompanied by MS.AggVer for key aggregation and for verification using the aggregated key. The intent of the algorithms is as follows:

Key generation: The randomized algorithm MS.Kg is used for key generation by each signing party individually. It takes no input apart from the scheme parameters and outputs a secret-public key pair.

Signing: The collection of algorithms $(\text{MS.Sign}_r)_{r=1}^{\text{MS.nr}}$ specifies the signing procedures to be run by each signing party, where MS.nr (the number of rounds) is specified by the scheme. Each round takes a subset of the following as input: a message, a vector of public keys along with the signer’s index in the vector, the

<p>Algorithm $\text{Exec}_{\text{MS}}((vk_i)_{i=1}^n, (sk_i)_{i=1}^n, m)$:</p> <ol style="list-style-type: none"> 1 For $i = 1, \dots, n$ do: 2 $i.\text{st}.sk \leftarrow sk_i, i.\text{st}.vk \leftarrow vk_i$ 3 out $\leftarrow (0)_{i=1}^n$ // output of current round 4 For $r = 1, \dots, \text{MS.nr}$ do: 5 For $i = 1, \dots, n$ do: 6 $(\sigma_i, i.\text{st}) \leftarrow \text{MS.Sign}_r(i.\text{st}, (vk_j)_{j=1}^n, m, \text{out})$ 7 out $\leftarrow (\sigma_i)_{i=1}^n$ 8 Return σ_1 	<p>Game $\mathbf{G}_{n,m}^{\text{ms-corr}}[\text{MS}]$</p> <p>FIN:</p> <ol style="list-style-type: none"> 1 For $i = 1, \dots, n$ do: 2 $(vk_i, sk_i) \leftarrow \text{MS.Kg}()$ 3 $\sigma \leftarrow \text{Exec}_{\text{MS}}((vk_i)_{i=1}^n, (sk_i)_{i=1}^n, m)$ 4 Return $\text{MS.Verify}((vk_i)_{i=1}^n, m, \sigma)$
--	---

Fig. 1. Left: an honest execution of the signing protocol of a multi-signature scheme MS . Note that signing rounds may only use a subset of the provided input. **Right:** a game defining the correctness of a scheme. A scheme satisfies perfect correctness for a natural number n if $\Pr[\mathbf{G}_{n,m}^{\text{ms-corr}}[\text{MS}]] = 1$ for each supported message m .

output of previous signing rounds, and some other information saved in the state of at most one signer (including the secret key). The algorithm produces an output and updates the state of the signing party, and the multi-signature is the output of the last round $\text{Sign}_{\text{MS.nr}}$. These algorithms may be randomized.

Key aggregation: If the scheme supports key aggregation, the algorithm MS.KeyAgg takes a list of n public keys $(vk_i)_{i=1}^n$ as input and outputs a single aggregated verification key.

Verification: If MS does not support key aggregation, then it has an algorithm MS.Verify that takes a list of public keys, a signature, and a message as input and returns a boolean value signifying whether the signature is valid. If MS supports key aggregation, it has the algorithm MS.AggVer with the same functionality that takes an aggregated public key as input instead of a list of public keys. In this case, a standard Verify algorithm can be obtained by setting $\text{MS.Verify}((vk_i)_{i=1}^n, m, \sigma) = \text{MS.AggVer}(\text{MS.KeyAgg}((vk_i)_{i=1}^n), m, \sigma)$. Hence, without loss of generality, we will only consider MS.Verify in the correctness and security definitions.

The signers maintain a state st which may change throughout the protocol. In particular, using the convention from [7], each signer i has a long-standing secret key $i.\text{st}.sk$ and public key $i.\text{st}.vk$ as well as information associated with each signing session s that they participate in, denoted by $i.\text{st}_s$. The session state includes $\text{st}_s.n$, $(\text{st}_s.vk_j)_{j=1}^{\text{st}_s.n}$, $\text{st}_s.m$, $\text{st}_s.\text{rnd}$, and $\text{st}_s.\text{me}$ which refers to the number of parties, the public keys of those signers, the message being signed, the last completed signing round, and the index of the party within the signers. It is required that signers refuse requests to run the algorithm Sign_r for a session s if $r \neq \text{st}_s.\text{rnd} + 1$. The state may also include other information such as the output of previous signing rounds or the discrete log of a nonce.

Figure 1 describes an honest execution of a multi-signature scheme and provides a correctness definition.

3.2 Threshold Signatures

A threshold signature scheme allows any subset of sufficient size of a group of signers to provide succinct joint signatures for an agreed upon message. This allows the group to produce signatures even if some signers are offline, unresponsive, or adversarial. The group has an aggregated group public key for signature verification, and honestly generated signatures by any subset of the signers are valid with respect to that key.

IDEALIZED KEY GENERATION. Typically, threshold signatures require a distributed key generation protocol or a trusted dealer of secret keys. For the sake of simplicity, and in line with the framework of [6, 12], we idealize key generation, which allows for simpler security definitions. While this model is not representative of some applications, it is straightforward to extend our definitions to include distributed key generation and other security goals.

AGGREGATOR NODE. As with multi-signature, we require that in each signing round the signers broadcasts the output to all other signers. It is possible to use a single aggregator node to facilitate communication between the signers, which does not affect the unforgeability of a scheme.

FORMAL SYNTAX. A threshold signature scheme TS is a collection of algorithms TS.Kg , $(\text{TS.Sign}_r)_{r=1}^{\text{TS.nr}}$, and TS.Verify , where TS.nr is the number of signing rounds specified by the scheme. The scheme also specifies

<p>Algorithm $\text{Exec}_{\text{TS}}(\tilde{vk}, (vk_i)_{i=1}^n, S, (sk_i)_{i \in S}, t, m)$:</p> <ol style="list-style-type: none"> 1 Initialize the state of each signer $i \in S$ with $i, sk_i, (vk_j)_{j=1}^n, \tilde{vk}, t$ and initialize $i.\text{st}_1$ with S, m 2 $\text{out} \leftarrow (0)_{i \in S}$ // output of current round 3 For $r = 1, \dots, \text{TS.nr}$ do: 4 For $i \in S$ do: 5 $(\sigma_i, i.\text{st}) \leftarrow_{\\$} \text{TS.SignKey}_r(i.\text{st}, (vk_i)_{i=1}^n, S, m, \text{out})$ 6 $\text{out} \leftarrow (\sigma_i)_{i \in S}$ 7 Return σ_i for some $i \in S$ <hr/> <p>Game $\mathbf{G}_{n,t,m}^{\text{ts-corr}}[\text{TS}]$</p> <p>FIN:</p> <ol style="list-style-type: none"> 1 $((sk_i, vk_i)_{i=1}^n, \tilde{vk}) \leftarrow_{\\$} \text{TS.Kg}(n, t)$ 2 $S \leftarrow_{\\$} \{R \in \mathcal{P}(\{1, \dots, n\}) : R \geq t\}$ // \mathcal{P} denotes the power set 3 $\sigma \leftarrow_{\\$} \text{Exec}_{\text{TS}}(\tilde{vk}, (vk_i)_{i=1}^n, S, (sk_i)_{i \in S}, t, m)$ 4 Return $\text{TS.Verify}(\tilde{vk}, m, \sigma)$
--

Fig. 2. Top: an honest execution of the signing protocol of a threshold signature scheme TS. Note that signing rounds may only use a subset of the provided input. **Bottom:** a game defining the correctness of a scheme. A scheme satisfies perfect correctness for a group size n and a threshold t if $\Pr[\mathbf{G}_{n,t,m}^{\text{ts-corr}}[\text{TS}]] = 1$ for each message m that is supported by TS.

the last interactive signing round TS.lir , after which it is possible to construct a threshold-signature without knowledge of the signers secret information. The intent of the algorithms is as follows:

Key generation: We use idealized key generation, where the randomized algorithm TS.Kg takes the number of signers n and the threshold t as input and outputs the keys for each signer $(sk_i, vk_i)_{i=1}^n$ and an aggregate verification key \tilde{vk} .

Signing: The collection of algorithms $(\text{TS.SignKey}_r)_{r=1}^{\text{TS.nr}}$ specifies the signing procedure to be run by each participating signing party, and TS.nr (the number of rounds) is specified by the scheme. Each round takes a subset of the following as input: a message, a vector of public keys who participate in the session, the output of previous signing rounds, and other information saved in the state of at most one signer (including the secret key). The algorithm produces an output, as well as updates the state of the signing party. The threshold signature is the output of the last round $\text{Sign}_{\text{TS.nr}}$. These algorithms may be randomized.

Verification: The verification algorithm takes an aggregated public key \tilde{vk} , a message, and a threshold signature as an input and returns a boolean value signifying whether the signature is valid.

As in the case of multi-signatures, signers have a state st which is updated through the protocol. For each signer i the state $i.\text{st}$ includes the fields $\text{st}.n$ and $\text{st}.t$ denoting the number of signers in the group and the required threshold for signatures, $\text{st}.vk$ and $\text{st}.sk$ denoting the signers own public and secret keys, $(\text{st}.vk_j)_{j=1}^n$ denoting the public keys of all signers in the group as well as the group verification key $\text{st}.\tilde{vk}$, and $\text{st}.me$ denoting the index of signer i in the signing group. Additionally, signers hold information for each session s that they participate in, which includes $\text{st}_s.m$ (the message being signed) and $\text{st}_s.\text{rnd}$ (the most recent signing round they completed). It is assumed and required that signers refuse requests to run the algorithm Sign_r for a session s if $r \neq \text{st}_s.\text{rnd} + 1$.

Figure 2 describes an honest execution of a threshold signature scheme and provides a correctness definition.

4 Existential and Strong Unforgeability

Figure 3 defines existential and strong unforgeability for plain (single signer) digital signature schemes.

In both security games, the adversary is given an input public key vk and attempts to forge a signature σ for a message m of their choice that is valid for the said key. The adversary also has access to a signing

Games	$\mathbf{G}^{\text{suf-cma}}[\Sigma]$, $\mathbf{G}^{\text{euf-cma}}[\Sigma]$
INIT():	FIN(m, σ):
1 $(vk, sk) \leftarrow_s \Sigma.\text{Kg}()$	7 If $\Sigma.\text{Verify}(vk, m, \sigma)$:
2 $Q \leftarrow \emptyset$ // message-signature pairs obtained legitimately	8 $\boxed{\text{If } (m, \sigma) \notin Q:}$
3 Return vk	9 $\boxed{\text{If } \{(m', \sigma') \in Q : m' = m\} = \emptyset:}$
SIGNO(m):	10 Return true
4 $\sigma \leftarrow \Sigma.\text{sign}(m)$	11 Return false
5 $Q \leftarrow Q \cup \{(m, \sigma)\}$	
6 Return σ	

Fig. 3. Games used to define the existential and strong unforgeability of a single signer digital signature scheme Σ . The definition for strong unforgeability, $\mathbf{G}^{\text{suf-cma}}$, contains all but the dashed box. The definition for existential unforgeability, $\mathbf{G}^{\text{euf-cma}}$, contains all but the solid box.

oracle that they can query for signatures on adaptively chosen messages. To win, the adversary needs to output a *non-trivial* forgery. For existential unforgeability, a forgery (m, σ) is non-trivial if m was not a signing oracle query. For strong unforgeability (m, σ) is non-trivial if σ was not a signing oracle response on query m . Thus, a strongly unforgeable scheme guarantees that every signature that an adversary possesses was obtained from the signing oracle, or equivalently that the adversary cannot obtain more signatures for a message than the number of such signature produced by the signing oracle.

In this section, we extend the definition of strong unforgeability to multi- and threshold signature using one-more unforgeability.

4.1 Extending Strong Unforgeability to Multi-Signatures

Existential unforgeability of multi-signature in the plain public key model is an extension of the existential unforgeability definition for plain signatures. In the standard definition, the adversary is given a public key vk of an “honest signer” as input, and is able to query a signing oracle in which the honest signer completes the signing algorithms for messages and signing groups chosen by the adversary. If the scheme contains multiple signing rounds, then the adversary may also adaptively choose the input to each signing round as well as interweave the rounds of different signing sessions. The adversary wins if they output a non-trivial valid signature for a message m and a group of public keys $(vk_i)_{i=1}^n$ that contains vk , where non-trivial means that the signing oracle did not complete a signing session to sign m with this group of signers.

Extending strong unforgeability to multi-signature schemes is more challenging. The natural security goal is the guarantee that every valid signature was legitimately obtained, but it is unclear how to formalize this goal into a precise definition. First, an interaction with the signing oracle does not output a multi-signature but a signature share, whereas the winning condition for the adversary includes producing a valid multi-signature. Furthermore, the single signature share may not uniquely define the aggregate signature, which also depends on input from signers that are controlled by the adversary. Therefore, simply tracking the outputs of the signing oracle does not allow us to distinguish between trivial and non-trivial forgeries.

Thus, we turn to a different approach to defining strong unforgeability. For plain signatures, strong unforgeability is equivalent to the guarantee that an adversary cannot obtain more valid signatures for each message than the number of signatures obtained legitimately via the signing oracle. This notion does apply to multi-signatures, and can be formalized using *one-more unforgeability*. In our strong unforgeability definition, we count how many signature shares the adversary obtains from the signing oracles for each message and signing group, and require that the adversary cannot compute more valid signatures. Thus, a secure scheme guarantees that the adversary cannot obtain more signatures than those that can be computed trivially from the shares it obtained from the signing oracles.

We put this notion into a game definition in Figure 4, which compares it with the definition of existential unforgeability. Note that the only difference between the existential and strong unforgeability games is the

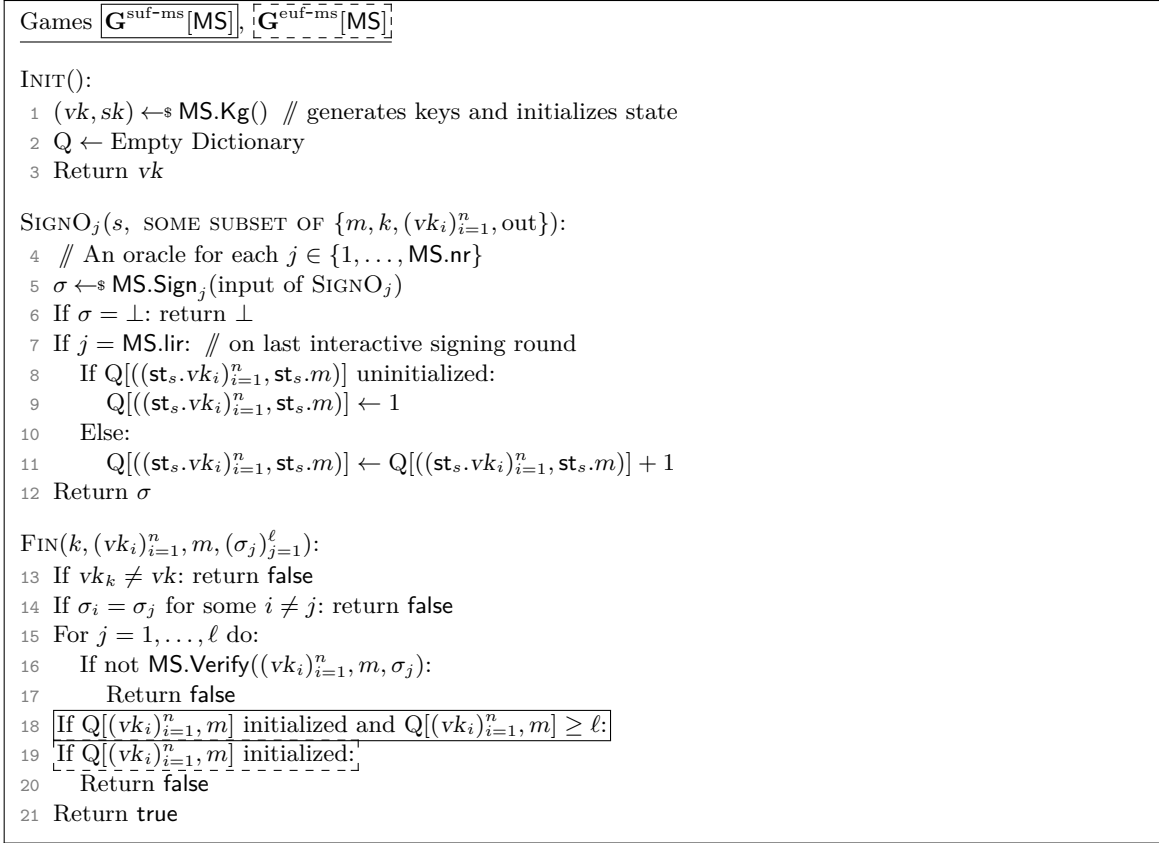


Fig. 4. Games used to define the existential and strong unforgeability of a multi-signature scheme MS. The definition for strong unforgeability, $\mathbf{G}^{\text{suf-ms}}$, contains all but the dashed box. The definition for existential unforgeability, $\mathbf{G}^{\text{euf-ms}}$, contains all but the solid box.

winning condition, and whenever an adversary wins the existential unforgeability game it also wins the strong unforgeability game. Hence, strong unforgeability implies existential unforgeability, as expected.

AT WHICH ROUND IS A MESSAGE SIGNED. Some authors (for example [9, 37, 14, 7]) consider a forgery for a message and a group of signers trivial if the adversary initiated a signing session with those parameters. However, for multi-round schemes, an adversary should not be able to obtain a signature unless all interactive signing rounds have been completed.

Thus, in our syntax a scheme specifies its last interactive signing round, MS.lir. It is expected that after querying the signing oracle for the last interactive round the adversary can produce a multi-signature, but not before, for both existential and strong unforgeability. Therefore, our security game registers that a legitimate multi-signature has been provided only on calls to the signing oracle for the last interactive round.

WHICH SIGNING ROUNDS ARE MESSAGE AND GROUP DEPENDENT. In some multi-signature schemes, some signing rounds can be completed before the message to sign or the identities of the signers in the group are determined (for example, [42, 49]). Our definitions support such schemes by allowing each scheme to define the input for each signing round in our syntax and security definitions.

TOY STRONGLY UNFORGEABLE SCHEME. We present a toy multi-signature scheme in Appendix A to help demonstrate the difference and separation between existential and strong unforgeability.

COMPARISON TO OMUF OF BLIND SIGNATURES. While we refer to our security notion as one-more unforgeability, it is more similar to the so called *strong one-more unforgeability* of blind signatures. Whereas existential one-more unforgeability of blind signatures typically means that an adversary cannot come up

with $\ell + 1$ signatures for distinct messages after completing ℓ signing sessions, strong OMUF does not require the messages to be distinct.

Our approach to defining OMUF and strong OMUF of blind signatures are similar. On one hand, if the number of message-signature pairs exceeds the number of signing sessions, there must be one message that has been signed fewer times than the number of signatures the adversary produced for that message, breaking our definition of OMUF. Conversely, if for some message the adversary can produce more signatures than the number of sessions that signed it, we can extend this to an attack producing more message-signature pairs than the number of signing executions.

4.2 Strong Unforgeability of Threshold Signatures

As with multi-signature, we define strong unforgeability for threshold signatures using one-more unforgeability.

One of the challenges with threshold signatures is the abundance of different security definitions, even for existential unforgeability. In this paper, we only formally define strong unforgeability that corresponds to the simple TS-UF-0 unforgeability definition of [6, 12]. We also discuss stronger definitions of existential unforgeability, and how to extend them to strong unforgeability.

EXISTENTIAL AND STRONG UNFORGEABILITY. A t -out-of- n threshold signature scheme with n signers and a signing threshold t can only provide security as long as less than t of the signers are corrupted, since otherwise the corrupt signers can produce signatures by following the protocol. Thus, we enforce that less than t of the signers are corrupt in all security games. The adversary can query the honest signers for signature shares via a signing oracle, and may also adaptively choose the input to each signing round and interweave the rounds of different signing sessions. The adversary wins if they obtain a non-trivial valid signature for the group public key.

Existential unforgeability (TS-UF-0) considers signatures for a message m trivial if the adversary has obtained some signature share for m via the signing oracle, or more specifically if all signing rounds of the signing protocol for m were completed by a signing oracle, up to the last interactive round (which is defined by the scheme). As with multi-signatures, extending this notion to strong unforgeability calls for one-more unforgeability. The strong unforgeability game counts how many signatures shares the adversary obtains from the signing oracle for each message, and the adversary wins if they can produce more signature than that for some message. This corresponds to the adversary obtaining more threshold signatures than the number they can trivially obtain using the signing oracle signature shares, had the adversary corrupted $t - 1$ signers.

Figure 5 provides our game definition, compared with the definition of existential unforgeability. Note that as with multi-signature, whenever an adversary wins the existential unforgeability game they also win the strong unforgeability game, and therefore strong unforgeability implies existential unforgeability.

OTHER CONSIDERATIONS. As with multi-signatures, our definition supports schemes where the message to sign is selected after the first signing round by allowing the scheme to define which input is taken by each signing round. Our definition also considers a message signed by the signing oracle if it completes the last interactive signing round, as opposed to the first round as is done in some prior work ([20], for example).

STRONGER SECURITY GOALS. It is natural to seek stronger security goals for threshold signatures for the case where the adversary corrupts less than $t - 1$ signers. In particular, a forged signature may not be considered trivial whenever the adversary obtained a signature share from a single honest signer for this message; instead, a forgery should only be considered trivial if the adversary obtained a partial signature from $t - |CS|$ signers (where $|CS|$ is the number of signers corrupted by the adversary). For existential unforgeability, this is precisely the distinction between TS-UF-0 and the stronger TS-UF-1 in [6, 12].

Using one-more unforgeability, we can extend US-UF-1 to strong unforgeability. To do this, we require that $t - |CS|$ signature shares from distinct signers must be used to construct each threshold signature, and that no partial signature can be used to construct two threshold signatures. In other words, a scheme that satisfies this security goal would guarantee that the maximum number of signatures the adversary can obtain for a message m is no more than the maximum number of threshold signatures the adversary could have constructed if each threshold signature used $t - |CS|$ signature shares obtained via the signing oracle from distinct signers and no signature share is used twice.

Some constructions (for example [1, 34, 20, 3]) seek to provide security against adversaries who adaptively choose which signers to corrupt. In the corresponding security definition, the adversary does not have to input the set of corrupted servers in advance, but can at any point choose to corrupt a signer and obtain their private keys and state. These definitions can easily be generalized to strong unforgeability using one-more unforgeability.

Lastly, it is straightforward to generalize definitions that include a concrete distributed key generation protocol ([17], for example) to strong unforgeability using one more unforgeability by changing the winning condition.

PREVIOUS STRONG UNFORGEABILITY DEFINITIONS. In [6, 12], Bellare et al. study the strong unforgeability of threshold signature schemes that are semi non-interactive, meaning they have a single signing round that requires signers to know the message and signing subset, and for which a signature is uniquely defined by the input to the last interactive signing round (called the leader request). While the class appears limited, it contains the scheme FROST and is thus of practical interest.

They present multiple strong unforgeability definitions, the weakest of which (TS-SUF-2) considers a forgery trivial only if there exist a leader request for which $t - |CS|$ honest signers replied with a signature share (where $|CS|$ denotes the number of corrupted signers). This definition is strictly stronger than our definition, and any scheme that satisfies TS-SUF-2 satisfies our strong unforgeability definition, as we show in Section 6.1 (where we also present their definition in detail). In the same section, we also present a toy scheme that satisfies our strong unforgeability definition but not TS-SUF-2, showing that the definitions are not equivalent.

Concretely, the fact that TS-SUF-2 implies our strong unforgeability definition means that the results in [6, 12] prove that FROST 1 [32] and FROST 2 [19] are strongly unforgeable by our definition (assuming idealized key generation).

5 Multi-Signature Schemes

5.1 Analysis of HBMS

In [7] Bellare and Dai present HBMS (“Hash-Based Multi-Signature”), a two round multi-signature scheme, and prove its existential unforgeability using the discrete log assumption in the random oracle model. We will show that HBMS does not satisfy our definition of strong unforgeability by providing a concrete polynomial time attack by an adversary who corrupts at least one signer and can participate in concurrent signing sessions. The attack uses the algorithm of Benhamouda et al. [13] to solve the ROS problem [48], which broke the unforgeability of many multi-signature schemes including an older variant of MuSig [36].

In the attack the adversary completes the first signing round of ℓ concurrent signing sessions for some $\ell \geq \lceil \log_2(p) \rceil$, where each session has the same group of signers and the same message and p is the order of the underlying group. Then, the adversary completes the signing sessions to obtain one multi-signature from each, and uses the output of those sessions to construct an additional signature for the same message. Thus, the adversary obtains $\ell + 1$ multi-signatures, of which ℓ are obtained legitimately and one is a forgery, breaking strong unforgeability.

The attack is practical against a group who produces signatures together repeatedly, and it can be carried out by a single malicious signer regardless of the number of signers in the group. We emphasize, however, that it does not compromise the existential unforgeability of HBMS nor violate existing security proofs.

We point out that HBMS is strongly unforgeable against adversaries that don’t exploit the fact that it is an interactive multi-signature scheme (i.e. if we assume an atomic execution of the signing protocol and no corrupt signers). In other words, HBMS produces strongly unforgeable plain signatures in a distributed way, without being strongly unforgeable itself. In Appendix B we formally define this weaker security notion (which we call non-interactive strong unforgeability), and we prove that it is satisfied by HBMS in Appendix B.1.

THE HBMS SCHEME. We describe the scheme informally. A formal description of the scheme is given in Figure 9 of [7].

HBMS involves three hash functions: H_0 with codomain \mathbb{G} and H_1, H_2 with codomain \mathbb{Z}_p , where \mathbb{G} is a multiplicative group of order p with a generator g provided by the scheme parameters. For key generation,

Games	$\mathbf{G}_{n,t}^{\text{suf-ts}}[\text{TS}]$, $\mathbf{G}_{n,t}^{\text{euf-ts}}[\text{TS}]$
INIT(CS):	<ol style="list-style-type: none"> 1 Require $CS \subseteq \{1, \dots, n\}$ and $CS < t$: // corrupt set 2 $HS \leftarrow \{1, \dots, n\} \setminus CS$ // honest parties 3 $Q \leftarrow$ Empty Dictionary // tracks legit signatures 4 $((vk_i, sk_i)_{i=1}^n, \tilde{vk}) \leftarrow_s \text{TS.Kg}(n, t)$ // generates keys and initializes state 5 Return $\tilde{vk}, (vk_i)_{i=1}^n, (sk_j)_{j \in CS}$
SIGN $O_j(k, s, \text{SOME SUBSET OF } \{m, S, \text{out}\})$:	<ol style="list-style-type: none"> 6 // An oracle for each $j \in \{1, \dots, \text{TS.nr}\}$ 7 If $k \notin HS$: return \perp 8 $\sigma \leftarrow_s k.\text{TS.Sign}_j(\text{input to SIGN}O_j)$ 9 If $\sigma = \perp$: return \perp 10 If $j = \text{TS.lir}$: // on last interactive signing round 11 If $Q[\text{st}_s.m]$ uninitialized: $Q[\text{st}_s.m] \leftarrow 1$ 12 Else: $Q[\text{st}_s.m] \leftarrow Q[\text{st}_s.m] + 1$ 13 Return σ
FIN($m, (\sigma_j)_{j=1}^\ell$):	<ol style="list-style-type: none"> 14 If $\sigma_i = \sigma_j$ for some $i \neq j$: return false 15 For $j = 1, \dots, \ell$ do: 16 If not $\text{TS.Verify}(\tilde{vk}, m, \sigma_j)$: 17 Return false 18 If $Q[m]$ initialized and $Q[m] \geq \ell$: 19 If $Q[m]$ initialized: 20 Return false 21 Return true

Fig. 5. Game used to define the existential and strong unforgeability of a threshold-signature scheme TS with a threshold t out of n . The definition for strong unforgeability, $\mathbf{G}_{n,t}^{\text{suf-ts}}$, contains all but the dashed box. The definition for existential unforgeability, $\mathbf{G}_{n,t}^{\text{euf-ts}}$, contains all but the solid box.

each signer i of the n signers samples a secret key x_i uniformly at random from \mathbb{Z}_p and a public key $X_i \leftarrow g^{x_i}$. The aggregate key is $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i, X_1, \dots, X_n)}$.

To sign a message m with a group of signers $(X_i)_{i=1}^n$, the scheme involves two interactive signing rounds. In the first round, given the message and signing group as input, each signer i calculates $h \leftarrow H_0(X_1, \dots, X_n, m) \in \mathbb{G}$, samples r_i and s_i uniformly at random from \mathbb{Z}_p , and computes a commitment $M_i \leftarrow h^{s_i} g^{r_i}$ which is sent to every other signer. In the second round, each signer receives a list of commitments (M_1, \dots, M_n) from all the signers and computes $T \leftarrow \prod_{i=1}^n M_i$. Each signer then computes the challenge $c \leftarrow H_1(T, \tilde{X}, m)$ and the reply $z_i \leftarrow r_i + x_i \cdot c \cdot H_2(i, X_1, \dots, X_n)$, and sends (s_i, z_i) to every other signer. Finally, every signer can now compute the final signature (T, s, z) where $s \leftarrow \sum_{i=1}^n s_i$, $z \leftarrow \sum_{i=1}^n z_i$, and $T \leftarrow \prod_{i=1}^n M_i$.

To verify a signature (T, s, z) with respect to public keys (X_1, \dots, X_n) and a message m , the verifier computes $h \leftarrow H_0(X_1, \dots, X_n, m)$ and $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i, X_1, \dots, X_n)}$, and returns true if and only if the equation

$$g^z h^s = T \cdot \tilde{X}^{H_1(T, \tilde{X}, m)}$$

holds. Note that during verification the entire vector of public keys is needed for computing h , and hence HBMS does not support key aggregation. Perfect correctness is easy to verify, and [7] proves the existential unforgeability of HBMS.

THE ATTACK. We will present an attack in the two signers setting where one signer is corrupt, which is sufficient to break our definition of strong unforgeability. It is easy to generalize it to a setting with more signers, as long as at least one signer is corrupt.

Let S_1 be a corrupt signer controlled by the adversary and S_2 an honest signer (with whom the adversary can communicate via a signing oracle). Let m be a message of the adversary's choice and pick $\ell \geq \lceil \log_2(p) \rceil$. Each signer $S_i \in \{S_1, S_2\}$ proceeds with the key generation honestly by picking $x_i \leftarrow \mathbb{Z}_p$ and $X_i \leftarrow g^{x_i}$ and computing $\tilde{X} \leftarrow X_1^{H_2(1, X_1, X_2)} X_2^{H_2(2, X_1, X_2)}$.

Now, for each $j \in \{1, \dots, \ell\}$, the adversary opens a signing session with signing group (X_1, X_2) and message m , and receive a nonce $N_j = h^{s_j} g^{r_j}$ from the honest signer S_2 , where $h \leftarrow H_0(X_1, X_2, m)$. For each $j \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$, the adversary samples \bar{r}_j^b and \bar{s}_j^b uniformly at random from \mathbb{Z}_p and computes $\bar{N}_j^b \leftarrow h^{\bar{s}_j^b} g^{\bar{r}_j^b}$ and $\bar{T}_j^b \leftarrow N_j \cdot \bar{N}_j^b$ as well as $\bar{c}_j^b \leftarrow H_1(\bar{T}_j^b, \tilde{X}, m)$. The adversary must also ensure that all of the \bar{T}_j^b are distinct and that $\bar{c}_j^0 \neq \bar{c}_j^1$ for each j by regenerating the nonces if needed.

Now, define the group homomorphisms $\rho_+ : (\mathbb{Z}_p)^\ell \rightarrow \mathbb{Z}_p$ and $\rho_\times : \mathbb{G}^\ell \rightarrow \mathbb{G}$ given by

$$\rho_+(g_1, \dots, g_\ell) = \sum_{j=1}^{\ell} \frac{2^{j-1} g_j}{\bar{c}_j^1 - \bar{c}_j^0}$$

and

$$\rho_\times(g_1, \dots, g_\ell) = \prod_{j=1}^{\ell} g_j^{\frac{2^{j-1}}{\bar{c}_j^1 - \bar{c}_j^0}}.$$

Let $T_{\ell+1} \leftarrow \rho_\times(N_1, \dots, N_\ell)$ and calculate $c_{\ell+1} \leftarrow H_1(T_{\ell+1}, \tilde{X}, m)$. Let $d \leftarrow c_{\ell+1} - \rho_+(\bar{c}_1^0, \dots, \bar{c}_\ell^0)$ and write it in binary as $d = \sum_{j=1}^{\ell} 2^{j-1} b_j$ for some $b_1, \dots, b_\ell \in \{0, 1\}$, which is possible since $\ell \geq \lceil \log_2(p) \rceil$.

Next, continue to the second round of each signing session j by sending $\bar{N}_j^{b_j}$ to the honest signer and obtaining the returned signature shares s_j and z_j . The adversary can now obtain ℓ legitimate signatures for the message m by computing

$$\sigma_j \leftarrow (\bar{T}_j^{b_j}, s_j + \bar{s}_j^{b_j}, z_j + \bar{r}_j^{b_j} + x_1 \cdot \bar{c}_j^{b_j} \cdot H_2(1, X_1, X_2))$$

for each $j \in \{1, \dots, \ell\}$, as well as a forgery

$$\sigma_{\ell+1} \leftarrow (T_{\ell+1}, \rho_+(s_1, \dots, s_\ell), \rho_+(z_1, \dots, z_\ell) + c_{\ell+1} \cdot x_1 \cdot H_2(1, X_1, X_2)).$$

We will prove below that all $\ell + 1$ signatures $(\sigma_1, \dots, \sigma_\ell, \sigma_{\ell+1})$ are valid for the message m and signing group (X_1, X_2) , and that they are all distinct with high probability. This implies that the adversary obtained $\ell + 1$ valid signatures after only completing ℓ signing oracle signing sessions, breaking the strong unforgeability of HBMS.

VALIDITY OF $\sigma_1, \dots, \sigma_\ell$. Since all of the $\bar{T}_j^{b_j}$ are distinct, all of the σ_j are distinct for $j \in \{1, \dots, \ell\}$. Also note that each of those signatures was obtained legitimately with both signers following the protocol, and hence by the perfect correctness of HBMS they are valid.

VALIDITY OF $\sigma_{\ell+1}$. The signature $\sigma_{\ell+1} = (T_{\ell+1}, \rho_+(s_1, \dots, s_\ell), \rho_+(z_1, \dots, z_\ell) + c_{\ell+1} \cdot x_1 \cdot H_2(1, X_1, X_2))$ is the forged signature, and is the only one that is not trivial to obtain.

For the distinctiveness of $\sigma_{\ell+1}$, note that the collection $\{\bar{T}_1^{b_1}, \dots, \bar{T}_\ell^{b_\ell}\}$ is selected uniformly at random from all subsets of \mathbb{G} of cardinality ℓ , independently of (N_1, \dots, N_ℓ) . Hence, the probability that it contains $T_{\ell+1} = \rho_\times(N_1, \dots, N_\ell)$ is $\frac{\ell}{|\mathbb{G}|} \approx \frac{\log_2(p)}{p}$, which is very small. Hence, with large probability, $T_{\ell+1} \neq \bar{T}_j^{b_j}$ and therefore $\sigma_{\ell+1} \neq \sigma_j$ for all $j \in \{1, \dots, \ell\}$.

We will now verify that $\sigma_{\ell+1}$ is valid. To check its validity, we must verify that

$$g^{\rho_+(z_1, \dots, z_\ell) + c_{\ell+1} \cdot x_1 \cdot H_2(1, X_1, X_2)} \cdot h^{\rho_+(s_1, \dots, s_\ell)} = T_{\ell+1} \cdot \tilde{X}^{c_{\ell+1}}$$

where $h \leftarrow H_0(X_1, X_2, m)$. Starting from the right-hand side, we have that

$$\begin{aligned} T_{\ell+1} \cdot \tilde{X}^{c_{\ell+1}} &= \rho_\times(N_1, \dots, N_\ell) \cdot (X_1^{H_2(1, X_1, X_2)} X_2^{H_2(2, X_1, X_2)})^{c_{\ell+1}} \\ &= g^{\rho_+(r_1, \dots, r_\ell)} \cdot h^{\rho_+(s_1, \dots, s_\ell)} \cdot g^{(x_1 \cdot H_2(1, X_1, X_2) + x_2 \cdot H_2(2, X_1, X_2)) c_{\ell+1}}. \end{aligned}$$

Applying Lemma 3, which states that $c_{\ell+1} = \rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell})$, we can simplify the equation to

$$= h^{\rho_+(s_1, \dots, s_\ell)} g^{x_1 \cdot H_2(1, X_1, X_2) c_{\ell+1} + \rho_+(r_1, \dots, r_\ell) + x_2 \cdot H_2(2, X_1, X_2) \cdot \rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell})}$$

and therefore, since ρ_+ is homomorphic and $z_j = r_j + x_2 \cdot H_2(2, X_1, X_2) \cdot \bar{c}_j^{b_j}$ for each j ,

$$= h^{\rho_+(s_1, \dots, s_\ell)} g^{x_1 \cdot H_2(1, X_1, X_2) c_{\ell+1} + \rho_+(z_1, \dots, z_\ell)}$$

which is what we wanted to prove. Hence, $\sigma_{\ell+1}$ is a valid signature.

Lemma 3. *By the construction above, $c_{\ell+1} = \rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell})$.*

This lemma is at the heart of the attack, and the construction allowing this lemma to hold is precisely the algorithm of [13] to solve the ROS problem.

Proof (Lemma 3). By definition $\sum_{j=1}^{\ell} 2^{j-1} b_j = c_{\ell+1} - \rho_+(\bar{c}_1^0, \dots, \bar{c}_\ell^0)$. Hence, to prove the lemma, it is sufficient to show that

$$\rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell}) - \rho_+(\bar{c}_1^0, \dots, \bar{c}_\ell^0) = \sum_{j=1}^{\ell} 2^{j-1} b_j.$$

Starting from the left-hand side, we have that

$$\rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell}) - \rho_+(\bar{c}_1^0, \dots, \bar{c}_\ell^0) = \sum_{j=1}^{\ell} \frac{2^{j-1}(\bar{c}_j^{b_j} - \bar{c}_j^0)}{\bar{c}_j^1 - \bar{c}_j^0}.$$

For each j , we have that $\frac{2^{j-1}(\bar{c}_j^{b_j} - \bar{c}_j^0)}{\bar{c}_j^1 - \bar{c}_j^0}$ is equal to 0 whenever b_j is 0 and is equal to 2^{j-1} whenever b_j is 1.

Consequently, $\frac{2^{j-1}(\bar{c}_j^{b_j} - \bar{c}_j^0)}{\bar{c}_j^1 - \bar{c}_j^0} = 2^{j-1} b_j$. Thus,

$$\rho_+(\bar{c}_1^{b_1}, \dots, \bar{c}_\ell^{b_\ell}) - \rho_+(\bar{c}_1^0, \dots, \bar{c}_\ell^0) = \sum_{j=1}^{\ell} 2^{j-1} b_j$$

which is what we wanted to prove.

5.2 Analysis of MuSig

In this section we prove the strong unforgeability of the multi-signature scheme MuSig which consists of three interactive signing rounds and supports key aggregation, presented in [37].

We emphasize that only the 3-round version of MuSig is strongly unforgeable, whereas the prior two-round version [36] is insecure [24, 13].

THE SCHEME. We now describe the scheme informally. A formal description of the scheme using our syntax for multi-signatures can be found in Figure 6.

The scheme involves a group \mathbb{G} of prime order p with a generator g and the hash functions H_{com} , H_{sign} , and H_{agg} with codomain \mathbb{Z}_p that are used for commitments, signing, and key aggregation respectively. In key generation, each signing party generates a private key $sk \leftarrow_s \mathbb{Z}_p$ and a public key $vk \leftarrow g^{sk}$. The aggregate public key for a group of n signers with public keys vk_1, \dots, vk_n is computed by

$$\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{H_{\text{agg}}(i, vk_1, \dots, vk_n)}.$$

In the first signing rounds, each signer k chooses $r_k \leftarrow_s \mathbb{Z}_p$, computes $R_k \leftarrow g^{r_k}$, and sends a commitment $t_k \leftarrow H_{\text{com}}(R_k)$ to all the other signers. In the second round, each signer k receives the commitments t_1, \dots, t_n from all other signers, and sends R_k to all other signers. In the third round, the signer receives nonces R_1, \dots, R_n from all the signers and verifies the commitments by checking that $t_i = H_{\text{com}}(R_i)$ for each i . Then, they compute $R \leftarrow \prod_{i=1}^n R_i$, the aggregate public key \tilde{vk} as described above, and a challenge $c \leftarrow H_{\text{sign}}(\tilde{vk}, R, m)$. Then, they output a signature share $z_k \leftarrow r_k + sk_k \cdot c \cdot H_{\text{agg}}(k, vk_1, \dots, vk_n)$. Now, any of the signer can output the multi-signature (R, z) where $z \leftarrow \sum_{i=1}^n z_i$.

A signature (R, z) is valid with respect to an aggregated verification key \tilde{vk} and a message m if and only if

$$g^z = R \cdot \tilde{vk}^{H_{\text{sign}}(\tilde{vk}, R, m)}.$$

MuSig satisfies perfect correctness, and the verification of a MuSig multi-signature with respect to an aggregated key \tilde{vk} is identical to the verification of a standard Schnorr signature.

WHICH SIGNING ROUNDS ARE MESSAGE DEPENDENT. The signers in MuSig do not use the message in the first two signing rounds. Thus, it is natural to ask whether it is possible to pre-execute the first two signing rounds before the message to sign arrives. If so, the scheme would involve a single interactive signing round when the message arrives, resulting in an almost non-interactive signature scheme (this property is claimed

<p>Scheme $\text{MuSig}_{\mathbb{G}, g, H_{\text{agg}}, H_{\text{com}}, H_{\text{sign}}}$:</p> <p>$\text{MuSig.nr} = 4$ $\text{MuSig.lir} = 3$</p> <p>KeyGen():</p> <ol style="list-style-type: none"> 1 $x \leftarrow_{\mathcal{S}} \mathbb{Z}_p; X \leftarrow g^x$ 2 $\text{st.sk} \leftarrow x; \text{st.vk} \leftarrow X$ 3 Return $(\text{sk} = x, \text{vk} = X)$ <p>KeyAgg($\text{vk}_1, \dots, \text{vk}_n$):</p> <ol style="list-style-type: none"> 4 Return $\prod_{i=1}^n \text{vk}_i^{H_{\text{agg}}(i, \text{vk}_1, \dots, \text{vk}_n)}$ <p>AggVer($m, \sigma, \tilde{\text{vk}}$):</p> <ol style="list-style-type: none"> 5 $(R, z) \leftarrow \sigma$ 6 Return $[g^z = R \cdot \tilde{\text{vk}}^{H_{\text{sign}}(\tilde{\text{vk}}, R, m)}]$ <p>Sign₁():</p> <ol style="list-style-type: none"> 7 $\text{st.j} \leftarrow \text{st.j} + 1; j \leftarrow \text{st.j}$ 8 $\text{st.j.r} \leftarrow_{\mathcal{S}} \mathbb{Z}_p; \text{st.j.R} \leftarrow g^{\text{st.j.r}}$ 9 $\text{st.j.t} \leftarrow H_{\text{com}}(\text{st.j.R})$ 10 $\text{st.j.rnd} \leftarrow 1$ 11 Return st.j.t 	<p>Sign₂($j, (t_i, \text{vk}_i)_{i=1}^n, m$):</p> <ol style="list-style-type: none"> 12 If $\text{st.j.rnd} \neq 1, \text{vk}_k \neq \text{st.vk}$, or $\text{st.j.t} \neq t_k$: 13 Return \perp 14 $\text{st.j.m} \leftarrow m; \text{st.j.n} \leftarrow n; \text{st.j.k} \leftarrow k$ 15 For i from 1 to n: 16 $\text{st.j.t}_i \leftarrow t_i; \text{st.j.vk}_i \leftarrow \text{vk}_i$ 17 $\text{st.rnd} \leftarrow 2$ 18 Return st.j.R <p>Sign₃(j, R_1, \dots, R_n):</p> <ol style="list-style-type: none"> 19 If $\text{st.j.rnd} \neq 2$ or $R_{\text{st.j.k}} \neq \text{st.j.R}$: 20 Return \perp 21 If $\exists i$ such that $\text{st.j.t}_i \neq H_{\text{com}}(R_i)$: 22 Return \perp 23 $\tilde{\text{vk}} \leftarrow \text{KeyAgg}(\text{st.j.vk}_1, \dots, \text{st.j.vk}_n)$ 24 $R \leftarrow \prod_{i=1}^n R_i$ 25 $c \leftarrow H_{\text{sign}}(R, \tilde{\text{vk}}, \text{st.j.m})$ 26 $a \leftarrow H_{\text{agg}}(\text{st.j.k}, \text{st.j.vk}_1, \dots, \text{st.j.vk}_n)$ 27 $z \leftarrow \text{st.j.r} + x \cdot c \cdot a$ 28 $\text{st.j.rnd} \leftarrow 3$ 29 Return (R, z) <p>Sign₄(R, z_1, \dots, z_n):</p> <ol style="list-style-type: none"> 30 Return $(R, \sum_{i=1}^n z_i)$
---	--

Fig. 6. A description of the MuSig scheme over a group \mathbb{G} of order p and generator g . The fourth round is often omitted since it can be performed by any observer of the protocol.

by MuSig2 [42], for example). The original MuSig paper [37] does not provide an explicit answer to this question.

The answer, however, is no. Such a shortcut leads to the scheme no longer being existentially unforgeable [40, 39]. For security, the signers must associate each signing execution with a message and a signing group when executing the second signing round (the “reveal” round of the nonce shares). Our security proof only applies in this setting.

PRIOR SECURITY PROOFS FOR MUsIG. The existential unforgeability of MuSig is proved in [37, 14, 7]. These proofs, however, use a security definition that considers a forgery trivial whenever the adversary opened a signing oracle signing session with the corresponding message and group of signers, regardless of whether the signing session was completed. Consequently, these proofs do not rule out adversaries who complete the first two signing rounds for some message and then forge a signature without completing the third signing round. This is problematic since the third signing round is where signers verify the commitments sent in previous rounds, and it is the only round where the signers use their private keys.

We fill this gap by providing a security proof of strong unforgeability, and consequently existential unforgeability, using a definition that considers a forgery trivial only if the honest signer has completed all interactive rounds of a signing session with the corresponding message and public keys. This stronger definition comes at the cost of a looser reduction than the reduction in [7] by a factor of approximately q_s , where q_s denotes the maximum number of signing sessions opened by the adversary.

OUR RESULT, CHAIN REDUCTIONS, AND THE XIDL. In [7], Bellare and Dai construct a chain of reductions from the discrete log problem to their definition of the existential unforgeability of MuSig. One of the links in the chain is the Random Target Identification Logarithm (XIDL) game in Figure 7, and they show that it is hard whenever the discrete log assumption (Figure 7) holds, as written in Lemma 4.²

² They also achieve tighter security bounds using the algebraic group model [25], but this is orthogonal to this paper.

Game $\text{Gm}_{\mathbb{G},g}^{\text{dl}}$	Game $\text{Gm}_{\mathbb{G},g,q_1,q_2}^{\text{xidl}}$
INIT(): 1 $x \leftarrow_{\$} \mathbb{Z}_p; X \leftarrow g^x$ 2 Return X FIN(x'): 3 Return $[x = x']$	INIT(): 1 $x \leftarrow_{\$} \mathbb{Z}_p; X \leftarrow g^x$ 2 $j \leftarrow 0; i \leftarrow 0$ 3 Return X NEWTARGET(S): // at most q_1 queries. 4 $j = j + 1; S_j \leftarrow S$ 5 $e_j \leftarrow_{\$} \mathbb{Z}_p; T_j \leftarrow S_j \cdot X^{e_j}$ 6 Return e_j CHALLENGE(j_{sel}, R): // at most q_2 queries 7 $i \leftarrow i + 1; R_i \leftarrow R$ 8 $Y_i \leftarrow T_{j_{\text{sel}}}; c_i \leftarrow_{\$} \mathbb{Z}_p$ 9 Return c_i FIN(I, z): 10 Return $[g^z = R_I \cdot Y_I^{c_I}]$

Fig. 7. The Discrete Log (DL) and the Random Target Identification Logarithm (XIDL) games in a group \mathbb{G} with a generator g of prime order p .

In Lemma 5, we prove that MuSig is strongly unforgeable in the random oracle model if winning the XIDL is hard. The combination of these lemmas proves the strong unforgeability of MuSig in the ROM under the discrete log assumption.

Lemma 4 (DL \rightarrow XIDL; a combination of Theorems 3.2 and 3.4 of [7]). *Let \mathbb{G} be a group of order p with generator g . Let q_1, q_2 be positive integers. Let $\mathcal{A}_{\text{xidl}}$ be an adversary against $\text{Gm}_{\mathbb{G},g,q_1,q_2}^{\text{xidl}}$. Then, an adversary \mathcal{A}_{dl} can be constructed so that*

$$\text{Adv}_{\mathbb{G},g,q_1,q_2}^{\text{xidl}}(\mathcal{A}_{\text{xidl}}) \leq \sqrt{q_2(\sqrt{q_1 \cdot \text{Adv}_{\mathbb{G},g}^{\text{dl}}(\mathcal{A}_{\text{dl}})} + \frac{q_1}{p})} + \frac{q_2}{p}$$

and the running time of \mathcal{A}_{dl} is approximately four times the running time of $\mathcal{A}_{\text{xidl}}$.

We omit the proofs of Lemma 4 since it is non-trivial and can be found in the referenced paper.

Lemma 5 (XIDL \rightarrow SUF of MuSig in the ROM). *Let \mathbb{G} be a group of prime order p . Let g be a generator of \mathbb{G} . Let $\text{MS} = \text{MuSig}[\mathbb{G}, g]$ be the associated multi-signature scheme, with its hash functions modeled as random oracles. Let \mathcal{A}_{ms} be an adversary for the game $\mathbf{G}^{\text{suf-ms}}[\text{MS}]$ and assume the execution of \mathcal{A}_{ms} has at most q_0, q_1, q_2, q_s distinct queries to $H_{\text{com}}, H_{\text{agg}}, H_{\text{sign}}$, and SIGNO_1 , the number of signing parties in queries to signing oracle queries and FIN is at most n , and the number of signatures it outputs is at most ℓ . Let $q = q_0(q_0 + n \cdot q_s) + (q_s + q_1 + 1)^2 + q_2(q_s + q_1 + 1) + q_s(q_2 + q_s) + n \cdot q_s(q_0 + n \cdot q_s)$. Then, there exists an adversary $\mathcal{A}_{\text{xidl}}$ for the game $\text{Gm}_{\mathbb{G},g,q_1+q_s+1,q_2+\ell}^{\text{xidl}}$ such that*

$$\text{Adv}_{\text{MS}}^{\text{suf-ms}}(\mathcal{A}_{\text{ms}}) \leq (1 + q_s) \text{Adv}_{\mathbb{G},g,q_s+q_1+1,q_2+\ell}^{\text{xidl}}(\mathcal{A}_{\text{xidl}}) + \frac{q}{p}$$

and the running time of $\mathcal{A}_{\text{xidl}}$ is similar to that of \mathcal{A}_{ms} .

We now describe the proof idea for Lemma 5, and include a formal proof in Appendix C.

Proof Idea for Lemma 5. We describe informally how to win the XIDL game with high probability given an adversary that breaks the strong unforgeability of MuSig in the ROM.

SESSION PARAMETERS AND SIGNATURE TYPES. Each execution of the third signing round of the signing oracle uses a specific aggregate public key \tilde{vk} , aggregate nonce R , and message m . We refer this tuple (R, \tilde{vk}, m) as the session parameter of this signing session.

Now consider an adversary \mathcal{A}_{ms} in the random oracle model. If they wish to complete a signing session with the signing oracle, then for each corrupt signer they must provide a commitment t_i as input to the second

signing round and then an R_i for the third signing round satisfying $H_{\text{com}}(R_i) = t_i$. To have a non-negligible probability of completing the signing session, they must have called the random oracle H_{com} with input R_i before providing the input to the second signing round. Thus, whenever the adversary calls SIGNO_2 with input $(m, X_1, \dots, X_n, t_1, \dots, t_n)$, the reduction can recover all of the R_i and compute the session parameters.

Now suppose \mathcal{A}_{ms} breaks the strong unforgeability of MuSig. At the end of the game it outputs ℓ valid signatures $(R_j, z_j)_{j=1}^{\ell}$, where the R_j 's are all distinct, for some message m and a group of keys X_1, \dots, X_n with aggregated key \tilde{vk} . Each of these signatures must fall into one of the following cases:

- Case 1:** (R_j, \tilde{vk}, m) was the session parameters for some signing oracle signing session that executed the third signing round.
- Case 2:** (R_j, \tilde{vk}, m) was the session parameters for some signing oracle signing session that executed the second signing round, but not the third.
- Case 3:** (R_j, \tilde{vk}, m) was not the session parameters for any signing oracle signing session.

Since \mathcal{A}_{ms} wins the strong unforgeability game, at most $\ell - 1$ signing oracle sessions with message m and keys X_1, \dots, X_n completed the third signing round. Hence, at most $\ell - 1$ signatures fall into Case 1 and at least one signature falls into Case 2 or 3. We refer to such signatures as “forgeries.”

USING A FORGERY TO WIN XIDL. Let X denote the output of the XIDL INIT procedure, which the reduction sets to be the public key of the honest signer.

Now, suppose (R, z) is a valid multi-signature for a message m and a group of public keys vk_1, \dots, vk_n with $vk_k = X$. If the key aggregation exponent $H_{\text{agg}}(k, vk_1, \dots, vk_n)$ is an XIDL target³ and $c = H(R, \tilde{vk}, m)$ is a challenge obtained from the XIDL's CHALLENGE oracle with input R , then (R, z) wins the XIDL game. Thus, we program the random oracle so that responses to $H_{\text{agg}}(k, vk_1, \dots, vk_n)$ queries are indeed XIDL targets and c is an XIDL challenge corresponding to that target. It remains to show how to simulate the signing oracle so that we can program the random oracle in such a way.

HOW TO SIMULATE THE SIGNING ORACLE. In order to construct a reduction, we must simulate the signing oracle without knowing the secret key of the honest signer. We use the standard technique of simulating the Schnorr signing oracle without knowledge of the public key.

- 1st round:** The reduction simply outputs a random commitment $t_k \leftarrow_{\$} \mathbb{Z}_p$.
- 2nd round:** The reduction picks a uniformly random signature share and challenge $z_k, c \leftarrow_{\$} \mathbb{Z}_p$, chooses a nonce-share $R_k \leftarrow g^{z_k} X^{-c \cdot H_{\text{agg}}(k, vk_1, \dots, vk_n)}$, and sets $H_{\text{com}}(R_k) \leftarrow t_k$ so the commitment from the first round holds. As explained before, the reduction can now recover the session parameters (R, \tilde{X}, m) even though R is not yet known to the adversary. Therefore, it programs the random oracle $H_{\text{sign}}(R, \tilde{vk}, m) \leftarrow c$, and outputs the nonce share R_k .
- 3rd round:** The reduction outputs the partial signature z_k that it generated when simulating the second signing round. It is a valid signature share by construction.

GUESSING WHICH SESSION PARAMETERS ARE FOR FORGERY. When simulating the signing oracle for a session with parameters (R, \tilde{vk}, m) , we program the random oracle challenge $H_{\text{sign}}(R, \tilde{vk}, m) \leftarrow c$ for a c that we selected before determining R . Therefore, the reduction cannot use that challenge to win the XIDL game. This is why it needs a forgery.

Suppose the forgery is (R, z) and that is valid for a message m and an aggregated key \tilde{X} . If the forgery falls into Case 3 (the cases are defined at the beginning of the proof idea) we can use it to win the XIDL game, since its session parameters (R, \tilde{X}, m) were not used by the signing oracle and thus the corresponding challenge is an XIDL challenge. However, if the forgery falls into Case 2, then the reduction programmed $H_{\text{sign}}(R, \tilde{vk}, m) \leftarrow c$ when simulating SIGNO_2 and thus we cannot use it directly to win the XIDL game.⁴

To win the XIDL game using this type of forgery, the reduction generates an integer ρ uniformly at random from $\{1, \dots, q_s, q_s + 1\}$ where q_s refers to the maximum number of signing sessions that can be opened by

³ The above procedure works for the case where there exist a unique k such that $vk_k = X$. If more than one such k exists, then we can program the random oracle so that $\sum_{\{k: X_k = X\}} H(k, vk_i, \dots, vk_n)$ is an XIDL target and the hash function values appear uniformly random. See the formal proof for more details.

⁴ Previous MuSig security proofs do not consider signatures of Case 2 as forgeries, since they consider a forgery trivial whenever the adversary initiated a signing session with the signing oracle for the corresponding message.

<p>Game $\text{Gm}_{\mathbb{G},g}^{\text{aomdl}}$</p> <p>INIT():</p> <p>1 $c \leftarrow 0; q \leftarrow 0$</p> <p>CHALLENGE():</p> <p>2 $c \leftarrow c + 1$</p> <p>3 $x_c \leftarrow \mathbb{Z}_p; X_c \leftarrow g^{x_c}$</p> <p>4 Return X_c</p>	<p>$\text{DLOG}(\alpha, \beta_1, \dots, \beta_c)$:</p> <p>5 // Outputs the DLog of $X = g^\alpha \prod_{i=1}^c X_i^{\beta_i}$ given its algebraic representation in terms of the challenges</p> <p>6 $q \leftarrow q + 1$</p> <p>7 Return $\alpha + \sum_{i=1}^c \beta_i x_i$</p> <p>FIN($y_1, \dots, y_c$):</p> <p>8 If $q \geq c$ or $\exists i \in \{1, \dots, n\}$ such that $y_i \neq x_i$:</p> <p>9 Return false</p> <p>10 Return true</p>
--	--

Fig. 8. The Algebraic One More Discrete Log (AOMDL) game in a group \mathbb{G} with a generator g of prime order p .

the adversary. Then, it simulates all signing sessions of index different from ρ as described above. For the ρ_{th} session, however, it runs the first two signing rounds of the signing session honestly by picking the nonce share R_k first, then generating the commitment and programming the signing oracle $t_k \leftarrow H_{\text{com}}(R_k) \leftarrow \mathbb{Z}_p$. Once the session parameters (R, \tilde{vk}, m) are known at the initiation of the second round we can program the random oracle $H_{\text{sign}}(R, \tilde{vk}, m) \leftarrow c$ where c is an XIDL challenge. Note that the reduction cannot simulate the third signing round of the ρ_{th} signing session and will have to abort if the adversary asks for it. It will, however, be able to win the XIDL game if the adversary outputs a forgery that falls into Case 2 with the session parameters of the ρ_{th} signing session.

Thus, if the adversary outputs a forgery that falls into Case 2, and we chose ρ so it corresponds to the session with the same parameters as the forgery, then we win the XIDL game. Hence, if the adversary produces a forgery that falls into Case 2 then we win the XIDL game with probability of at least $\frac{1}{q_s+1}$. If the adversary outputs a forgery that falls into Case 3, then we win if we were able to simulate all signing oracle queries, which is guaranteed if we chose $\rho = q_s + 1$ and thus we win with a probability of at least $\frac{1}{q_s+1}$. Since every successful adversary against the strong unforgeability of MuSig must provide a forgery that falls into Case 2 or 3, this means that if an adversary breaks the strong unforgeability of MuSig then the reduction wins the XIDL game with probability of approximately $\frac{1}{q_s+1}$.

We include a formal proof in Appendix C.

5.3 Analysis of MuSig2

We prove the strong unforgeability of the multi-signature scheme MuSig2 [42] under the Algebraic One More Discrete Log assumption of [42] which we present in Figure 8 (a weaker falsifiable variant of the One More Discrete Log assumption [8]).

MuSig2 requires only two interactive signing rounds, of which one can be pre-processed before the message to sign and the set of signers have been determined. It also supports key aggregation and produces ordinary Schnorr signatures with respect to the aggregated signing key.

Our strong unforgeability proof of MuSig2 is nearly identical to its original existential unforgeability proof [42], and we strive to use similar structure and notation when presenting the proof as well as reuse as much of it as possible. The similarity of our proof to the existential unforgeability proof serves as evidence that our definition of strong unforgeability is straightforward to use.

THE SCHEME. We will describe the scheme informally. A formal description using our syntax is found in Figure 9.

The scheme uses a group \mathbb{G} of prime order p with a generator g and three hash functions H_{agg} , H_{nonce} , and H_{sign} with codomain \mathbb{Z}_p . Key generation and aggregation is the same as in MuSig,⁵ where each signer generates the keys $sk \leftarrow \mathbb{Z}_p$ and $vk \leftarrow g^{sk}$ and the aggregate verification key for a group of n signers is $\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{H_{\text{agg}}(i, X_1, \dots, X_n)}$.

⁵ We slightly deviate from the original MuSig2 scheme by writing $\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{H_{\text{agg}}(i, vk_1, \dots, vk_n)}$, as opposed to $\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{H_{\text{agg}}(vk_i, vk_1, \dots, vk_n)}$. This follows the convention of [7], which views the public keys of the signing group as a list as opposed to a multi-set in the security definitions.

<p>Scheme $\text{MuSig2}_{\mathbb{G},g,H_{\text{agg}},H_{\text{nonce}},H_{\text{sign}}}$:</p> <p>$\text{MuSig2.nr} = 3$ $\text{MuSig2.lir} = 2$</p> <p>KeyGen():</p> <ol style="list-style-type: none"> 1 $x \leftarrow_{\mathcal{S}} \mathbb{Z}_p; X \leftarrow g^x$ 2 $\text{st.sk} \leftarrow x; \text{st.vk} \leftarrow X$ 3 Return $(\text{sk} = x, \text{vk} = X)$ <p>KeyAgg($\text{vk}_1, \dots, \text{vk}_n$):</p> <ol style="list-style-type: none"> 4 Return $\prod_{i=1}^n \text{vk}_i^{H_{\text{agg}}(i, \text{vk}_1, \dots, \text{vk}_n)}$ <p>Sign₁():</p> <ol style="list-style-type: none"> 5 $\text{st.j} \leftarrow \text{st.j} + 1$ // num of signing sessions 6 $j \leftarrow \text{st.j}, \text{st}_j.\text{rnd} \leftarrow 1$ 7 For $i \in \{1, \dots, 4\}$: 8 $\text{st}_j.r_i \leftarrow_{\mathcal{S}} \mathbb{Z}_p; R_i \leftarrow g^{\text{st}_j \cdot r_i}$ 9 Return (R_1, \dots, R_4) 	<p>Sign₂($j, k, m, (\text{vk}_i, R_{i,1}, \dots, R_{i,4})_{i=1}^n$):</p> <ol style="list-style-type: none"> 10 If $\text{st}_j.\text{rnd} \neq 1$ or $\text{vk}_k \neq \text{st.vk}$: 11 Return \perp 12 $\text{st}_j.\text{rnd} \leftarrow 2$ 13 $\tilde{\text{vk}} \leftarrow \text{KeyAgg}(\text{vk}_1, \dots, \text{vk}_n)$ 14 For $\ell = 1, \dots, 4$: 15 $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ 16 $b \leftarrow H_{\text{nonce}}(\tilde{\text{vk}}, R_1, \dots, R_4, m)$ 17 $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$ 18 $c \leftarrow H_{\text{sign}}(R, \tilde{\text{vk}}, m)$ 19 $z \leftarrow \sum_{\ell=1}^4 \text{st}_j.r_\ell \cdot b^{\ell-1} + c \cdot \text{sk}_k \cdot H_{\text{agg}}(k, \text{vk}_1, \dots, \text{vk}_n)$ 20 Return (R, z) <p>Sign₃(R, z_1, \dots, z_n):</p> <ol style="list-style-type: none"> 21 Return $(R, \sum_{i=1}^n z_i)$ <p>AggVer($m, \sigma, \tilde{\text{vk}}$):</p> <ol style="list-style-type: none"> 22 $(R, z) \leftarrow \sigma$ 23 Return $[g^z = R \cdot \tilde{\text{vk}}^{H_{\text{sign}}(\tilde{\text{vk}}, R, m)}]$
---	---

Fig. 9. A description of MuSig2 over a group \mathbb{G} of order p with generator g . The third round is often omitted since it can be performed by any observer of the protocol.

In the first signing round each signer k generates four⁶ random values $r_{k,1}, \dots, r_{k,4} \leftarrow_{\mathcal{S}} \mathbb{Z}_p$ and sends $R_{k,\ell} \leftarrow g^{r_{k,\ell}}$ for each $\ell \in \{1, \dots, 4\}$ to all other signers. In the second round, on input $((\text{vk}_i, R_{i,1}, \dots, R_{i,4})_{i=1}^n, m)$, each signer k computes $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ for $\ell \in \{1, \dots, 4\}$, the aggregate verification key $\tilde{\text{vk}}$, and $b \leftarrow H_{\text{nonce}}(\tilde{\text{vk}}, R_1, \dots, R_4, m)$. Then each signer computes the aggregate nonce $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$, the challenge $c \leftarrow H_{\text{sign}}(\tilde{\text{vk}}, R, m)$, and their partial signature $z_k \leftarrow \sum_{\ell=1}^4 r_{k,\ell} \cdot b^{\ell-1} + c \cdot \text{sk}_k \cdot H_{\text{agg}}(k, \text{vk}_1, \dots, \text{vk}_n)$ which they send to all other signers. The final multi-signature is given by $(R, \sum_{i=1}^n z_i)$, which can be computed by any of the signers.

A multi-signature (R, z) can be verified with respect to a message m and an aggregate verification key $\tilde{\text{vk}}$ by checking that $g^z = R \cdot \tilde{\text{vk}}^{H_{\text{sign}}(\tilde{\text{vk}}, R, m)}$, which is identical to the verification of a standard Schnorr signature. MuSig2 satisfies perfect correctness.

AGGREGATOR NODE. In the setting with an aggregator node, the communication cost of MuSig2 can be reduced by having the aggregator compute the R_j 's instead of the signers. More specifically, after the first signing round the aggregator receives $(R_{i,1}, \dots, R_{i,4})_{i=1}^n$ and computes $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ for $\ell \in \{1, \dots, 4\}$. The R_ℓ 's can now be used as the input to the second signing round of each signer, as opposed to all of the $R_{i,\ell}$'s.

This shortcut does not affect the existential and strong unforgeability of MuSig2, since an adversary can compute the R_ℓ 's given the $R_{i,\ell}$'s, and because given a uniformly random R_ℓ an adversary can simulate a selection of $R_{1,\ell}, \dots, R_{n,\ell}$ that appear uniformly random and have product R_ℓ . Therefore, without loss of generality, we do not consider this shortcut in our proof.

STRONG UNFORGEABILITY. The strong unforgeability of MuSig2 is given in the following lemma, which we prove in Appendix D.

Lemma 6 (AOMDL \rightarrow SUF of MuSig2 in the ROM). *Let \mathbb{G} be a group with prime order p and generator g . Let $\text{MS} = \text{MuSig2}[\mathbb{G}, g]$, where its hash functions are modeled as random oracles. Let \mathcal{A}_{ms} be an adversary against $\mathbf{G}^{\text{suf-ms}}[\text{MS}]$ making at most q_s queries to the signing oracle SIGNO_1 and q_h queries to*

⁶ There is a simpler variant of MuSig2 that uses only two nonces [42] that we do not consider in this paper. Its security proof relies on the algebraic group model.

each random oracle. Let $q = 4q_h + 3q_s + 2$. Then, there exists an algorithm \mathcal{D} such that

$$\text{Adv}_{\text{MS}}^{\text{suf-ms}}(\mathcal{A}_{\text{ms}}) \leq \left(2q^3 (\text{Adv}_{\mathbb{G},g}^{\text{aomdl}}(\mathcal{D}) + \frac{32q^2 + 12}{p}) \right)^{1/4}$$

and \mathcal{D} runs in approximately 4 times the runtime of \mathcal{A}_{ms} .

PROOF IDEA: DOUBLE FORKING TECHNIQUE. The proof of the strong unforgeability of MuSig2 is nearly identical to the existential unforgeability proof of [42].

In the existential unforgeability proof, an adversary \mathcal{A}_{ms} that breaks the existential unforgeability of MuSig2 in the ROM on input public key vk is used to win the AOMDL game in the following manner: suppose \mathcal{A}_{ms} outputs a forged signature (R, s) on a message m and public keys (vk_1, \dots, vk_n) where $vk_k = vk$ after making q_s signing oracle queries. First, they construct an adversary \mathcal{B} that simulates $\mathbf{G}^{\text{suf-ms}}[\text{MuSig2}]$ for \mathcal{A}_{ms} by taking $4q_s + 1$ AOMDL challenges but only making q_s queries to the DLOG oracle. Then, they construct an adversary \mathcal{C} that “rewinds” \mathcal{B} by executing it again using the same randomness initially, but with a different value for $H_{\text{sign}}(R, \tilde{vk}, m)$, where \tilde{vk} is the aggregate public key corresponding to (vk_1, \dots, vk_n) . According to the Generalized Forking Lemma [9], the second execution of \mathcal{B} produces a forgery using the same parameters (R, \tilde{vk}, m) with non-negligible probability, allowing the reduction to extract the discrete log of \tilde{vk} . Note that \mathcal{C} runs \mathcal{B} twice, and therefore it makes at most $2q_s$ queries to the DLOG oracle.

Now, to extract the discrete log of vk the reduction constructs an adversary \mathcal{D} that rewinds \mathcal{C} by executing it again using the same randomness initially, but with a different value of $H_{\text{agg}}(k, vk_1, \dots, vk_n)$. If \mathcal{C} succeeds both times with the same list of public keys vk_1, \dots, vk_n , which happens with non-negligible probability, then we can use the discrete logs of the aggregated public key to extract the discrete log of vk . Note that \mathcal{D} runs \mathcal{C} twice, and hence makes at most $4q_s$ queries to the DLOG oracle. However, the reduction algorithm is set so that learning the discrete log of vk , which is the first OMDL challenge, allows the reduction to learn the discrete log of all $4s + 1$ discrete log challenges (the other challenges are used for the nonces in \mathcal{B} ’s simulation of SIGN_{O_2}), hence winning the AOMDL game.

We refer readers to [42] for a thorough explanation of the proof technique.

WHAT WE DO. In Lemma 10 of Appendix D, we construct a modified adversary \mathcal{B} , which uses an adversary \mathcal{A}_{ms} that breaks the strong unforgeability of MuSig2 (as opposed to the existential unforgeability in [42]). Our \mathcal{B} takes the same input and has the same output as the algorithm \mathcal{B} in the proof of [42] and uses its input in the exact same way. Thus, once our Lemma 10 is established, we can plug in our \mathcal{B} to Lemma 3 and Theorem 1 of [42], and obtain a proof of the strong unforgeability of MuSig2 as stated in Lemma 6.

5.4 Analysis of mBCJ

In [4], Bagherzandi et al. present BCJ, a two-round multi-signature scheme. Approximately a decade later, Drijvers et al. found an error in the security proof of BCJ and a sub-exponential attack against its existential unforgeability when concurrent signing sessions are permitted [24], using Wagner’s algorithm for the generalized birthday problem [50]. This attack can be improved to polynomial time using the algorithm of Benhamouda et al. to solve the ROS problem [13]. As an alternative to the insecure BCJ, Drijvers et al. present mBCJ, a modification to the scheme that prevents the mentioned attacks [24].

The modified scheme mBCJ is nearly identical to BCJ, except that some of the scheme parameters (the “commitment parameters”) are computed as the hash of the message being signed, as opposed to public parameters that are the same for every signing session. Thus, a forged mBCJ signature on an unsigned message has to be valid for the corresponding commitment parameters, which are different from the parameters used by the signing oracle for signing different messages. The information gained from the signing oracle is now useless for forging a signature for an unsigned message, and the BCJ attack no longer works. Signing oracle queries for the same message, however, use the same commitment parameters and can assist the adversary in forging an additional signature for the same message. In this section, we use this observation to modify the attack against BCJ to break the one-more unforgeability of mBCJ. Similarly, since the commitment parameters are dependent on the message being signed but not on the signing group, we show that the adversary can forge a signature for a message that was signed by the signing oracle, and is valid with a signing set of the adversary’s choice.

Similarly to HBMS, mBCJ is strongly unforgeable against adversaries that ignore the fact that it is an interactive multi-signature scheme (we define this notion formally in Appendix B, and prove it for mBCJ in

Appendix B.2). In other words, mBCJ produces strongly unforgeable plain signatures in a distributed way, without being strongly unforgeable itself.

SECURITY PROOF OF MBCJ, AND OUR RESULTS. The existential unforgeability of mBCJ is proved under the discrete log assumption [24], but using a weaker security definition than ours. First, the scheme is presented in the key-verification model [5], where signers attach a secret-key proof-of-possession to their public key, to be checked when verifying a signature. Consequently, to win in the corresponding security definition, the adversary must attach valid proofs-of-possession to the public keys used in the forgery. Secondly, in the definition used to prove mBCJ secure, to win the adversary must forge a signature for a message that the honest signer did not sign, regardless of the signing sets used for signing oracle queries and for the forgery. In particular, if the honest signer signed a message m with signing set S , forging a signature for m with a different signing set S' that contains the honest signer is not considered a win for the adversary. This is different from many definitions in literature (notably that of Bellare and Neven [9], and ours), which consider a multi-signature as a signature on the pair (m, S) , where S is the signing set generating the signature.

In this section we first show that even in the setting where mBCJ is proven secure it does not satisfy one-more unforgeability. More specifically, an adversary can query the signing oracle ℓ times for some message m , and obtain $\ell + 1$ signatures for the same message, and the attack is polynomial time. Next, we show that a modification of this attack allows the adversary to not only forge an additional signature for m , but it is possible for that signature to be valid for a group of signers of the adversary's choice, which may be different from the group of signers used in signing oracle queries.

We point out that neither of these attacks contradict the mBCJ security proof of [24], as their security definition does not consider such attacks a win for the adversary.

TREE-BASED COMMUNICATION MODEL. To improve efficiency and reduce the communication complexity of mBCJ, it is described with a tree-based communication model. The signers are arranged in a directed tree with a root, who initiates each signing round and obtains the final signature. At each round, each signer receives the input from their parent and passes it to their children. When the input reaches a leaf node, they complete the signing round and pass their output to their parent, who combines their children's output with their own output and passes it up the tree towards the root.

In mBCJ, the tree structure allows the outputs of both signing rounds to be aggregated as they percolate up the tree, and in the second round the signers only learn the aggregation of the first signing round outputs as opposed to the output of all the signers. While we present the scheme without this optimization (for the sake of consistency with the other schemes in this paper), our attacks work in that setting as well.

DESCRIPTION OF MBCJ. The mBCJ scheme is parameterized by a group \mathbb{G} of prime order p with a generator g_1 , and hash functions H_0, H_1 with codomain \mathbb{Z}_p and H_2 with codomain \mathbb{G}^3 .

For key generation, each signer picks $x \leftarrow_{\$} \mathbb{Z}_p$ and computes $X \leftarrow g^x$. Then, they compute a proof-of-possession $\pi = (c, z)$ by choosing $r \leftarrow_{\$} \mathbb{Z}_p$, computing $c \leftarrow H_1(X, g_1^r)$, and $z \leftarrow r + c \cdot x$. The output is $(sk = x, vk = (X, \pi))$.

In the first signing round, on input message m , each signer i computes the commitment parameters $(g_2, h_1, h_2) \leftarrow H_2(m)$. Then, they choose $(r_i, \alpha_{i,1}, \alpha_{i,2}) \leftarrow_{\$} \mathbb{Z}_p^3$ and compute $t_{i,1} \leftarrow g_1^{\alpha_{i,1}} h_1^{\alpha_{i,2}}$ and $t_{i,2} \leftarrow g_2^{\alpha_{i,1}} h_2^{\alpha_{i,2}} g_1^{r_i}$, outputting $(t_{i,1}, t_{i,2})$. For the second signing round, on input $(t_{j,1}, t_{j,2}, X_j)_{j=1}^n$, each signer i computes the aggregate public key $\bar{X} \leftarrow \prod_j X_j$, the aggregate nonces $\bar{t}_1 \leftarrow \prod_j t_{j,1}$ and $\bar{t}_2 \leftarrow \prod_j t_{j,2}$, and the challenge $c \leftarrow H_0(\bar{t}_1, \bar{t}_2, \bar{X}, m)$. Finally, they output $(\alpha_{i,1}, \alpha_{i,2}, s_i)$, where $s_i = r_i + c \cdot x_i$. The final multi-signature is $(\bar{t}_1, \bar{t}_2, \bar{s}, \bar{\alpha}_1, \bar{\alpha}_2)$, where $\bar{s} \leftarrow \sum_j s_j$, $\bar{\alpha}_1 \leftarrow \sum_j \alpha_{j,1}$, and $\bar{\alpha}_2 \leftarrow \sum_j \alpha_{j,2}$.

To verify a signature $(\bar{t}_1, \bar{t}_2, \bar{s}, \bar{\alpha}_1, \bar{\alpha}_2)$ with respect to a message m and keys $vk_1 = (X_1, \pi_1), \dots, vk_n = (X_n, \pi_n)$, the verifier first verifies all the proofs-of-possession by parsing each π_i as (c_i, z_i) and checking that $c_i = H_1(X_i, g_1^{z_i} X_i^{-c_i})$, outputting false if any of the checks fails. Then, they compute the commitment parameters $(g_2, h_1, h_2) \leftarrow H_2(m)$, the aggregate key $\bar{X} \leftarrow \prod_j X_j$, and the challenge $c \leftarrow H_0(\bar{t}_1, \bar{t}_2, \bar{X}, m)$. The verification returns true if and only if $\bar{t}_1 = g_1^{\bar{\alpha}_1} h_1^{\bar{\alpha}_2}$ and $\bar{t}_2 = g_2^{\bar{\alpha}_1} h_2^{\bar{\alpha}_2} g_1^{\bar{s}} \cdot \bar{X}^{-c}$.

ONE-MORE UNFORGEABILITY ATTACK.⁷ We present the attack in the two-signer setting where one signer is corrupt. It is easy to generalize it to a setting with more signers, as long as at least one is corrupt. This attack also applies in the tree-based communication model when the root is corrupt.

Let S_1 be a corrupt signer controlled by the adversary, and let S_2 be an honest signer with whom the adversary can communicate via a signing oracle. Let m be a message of the adversary's choice, let $(g_2, h_1, h_2) \leftarrow H_2(m)$, and pick $\ell \geq \lceil \log(p) \rceil$. Each signer $S_i \in \{S_1, S_2\}$ proceeds with key generation honestly by picking $x_i \leftarrow_{\$} \mathbb{Z}_p$, computing $X_i \leftarrow g^{x_i}$, and computing a proof-of-possession π as described in the scheme description.

The adversary begins ℓ signing session for the message m , obtaining $t_{2,1,j}$ and $t_{2,2,j}$ from the S_2 signing oracle for each session j . Now, the adversary chooses $(\alpha_{1,1,j}, \alpha_{1,2,j}) \leftarrow_{\$} \mathbb{Z}_p^2$ for each session j , as well as two options $(r_{1,j}^0, r_{1,j}^1) \leftarrow_{\$} \mathbb{Z}_p^2$. Compute the nonce shares $t_{1,1,j} \leftarrow g_1^{\alpha_{1,1,j}} h_1^{\alpha_{1,2,j}}$ and $t_{1,2,j}^b = g_2^{\alpha_{1,1,j}} h_2^{\alpha_{1,2,j}} g_1^{r_{1,j}^b}$ for $b \in \{0, 1\}$, and the consequent aggregate nonces $\bar{t}_{1,j} = t_{1,1,j} t_{2,1,j}$ and $\bar{t}_{2,j}^b = t_{1,2,j}^b t_{2,2,j}$ for $b \in \{0, 1\}$. For each session $j \in \{1, \dots, \ell\}$ and $b \in \{0, 1\}$ compute the challenge $c_j^b \leftarrow H_0(\bar{t}_{1,j}, \bar{t}_{2,j}^b, X_1 X_2, m)$. Now, define the group homomorphisms $\rho_+ : (\mathbb{Z}_p)^\ell \rightarrow \mathbb{Z}_p$ and $\rho_\times : \mathbb{G}^\ell \rightarrow \mathbb{G}$ given by

$$\rho_+(y_1, \dots, y_\ell) = \sum_{j=1}^{\ell} \frac{2^{j-1} y_j}{c_j^1 - c_j^0}$$

and

$$\rho_\times(y_1, \dots, y_\ell) = \prod_{j=1}^{\ell} y_j^{\frac{2^{j-1}}{c_j^1 - c_j^0}},$$

and define $\bar{t}_{1,\ell+1} \leftarrow \rho_\times(t_{2,1,1}, \dots, t_{2,1,\ell})$ and $\bar{t}_{2,\ell+1} \leftarrow \rho_\times(t_{2,2,1}, \dots, t_{2,2,\ell})$. Let $c_{\ell+1} \leftarrow H_0(\bar{t}_{1,\ell+1}, \bar{t}_{2,\ell+1}, X_1 X_2, m)$, let $d \leftarrow c_{\ell+1} - \rho_+(c_1^0, \dots, c_\ell^0)$, and write it in binary as $d = \sum_{j=1}^{\ell} 2^{j-1} b_j$ for some $b_1, \dots, b_\ell \in \{0, 1\}$, which is possible since $\ell \geq \lceil \log_2(p) \rceil$.

Next, continue to the second round of each signing session j with the group of signers (S_1, S_2) by sending $(t_{1,1,j}, t_{1,2,j}^{b_j})$ to the S_2 signing oracle, and obtain a response $(s_{2,j}, \alpha_{2,1,j}, \alpha_{2,2,j})$. First, the adversary completes each of the j sessions by following the protocol to generate a legitimate multi-signature

$$\sigma_j \leftarrow (\bar{t}_{1,j}, \bar{t}_{2,j}^{b_j}, s_{2,j} + r_{1,j}^{b_j} + c_j^{b_j} \cdot x_1, \alpha_{1,1,j} + \alpha_{2,1,j}, \alpha_{1,2,j} + \alpha_{2,2,j}).$$

Additionally, the adversary forges a signature

$$\sigma_{\ell+1} = (\bar{t}_{1,\ell+1}, \bar{t}_{2,\ell+1}, \bar{s}_{\ell+1}, \bar{\alpha}_{1,\ell+1}, \bar{\alpha}_{2,\ell+1})$$

where the \bar{t} 's are as defined above, $\bar{s}_{\ell+1} \leftarrow \rho_+(s_{2,1}, \dots, s_{2,\ell}) + x_1 \cdot c_{\ell+1}$, and $\bar{\alpha}_{k,\ell+1} \leftarrow \rho_+(\alpha_{2,k,1}, \dots, \alpha_{2,k,\ell})$ for $k \in \{1, 2\}$.

Note that $\sigma_1, \dots, \sigma_\ell$ were obtained legitimately with both signers following the protocol, and hence by the perfect correctness of mBCJ they are valid. We will prove below that $\sigma_{\ell+1}$ is also valid for the message m and signing group (S_1, S_2) . This implies that the adversary obtained $\ell + 1$ signatures after only completing ℓ signing oracle signing sessions, breaking the one-more unforgeability of mBCJ. Also note that the adversary has a proofs-of-possession of the secret key associated with X_1 , and thus this attack works in the key-verification model.

VALIDITY OF $\sigma_{\ell+1}$. To verify the validity of $\sigma_{\ell+1}$ we must check that $\bar{t}_{1,\ell+1} = g_1^{\bar{\alpha}_{1,\ell+1}} h_1^{\bar{\alpha}_{2,\ell+1}}$ and $\bar{t}_{2,\ell+1} = g_2^{\bar{\alpha}_{1,\ell+1}} h_2^{\bar{\alpha}_{2,\ell+1}} g_1^{\bar{s}_{\ell+1}} \cdot (X_1 X_2)^{-c_{\ell+1}}$. Starting from the left hand side of the first equation, we have that

$$\begin{aligned} \bar{t}_{1,\ell+1} &= \rho_\times(t_{2,1,1}, \dots, t_{2,1,\ell}) = \rho_\times(g_1^{\alpha_{2,1,1}} h_1^{\alpha_{2,2,1}}, \dots, g_1^{\alpha_{2,1,\ell}} h_1^{\alpha_{2,2,\ell}}) \\ &= g_1^{\rho_+(\alpha_{2,1,1}, \dots, \alpha_{2,1,\ell})} h_1^{\rho_+(\alpha_{2,2,1}, \dots, \alpha_{2,2,\ell})} = g_1^{\bar{\alpha}_{1,\ell+1}} h_1^{\bar{\alpha}_{2,\ell+1}}, \end{aligned}$$

which is what we wanted to show.

For the second equation, starting from the right hand side, we have that

$$g_2^{\bar{\alpha}_{1,\ell+1}} h_2^{\bar{\alpha}_{2,\ell+1}} g_1^{\bar{s}_{\ell+1}} \cdot (X_1 X_2)^{-c_{\ell+1}} =$$

⁷ In this attack there are two signers, each with a share of two nonces per signing session, participating in ℓ sessions. This requires three-dimensional indexing, and we use subscripts i, j, k to denote signer i , nonce number j , in session k . An overline is used for aggregated nonces, keys, or signature shares, with the corresponding number and session in subscript. A superscript is used for a potential selection by the adversary.

$$\begin{aligned}
 &= g_2^{\rho_+(\alpha_{2,1,1}, \dots, \alpha_{2,1,\ell})} h_2^{\rho_+(\alpha_{2,2,1}, \dots, \alpha_{2,2,\ell})} g_1^{\rho_+(s_{2,1}, \dots, s_{2,\ell}) + x_1 \cdot c_{\ell+1}} (X_1 X_2)^{-c_{\ell+1}} \\
 &= \rho_\times(t_{2,2,1}, \dots, t_{2,2,\ell}) X_1^{c_{\ell+1}} X_2^{\rho_+(c_1^{b_1}, \dots, c_\ell^{b_\ell})} (X_1 X_2)^{-c_{\ell+1}} \\
 &= \bar{t}_{2,\ell+1} \cdot X_2^{\rho_+(c_1^{b_1}, \dots, c_\ell^{b_\ell}) - c_{\ell+1}}.
 \end{aligned}$$

However, by a proof identical to the proof of Lemma 3 (with slightly different notation), $c_{\ell+1} = \rho_+(c_1^{b_1}, \dots, c_\ell^{b_\ell})$. Consequently, the exponent of X_2 cancels out, leaving us with $\bar{t}_{2,\ell+1}$, which completes the proof.

FORGERY THAT IS VALID FOR AN ARBITRARY SIGNING GROUP. Again, consider an adversary who has signing oracle access to an honest signer. We will show that after only calling the signing oracle for the message m and signing set S , the adversary can find a valid signature for m that is valid for a different set of signers S' that contains the honest signer.

A trivial way to achieve this is to find two groups of signers S and S' with the same aggregated public key. In this case, any signature that is valid for m and S is also valid for m and S' . As a concrete example, consider the setting from before where there is an honest signer S_2 with public key X_2 (and a secret key proof-of-possession), and a signer S_1 with public key $X_1 = g_1^{x_1}$ controlled by the adversary. Using the signing oracle, the adversary can obtain a signature σ that is valid for m and the signing group (S_1, S_2) . Now, the adversary can choose $x_3, x_4 \in \mathbb{Z}_p$ satisfying $x_3 + x_4 = x_1$, and since $X_1 X_2 = X_1 g_1^{x_3} g_1^{x_4}$, the multi-signature σ is valid for the group of signers with public keys $(X_2, g_1^{x_3}, g_1^{x_4})$. Furthermore, the adversary knows the secret keys associated with $g_1^{x_3}$ and $g_1^{x_4}$, so they can provide proofs-of-possession for those keys and the attack works in the key-verification model.

This simple attack is enough to show that mBCJ does not satisfy our definition of existential unforgeability. It does not, however, allow the adversary to forge signatures that are valid for aggregate public keys that were not used in signing oracle queries. We present a simple modification of the one-more unforgeability attack that allows the adversary to do that.

As before, consider a corrupted signer S_1 with public key $X_1 = g_1^{x_1}$, an honest signer S_2 with public key X_2 , and additionally consider a signer S_{evil} with public key $X_{\text{evil}} = g_1^{x_{\text{evil}}}$ with their private key known to the adversary. The adversary wishes to forge a signature for the message m with signing set (S_{evil}, S_2) after only making signing oracle queries with the signing set (S_1, S_2) . To do that, the adversary carries out our one-more unforgeability attack against mBCJ with the following modifications.

- It sets the forgery challenge $c_{\ell+1} \leftarrow H_0(\bar{t}_{1,\ell+1}, \bar{t}_{2,\ell+1}, X_{\text{evil}} X_2, m)$, where X_{evil} replaces the X_1 in the original attack.
- In the forged signature $\sigma_{\ell+1}$, it sets $\bar{s} \leftarrow \rho_+(s_{2,1}, \dots, s_{2,\ell}) + x_{\text{evil}} \cdot c_{\ell+1}$, where x_{evil} replaces the x_1 in the original attack.

By following the steps to verify the validity of $\sigma_{\ell+1}$ in the original attack, we can see that $\sigma_{\ell+1}$ is valid for the message m and the signing set (S_{evil}, S_2) . Note that this set may have a different aggregate public key than those used in all signing oracle queries. Furthermore, the adversary knows the secret key x_{evil} and can generate a proof of possession, so the attack works in the key verification model.

6 Threshold Signatures Schemes

6.1 Comparison to Previous SUF-TS Definition, and FROST

In [6, 12], Bellare et al. provide a hierarchy of unforgeability definitions for a limited class of threshold signature schemes. As part of this hierarchy, they define strong unforgeability for schemes that satisfy the following properties:

Semi non-interactive: The scheme has at most one pre-processing signing round, which takes no input and outputs a pre-processing token pp . Additionally, the scheme has a single signing round that takes the message, the set of participating signers, as well as the pre-processing tokens of all participants as input. The input to the message dependent signing round is called a leader request, denoted lr , with fields $lr.m$, $lr.S$, and $(lr.pp_i)_{i \in S}$.

Strong-verification: The scheme has an additional strong verification algorithm $S\text{Verify}$, which verifies a signature with respect to a public key and a leader request. It is required that for each leader request

lr and aggregated public key \tilde{vk} there exist at most one signature σ such that $\text{SVerify}(\tilde{vk}, lr, \sigma) = \text{true}$. Correctness requires that $\text{SVerify}(\tilde{vk}, lr, \sigma) = \text{true}$ whenever all (of the t or more) signers in S honestly computed σ with lr as the input to the message dependent signing round.

While this class of schemes appears limited, it contains FROST, and is therefore of practical interest.

The hierarchy of strong unforgeability definitions in [6, 12] begins at TS-SUF-2, the weakest of their definitions, that both the FROST 1 [32] and FROST 2 [6, 19] variants satisfy under the one-more discrete log assumption [8] in the random oracle model (Theorem 5.3 and 5.1 of [12]). This definition considers an adversary who corrupts c out of n signers, where $c < t$. A secure scheme guarantees that if an adversary obtains a valid signature σ on a message m , then there exist a leader request lr with the same message that was signed by at least $t - c$ honest signers and $\text{SVerify}(\tilde{vk}, lr, \sigma) = \text{true}$. Figure 10 shows this definition using our syntax, and compares it to our strong unforgeability definition when restricted to the same class of schemes.

TS-SUF-2 IMPLIES OUR DEFINITION. We claim that a TS-SUF-2 secure scheme is also one-more unforgeable according to our definition by showing that any adversary that wins $\mathbf{G}^{\text{suf-ts}}$ can be modified to win $\mathbf{G}^{\text{ts-suf-2-crypto22}}$ with the same probability and no significant increase in runtime. Note that this implies the one-more unforgeability of FROST 1 and FROST 2 under the OMDL assumption in the ROM.

Suppose there exists an adversary \mathcal{A} that wins $\mathbf{G}_{n,t}^{\text{suf-ts}}[\text{TS}]$ with probability p . To construct an adversary \mathcal{A}' that plays $\mathbf{G}_{n,t}^{\text{ts-suf-2-crypto22}}[\text{TS}]$ we can simply execute \mathcal{A} (since the two games have access to the same oracles, with the same behavior) and maintain a set L of the leader requests that \mathcal{A} inputs to SIGN_2 (and receives a non- \perp response). If \mathcal{A} wins $\mathbf{G}_{n,t}^{\text{suf-ts}}[\text{TS}]$, it must come up with valid distinct signatures $(\sigma_j)_{j=1}^\ell$ for some message m such that $\ell \geq |\{lr \in L : lr.m = m\}|$. Hence, there exist some σ_j for which $\text{SVerify}(\tilde{vk}, lr, \sigma_j) = \text{false}$ for all $lr \in \{lr' \in L : lr'.m = m\}$, and thus \mathcal{A}' can use σ_j to win $\mathbf{G}_{n,t}^{\text{ts-suf-2-crypto22}}[\text{TS}]$. Note that \mathcal{A}' has roughly the same runtime as \mathcal{A} , and it wins $\mathbf{G}_{n,t}^{\text{ts-suf-2-crypto22}}[\text{TS}]$ with probability p .

OUR DEFINITION DOES NOT IMPLY TS-SUF-2. We note that TS-SUF-2 is strictly stronger than our one-more unforgeability definition. To show that, we present a toy scheme TS' that is one-more unforgeable but does not satisfy TS-SUF-2 (in fact, it does not satisfy the weaker TS-UF-2 in the security hierarchy of Bellare et al.).

Consider a semi non-interactive threshold signature scheme TS that is TS-SUF-2 secure (set $\text{TS} = \text{FROST 1}$, for example). Now, modify it in the following way to obtain TS' :

Key generation: The key generation of TS' is identical to that of TS .

First signing round (pre-processing round): The pre-processing round of TS' is the same as that of TS , except for an extra bit that is appended to the output and is set to 0.

Second signing round: The second signing round of TS' is the same as that of TS , and the extra bit at the end of the pre-processing tokens is ignored.

Verification: The signature verification of TS' is identical to that of TS .

First, note that TS is one-more unforgeable (OMUF) using our definition, since TS-SUF-2 implies OMUF. Consequently, TS' is OMUF, since an adversary \mathcal{A} that breaks the OMUF of TS' can easily be converted into an adversary \mathcal{A}' against the OMUF of TS with the same success probability.⁸

Now, we claim that TS' is not TS-SUF-2 (in fact, it is not even TS-UF-2). The reason for that is that the adversary can obtain signature shares from honest signers using different leader requests (by flipping the unused bit of the pre-processing tokens), but those shares combine to a valid threshold signature (since the flipped last bit is ignored during signing).

For a concrete attack, consider a group of two honest signers S_1 and S_2 with a signing threshold of two ($n = t = 2$). The adversary calls the signing oracle for the first signing round to obtain pre-processing tokens from each signer, pp_1 and pp_2 . Now, the adversary calls the second round signing oracle for S_1 with the message m , signing set $\{S_1, S_2\}$, and pre-processing tokens $(\text{pp}_1, \text{pp}_2)$ to obtain a signature share σ_1 .

⁸ \mathcal{A}' would simulate the first signing round to \mathcal{A} by calling the TS signing oracle and appending 0 to the output, and the second signing round by calling the TS signing oracle after removing the unnecessary bits from the pre-processing tokens. If \mathcal{A} wins the OMUF game against TS' , then its output is used by \mathcal{A}' to win the OMUF game against TS .

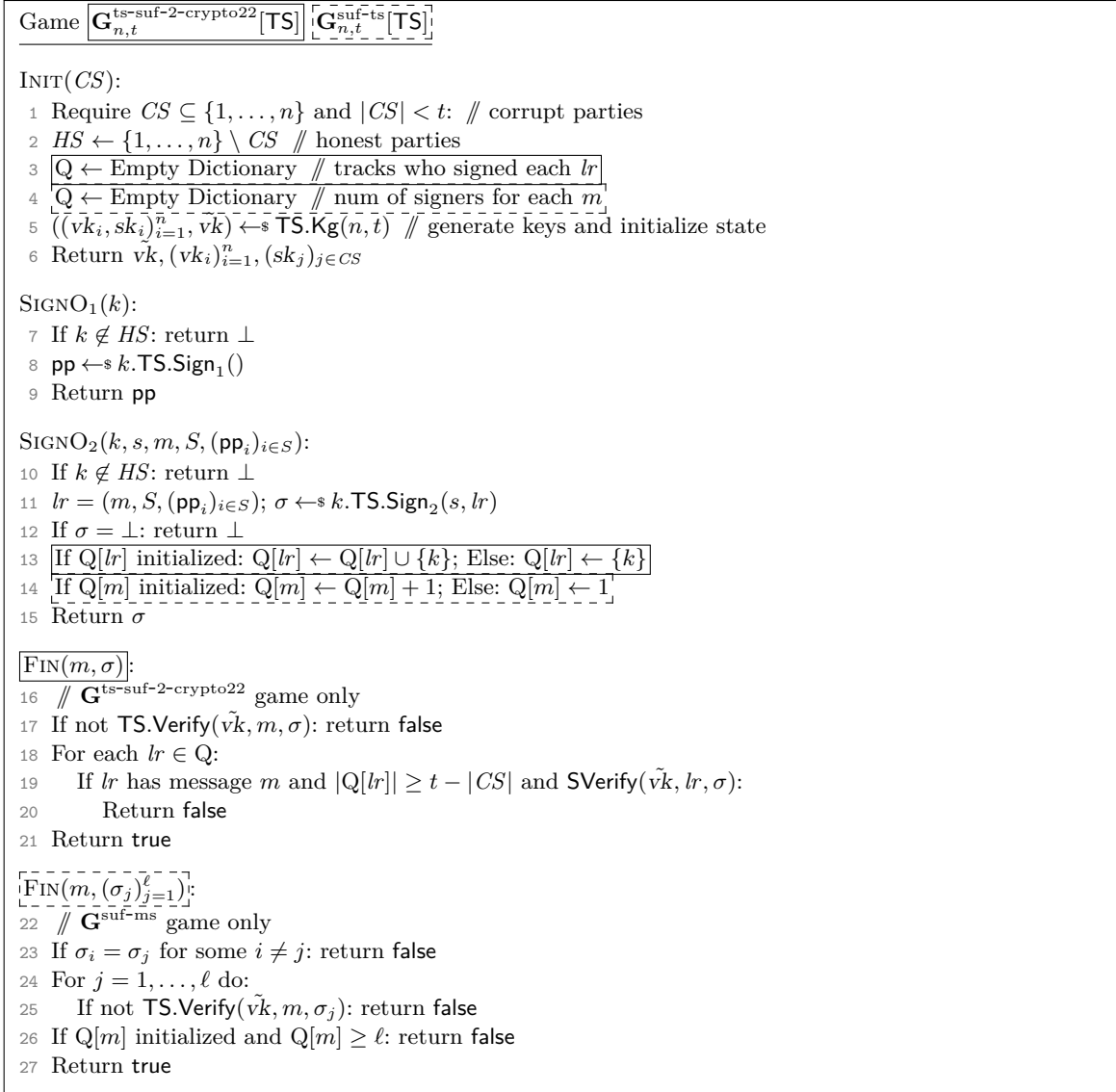


Fig. 10. Game $\mathbf{G}_{n,t}^{\text{ts-suf-2-crypto22}}$ used to define TS-SUF-2 for a threshold signature scheme TS in [6, 12] (adjusted to our syntax), compared to our strong unforgeability definition $\mathbf{G}_{n,t}^{\text{suf-ts}}$ restricted to schemes where TS-SUF-2 is defined.

Similarly, the adversary calls the second round signing oracle for S_2 with the same message and signing set, and same pre-processing tokens but with the last bit of pp_1 flipped to obtain σ_2 . Since the last bit of the pre-processing tokens is ignored during signing, σ_1 and σ_2 combine to a valid threshold signature for the message m . However, since each signer signed a different leader request, no leader request with the message m was signed by two signers and hence this attack breaks TS-SUF-2 (and TS-UF-2).

6.2 Analysis of ROAST

In FROST-like threshold signature schemes [26, 32, 17, 20, 49] signing a message involves selecting a signing subset (of sufficient size), after which it is required that each signing set member participates in a signing session honestly, else signature generation fails. This leads to a lack of robustness, losing one of the main advantages of threshold signatures over multi-signatures.

ROAST [46] is a simple wrapper algorithm that adds robustness to such a scheme Σ by executing the Σ signing protocol multiple times with different signing sets until one is successful. The only requirements on the underlying scheme are the following:

Identifiable aborts: Signers that do not participate honestly in a Σ signing sessions can be detected with overwhelming probability.

Semi non-interactive: Σ has one signing round that requires knowing the signing set, potentially in addition to a “pre-processing” signing round that can be executed before the signing set is selected.

The canonical schemes that satisfy these properties are FROST 1 [26] and FROST 3 [46]. ROAST also uses an aggregator who facilitates communication between the signers and is trusted for robustness, but not for unforgeability.

We argue that the robustness of ROAST comes at the cost of strong unforgeability. ROAST does not satisfy any notion of strong or one-more unforgeability, even if Σ is strongly unforgeable. This is in spite of the fact that ROAST does not fit our syntax and security definition for threshold signatures. It remains an important open problem to define strong unforgeability for a class of schemes that contains ROAST and construct a strongly unforgeable alternative.

THE SCHEME. We describe the scheme informally, and a formal description can be found in Figure 4 of [46].

Suppose Σ is a non-robust and semi non-interactive threshold signature scheme that supports identifiable aborts. Let S denote a signing group of size n with a signing threshold t . The key generation and verification in ROAST are the same as those of Σ .

To sign a message m , the aggregator maintains a set R of available signers, initially adding signers to R whenever they complete the pre-processing round. Whenever $|R| \geq t$, the aggregator picks a subset $T \subseteq R$ with $|T| = t$, initiates a Σ signing session with the message m for the signers of T , and removes them from R . Whenever a signer completes a signing session and another pre-processing round honestly, it is added back to R . The protocol terminates when one of the signing session produces a valid Σ signature.

This protocol is guaranteed to produce a valid signature whenever t of the signers are honest and responsive. Furthermore, since Σ supports identifiable aborts each signer can sabotage at most one Σ signing session, and thus ROAST terminates within a number of sessions that is at most the number of corrupt signers.

INCOMPATIBILITY WITH OUR UNFORGEABILITY DEFINITIONS. ROAST is inherently incompatible with our unforgeability definitions. To begin, the number of interactive signing rounds is not determined until the protocol terminates, and our syntax does not support such schemes. This issue can be circumnavigated with some syntactical overhead by allowing schemes to adaptively choose the number of signing rounds. More importantly, however, the signers cannot tell whether a signing round is the last interactive signing round. Thus, even in an honest execution, they cannot determine at which signing round the aggregator should be able to construct a threshold signature. In the context of our security definition, the unforgeability game cannot tell at which signing oracle query to increase the count of signatures obtained legitimately.

BREAKING STRONG UNFORGEABILITY. While it remains an open problem to formally define strong unforgeability for ROAST-like schemes, we show that ROAST does not satisfy any notion of strong unforgeability.

Consider, for instance, a group of 100 signers with a signing threshold of 67 and a corrupt aggregator that wishes to obtain many signatures for some message m . Even if all signers are honest, the aggregator can ask each signer to participate in multiple Σ signing sessions with various signing sets and obtain as many as $\binom{100}{67} \approx 3 \times 10^{26}$ signatures for m . This is despite the fact that each signer participated in a single ROAST session.

7 Acknowledgements

We thank the anonymous Asiacrypt 2024 reviewers for their insightful feedback, and for suggesting that mBCJ is not strongly unforgeable. This research was supported in part by NSF grants CNS-2026774, CNS-2154174, a JP Morgan Faculty Award, a CISCO Faculty Award, and a gift from Microsoft.

References

1. J. F. Almansa, I. Damgård, and J. B. Nielsen. Simplified threshold RSA with adaptive and proactive security. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 593–611, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 11
2. M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. On the malleability of bitcoin transactions. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *Financial Cryptography and Data Security*, pages 1–18, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. 3
3. R. Bacho, J. Loss, S. Tessaro, B. Wagner, and C. Zhu. Twinkle: Threshold signatures from ddd with full adaptive security. In M. Joye and G. Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 429–459, Cham, 2024. Springer Nature Switzerland. 11
4. A. Bagherzandi, J. H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM CCS 2008: 15th Conference on Computer and Communications Security*, pages 449–458, Alexandria, Virginia, USA, Oct. 27–31, 2008. ACM Press. 20
5. A. Bagherzandi and S. Jarecki. Multisignatures using proofs of secret key possession, as secure as the diffie-hellman problem. In R. Ostrovsky, R. De Prisco, and I. Visconti, editors, *Security and Cryptography for Networks*, pages 218–235, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 21
6. M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. Better than advertised security for non-interactive threshold signatures. In Y. Dodis and T. Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550, Santa Barbara, CA, USA, Aug. 15–18, 2022. Springer, Heidelberg, Germany. 3, 4, 6, 10, 11, 23, 24, 25
7. M. Bellare and W. Dai. Chain reductions for multi-signatures and the HBMS scheme. In M. Tibouchi and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 650–678, Singapore, Dec. 6–10, 2021. Springer, Heidelberg, Germany. 3, 4, 6, 9, 11, 12, 15, 16, 18, 32, 33
8. M. Bellare, C. Namprempe, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum’s blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003. 18, 24
9. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, Oct. 30 – Nov. 3, 2006. ACM Press. 3, 5, 9, 20, 21
10. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press. 5
11. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 4
12. M. Bellare, S. Tessaro, and C. Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. <https://eprint.iacr.org/2022/833>. 3, 4, 6, 10, 11, 23, 24, 25
13. F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. On the (in)security of ROS. In A. Canteaut and F.-X. Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 33–53, Zagreb, Croatia, Oct. 17–21, 2021. Springer, Heidelberg, Germany. 4, 11, 14, 20
14. D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In T. Peyrin and S. Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, Dec. 2–6, 2018. Springer, Heidelberg, Germany. 3, 9, 15
15. L. T. A. N. Brandão and M. Davidson. Notes on threshold EdDSA/Schnorr signatures. Technical Report NIST IR 8214B ipd, National Institute of Standards and Technology, Gaithersburg, MD, 2022. <https://doi.org/10.6028/NIST.IR.8214B.ipd>. 3
16. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, Oct. 14–17, 2001. IEEE Computer Society Press. 4
17. H. Chu, P. Gerhart, T. Ruffing, and D. Schröder. Practical Schnorr threshold signatures without the algebraic group model. In *Crypto 2023*, Aug. 19–24, 2023. 11, 25
18. D. Connolly, C. Komlo, I. Goldberg, and C. A. Wood. Two-Round Threshold Schnorr Signatures with FROST. Internet-Draft draft-irtf-cfrg-frost-10, Internet Engineering Task Force, Sept. 2022. Work in Progress. 3

19. E. Crites, C. Komlo, and M. Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Paper 2021/1375, 2021. 4, 11, 24
20. E. Crites, C. Komlo, and M. Maller. Fully adaptive Schnorr threshold signatures. In *Advances in Cryptology – CRYPTO 2023*, Aug. 2023. 3, 10, 11, 25
21. C. Decker and R. Wattenhofer. Bitcoin transaction malleability and mtgox. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014*, pages 313–326, Cham, 2014. Springer International Publishing. 3
22. Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO’87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, Aug. 16–20, 1988. Springer, Heidelberg, Germany. 3
23. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer, Heidelberg, Germany. 3
24. M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press. 4, 14, 20, 21
25. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany. 15, 32
26. R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1179–1194, Toronto, ON, Canada, Oct. 15–19, 2018. ACM Press. 25, 26
27. R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In M. Manulis, A.-R. Sadeghi, and S. Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany. 3
28. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersen’s distributed key generation protocol. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 373–390, San Francisco, CA, USA, Apr. 13–17, 2003. Springer, Heidelberg, Germany. 5
29. K. Itakura, K; Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC research & development*, 1983. 3
30. E. Kiltz, D. Masny, and J. Pan. Optimal security proofs for signatures from identification schemes. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61, Santa Barbara, CA, USA, Aug. 14–18, 2016. Springer, Heidelberg, Germany. 32
31. C. Komlo and I. Goldberg. Frost: flexible round-optimized Schnorr threshold signatures. In *International Conference on Selected Areas in Cryptography*, pages 34–65. Springer, 2020. 3
32. C. Komlo and I. Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In O. Dunkelman, M. J. J. Jr., and C. O’Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65, Halifax, NS, Canada (Virtual Event), Oct. 21–23, 2020. Springer, Heidelberg, Germany. 4, 11, 24, 25
33. J. Lau and P. Wuille. Dealing with signature encoding malleability. Bitcoin Improvement Proposal 146, 2016. <https://github.com/bitcoin/bips/blob/master/bip-0146.mediawiki>. 3
34. B. Libert, M. Joye, and M. Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *Theoretical Computer Science*, 2016. 11
35. Y. Lindell. Simple three-round multiparty Schnorr signing with full simulatability. *IACR Communications in Cryptology*, 1(1), 2024. 4
36. G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple Schnorr multi-signatures with applications to bitcoin (deprecated version). Cryptology ePrint Archive, Report 2018/068, version 1, 2018. <https://eprint.iacr.org/archive/2018/068/20180118:124757>. 11, 14
37. G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple Schnorr multi-signatures with applications to bitcoin. In *Design, Code, and Cryptography*, pages 2139–2164, September 2019. 3, 4, 5, 9, 14, 15
38. National Institute of Standards and Technology. Multi-Party Threshold Cryptography, 2018–Present. <https://csrc.nist.gov/Projects/threshold-cryptography>. 3
39. S. Navot. Insecurity of musig and bn multi-signatures with delayed message selection. Cryptology ePrint Archive, Report 2024/437, 2024. <https://eprint.iacr.org/2024/437>. 15
40. J. Nick. Insecure shortcuts in musig, 2019. <https://medium.com/blockstream/insecure-shortcuts-in-musig-2ad0d38a97da>. 15
41. J. Nick, T. Ruffing, and E. Jin. Musig2 for bip340-compatible multi-signatures. Bitcoin Improvement Proposal 327, 2022. <https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki>. 3

42. J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In T. Malkin and C. Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221, Virtual Event, Aug. 16–20, 2021. Springer, Heidelberg, Germany. 3, 4, 5, 9, 15, 18, 19, 20, 41
43. D. Pointcheval and J. Stern. Provably secure blind signature schemes. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology – ASIACRYPT’96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265, Kyongju, Korea, Nov. 3–7, 1996. Springer, Heidelberg, Germany. 4
44. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. In *Journal of Cryptology*. Journal of Cryptology, May 1998. 3
45. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. 4
46. T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. ROAST: Robust asynchronous Schnorr threshold signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS ’22*, page 2551–2564, New York, NY, USA, 2022. Association for Computing Machinery. 4, 26
47. C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer, Heidelberg, Germany. 3
48. C. P. Schnorr. Security of blind discrete log signatures against interactive attacks. In S. Qing, T. Okamoto, and J. Zhou, editors, *Information and Communications Security*, pages 1–12, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 4, 11
49. S. Tessaro and C. Zhu. Threshold and multi-signature schemes from linear hash functions. In *Advances in Cryptology – EUROCRYPT 2023*, Lyon, France, Apr. 23–27, 2023. 9, 25
50. D. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Santa Barbara, CA, USA, Aug. 18–22, 2002. Springer, Heidelberg, Germany. 20
51. P. Wuille. Dealing with malleability. Bitcoin Improvement Proposal 62, 2014. <https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki>. 3
52. P. Wuille, J. Nick, and T. Ruffing. Schnorr signatures for secp256k1. Bitcoin Improvement Proposal 340, 2020. <https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki>. 3

A Toy Strongly Unforgeable Multi-Signature Scheme

We first construct a toy multi-signature scheme that is strongly unforgeable. In the toy scheme, the size of the signatures is proportional to the number of signers, defying the point of using a multi-signature scheme in the first place. It is only included to help illustrate the difference of strong unforgeability from existential unforgeability.

Suppose Σ is a deterministic (single signer) digital signature scheme such that each message m and public key vk have a unique valid signature. We define a multi-signature scheme $\text{Toy}[\Sigma]$ with a single interactive signing round as follows:

Key generation: Each party runs $(vk, sk) \leftarrow^* \Sigma.\text{Kg}$, saves the result, and outputs vk .

Signing: On input $(k, m, (vk_i)_{i=1}^n)$, each party verifies that $vk_k = vk$ (and aborts otherwise). Then, it runs $\sigma_k \leftarrow \Sigma.\text{Sign}(sk, vk_1 \parallel \dots \parallel vk_n \parallel m)$, where \parallel denotes string concatenation. The final multi-signature is a list of all partial signatures $(\sigma_1, \dots, \sigma_n)$.

Verification: On input $\sigma = (\sigma_1, \dots, \sigma_n)$, m , and vk_1, \dots, vk_n , the verification algorithm returns true if and only if $\Sigma.\text{Verify}(vk_i, vk_1 \parallel \dots \parallel vk_n \parallel m, \sigma_i) = \text{true}$ for all $i \in \{1, \dots, n\}$.

It is easy to verify that $\text{Toy}[\Sigma]$ satisfies perfect correctness, and we claim that $\text{Toy}[\Sigma]$ is strongly unforgeable. First, note that for every list of public keys and a message, there exists exactly one valid multi-signature. Hence, for the adversary to win the $\mathbf{G}^{\text{suf-ms}}$ game or the $\mathbf{G}^{\text{euf-ms}}$ game, the input to the verification oracle must only contain one signature. However, when the input to the verification algorithm only contains one signature, the winning condition for existential and strong unforgeability is the same. Hence, $\text{Toy}[\Sigma]$ is existentially unforgeable if and only if it is strongly unforgeable.

To break the existential unforgeability of $\text{Toy}[\Sigma]$ the adversary is given access to a signing oracle to obtain Σ signatures from the honest signer, and to win the adversary must come up with a signature σ for some string of the form $(vk_1 \parallel \dots \parallel vk_n \parallel m)$ for which it did not receive a Σ signature from the signing oracle. Consequently, if the adversary is successful it must have also broken the existential unforgeability of Σ . This shows that $\text{Toy}[\Sigma]$ is as existentially unforgeable (and consequently as strongly unforgeable) as Σ is existentially unforgeable.

Note that it is necessary that Σ satisfies the deterministic property described above, or at least the weaker property that it is computationally infeasible to find two valid signatures for the same message and public key (even with access to the secret key). Otherwise, the adversary can trivially break the strong unforgeability of $\text{Toy}[\Sigma]$ by producing two different valid Σ signatures as partial signatures for some corrupt signer. In this case, however, $\text{Toy}[\Sigma]$ remains existentially unforgeable, showing a separation between existential unforgeability and strong unforgeability of multi-signatures.

B SUF of Underlying Plain Signature Schemes

In this section we prove that HBMS and mBCJ are strongly unforgeable against adversaries that don't exploit the fact that those protocols are interactive multi-signature schemes, or essentially treats them as a plain single signer digital signature scheme, where the signer is the group as a whole. Note that both HBMS and mBCJ are not strongly unforgeable against adversaries that corrupts some of the signers and exploits the interactive nature of the schemes. Hence, this section highlights the danger of integrating a multi-signature scheme (without verifying that it is strongly unforgeable) in a platform that is used for normal strongly unforgeable signature schemes, even if the multi-signatures are indistinguishable from the standard signatures.

DEFINING SECURITY OF THE UNDERLYING SCHEME. To define this security notion, which we call “non-interactive strong unforgeability,” we provide a game that an adversary plays against a multi-signature scheme, where the adversary is given a signing group and may query a signing oracle to obtain multi-signatures of that group for messages of the adversary's choice. The adversary wins if they can come up with a valid multi-signature for that group that was not obtained from the signing oracle. For the scheme to satisfy this security definition, it is required that efficient adversaries have a negligible winning probability for all n . See Figure 11 for a formal definition.

<p>Games $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{MS}]$</p> <p>INIT():</p> <ol style="list-style-type: none"> 1 For $i = 1, \dots, n$: 2 $(vk_i, sk_i) \leftarrow_s \text{MS.Kg}()$ 3 $Q \leftarrow \emptyset$ // message-signature pairs obtained legitimately 4 Return $(vk_i)_{i=1}^n$ 	<p>SIGNO(m):</p> <ol style="list-style-type: none"> 5 $\sigma \leftarrow \text{Exec}_{\text{MS}}((vk_i)_{i=1}^n), (sk_i)_{i=1}^n, m)$ 6 $Q \leftarrow Q \cup \{(m, \sigma)\}$ 7 Return σ <p>FIN(m, σ):</p> <ol style="list-style-type: none"> 8 If $\text{MS.Verify}((vk_i)_{i=1}^n, m, \sigma)$: 9 If $(m, \sigma) \notin Q$: 10 Return true 11 Return false
--	---

Fig. 11. Game used to define the “non-interactive strong unforgeability” of a multi-signature scheme MS.

B.1 Underlying Scheme of HBMS

We use the XIDL game (Figure 7) to prove that HBMS satisfies the non-interactive strong unforgeability definition. Since XIDL is hard in groups where the discrete log problem is hard (Lemma 4), this proves the non-interactive strong unforgeability of HBMS under the discrete log assumption.

Lemma 7 (XIDL \rightarrow NISUF of HBMS in the ROM). *Let \mathbb{G} be a group of prime order p with generator g . Let $\text{MS} = \text{HBMS}[\mathbb{G}, g]$ be the associated multi-signature scheme, where its hash functions are modeled as random oracles. Let \mathcal{A}_{ms} be an adversary against $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{MS}]$ that makes at most q_0, q_1, q_s queries to H_0, H_1 , and SIGNO respectively. Then, we can construct an adversary $\mathcal{A}_{\text{xidl}}$ against $\text{Gm}_{\mathbb{G}, g, 1, q_1+1}^{\text{xidl}}$ such that*

$$\text{Adv}_{\text{MS}}^{\text{ni-suf-ms}}(\mathcal{A}_{\text{ms}}) \leq (q_0 + q_s + 1) \text{Adv}_{\mathbb{G}, g, 1, q_1+1}^{\text{xidl}}(\mathcal{A}_{\text{xidl}}) + \frac{q_s(q_1 + q_s)}{p}$$

and the runtime of $\mathcal{A}_{\text{xidl}}$ is approximately the runtime of \mathcal{A}_{ms} .

PROOF IDEA FOR LEMMA 7. Suppose an adversary \mathcal{A}_{ms} break the non-interactive strong unforgeability of HBMS (i.e. wins the game in Figure 12). Then, given an aggregate public key \tilde{X} , they must provide a signature (s, z, T) for some message m such that $h^s g^z = T \tilde{X}^{H_1(T, \tilde{X}, m)}$, where $h = H_0(m, X_1, \dots, X_n)$. If \tilde{X} is an XIDL target and $H_1(T, \tilde{X}, m)$ is an XIDL challenge, then the element $z + s \cdot \text{DLOG}_g(h)$ wins the XIDL game, where the reduction can know $\text{DLOG}_g(h)$ since it programmed the random oracle H_0 . To simulate the signing oracle, we follow the standard procedure for simulating Schnorr signatures by first picking $c, z' \leftarrow_s \mathbb{Z}_p$, then letting $T = g^{z'} \tilde{X}^{-c}$, and then programming the random oracle $H_1(T, \tilde{X}, m) \leftarrow c$. Then, we can pick a random $s \leftarrow_s \mathbb{Z}_p$, and the triple $(s, T, z' - s \text{DLOG}_g(h))$ is a valid HBMS signature (where $h = H_0(m, X_1, \dots, X_n)$). Note that we cannot use an XIDL challenge for signing, since we have to pick it before deciding on the corresponding T . Thus, to win the XIDL game we need a signature on an (m, T) pair not used by the signing oracle, hence the need for a forgery.

If the forged signature indeed has a different (m, T) pair from any signing oracle response, then we can use it to win the XIDL; we call such a forgery a Type 1. However, it could be the case that the adversary obtained an HBMS signature (s, z, T) on the same (m, T) pair as some signing oracle response, (s^*, z^*, T) , which we call a Type 2 forgery. However, this means that $z + s \text{DLOG}_g(h) = z^* + s^* \text{DLOG}_g(h)$ for $s \neq s^*$, allowing the adversary to extract the discrete log of h .

To use this fact, we guess whether the adversary will output a Type 1 or Type 2 forgery. If we guess Type 1, we can win the XIDL game as described. If we guess type 2, we can set the XIDL input X to be one of the random oracles responses to $H_0(m, X_1, \dots, X_n)$ instead, and simulate the signing oracle by executing HBMS honestly with keys that the reduction generated. If we guessed the right query, which happens with probability inversely proportional to the maximum number of H_0 values for which the random oracle was programmed, the adversary learns the discrete log of X , and can trivially win the XIDL game.

Now, we make this idea more precise in a formal proof.

Proof (Lemma 7). In Figure 12 we construct an adversary $\mathcal{A}_{\text{xidl}}$ which plays the XIDL game by executing \mathcal{A}_{ms} and simulating perfectly the oracles that it has access to (as long as bad is not set to true). Without loss of generality, we assume that all random oracle queries of \mathcal{A}_{ms} are well-formed, since otherwise we can simply simulate their responses uniformly at random. We will now analyze its success probability.

<p>Game $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{HBMS}_{\mathbb{G},g,H_0,H_1,H_2}]$</p> <p>INIT():</p> <ol style="list-style-type: none"> 1 For $i = 1, \dots, n$: 2 $x_i \leftarrow_s \mathbb{Z}_p$; $X_i \leftarrow g^x$ 3 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i, X_1, \dots, X_n)}$ 4 $\tilde{x} \leftarrow \sum_{i=1}^n x_i \cdot H_2(i, X_1, \dots, X_n)$ 5 $Q \leftarrow \emptyset$ 6 Return (X_1, \dots, X_n) <p>SIGNO(m):</p> <ol style="list-style-type: none"> 7 $h \leftarrow H_0(X_1, \dots, X_n, m)$ 8 $s, r \leftarrow_s \mathbb{Z}_p$ 9 $T \leftarrow h^s g^r$ 10 $c \leftarrow H_1(T, \tilde{X}, m)$ 11 $z \leftarrow r + c \cdot \tilde{x}$ 12 $Q \leftarrow Q \cup \{(m, (s, T, z))\}$ 13 Return (s, T, z) <p>FIN(m, σ):</p> <ol style="list-style-type: none"> 14 If $(m, \sigma) \in Q$: Return false 15 Parse σ as (s, T, z) 16 $h \leftarrow H_0(X_1, \dots, X_n, m)$ 17 Return $[g^z h^s = T \cdot \tilde{X}^{H_1(T, \tilde{X}, m)}]$ 	<p>$\mathcal{A}_{\text{xidl}}^{\text{NewTarget, Challenge}}(X)$:</p> <ol style="list-style-type: none"> 1 // randomly selects whether to execute $\mathcal{B}_{\text{xidl}}$ or \mathcal{C}_{dl}, presented in Figure 13 2 $I \leftarrow_s \{0, 1, \dots, q_0 + q_s\}$ 3 If $I = 0$: 4 Return $\mathcal{B}_{\text{xidl}}^{\text{NewTarget, Challenge}}(X)$ 5 Else: 6 $x \leftarrow \mathcal{C}_{\text{dl}}(X, I)$ 7 If $x = \perp$ or $g^x \neq X$: Return \perp 8 $e \leftarrow \text{NewTarget}(\mathbb{1}_{\mathbb{G}})$ 9 $c \leftarrow \text{Challenge}(1, \mathbb{1}_{\mathbb{G}})$ 10 Return $(1, x \cdot c \cdot e)$
---	---

Fig. 12. Left: the non-interactive strong unforgeability of HBMS over a group \mathbb{G} with a generator g of order p . **Right:** the reduction algorithm used to prove Lemma 7, where the referenced subroutines are in Figure 13.

Now, suppose \mathcal{A}_{ms} wins $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{MS}]$ by returning some forgery (m, σ) where $\sigma = (s, z, T)$. Let $h = H_0(m, X_1, \dots, X_n)$ and $c = H_1(T, \tilde{X}, m)$, and note that $h^s g^z = T \tilde{X}^c$. Now consider the following cases:

Case 1 There was no signing oracle query that returned a signature (s', T', z') on m with $T = T'$. In this case, if $\rho = 0$, then the forgery allows $\mathcal{B}_{\text{xidl}}$ to win the XIDL game, as long as **bad** was not set to **true**. For each signing oracle query, since T is uniformly random independently of the previous execution of the algorithm, we have that **bad** is set to **true** with probability of at most $\frac{q_1 + q_s}{p}$, and thus across all signing oracle queries the probability that **bad** \leftarrow **true** is bounded above by $\frac{q_s(q_1 + q_s)}{p}$. If **bad** was not set to **true** in the execution of \mathcal{A}_{ms} and we have a forgery of the type corresponding to this case, then $C[T]$ would be initialized in the execution of \mathcal{A}_{ms} and thus **bad** would not be set in line 16.

Case 2 There was a signing oracle query that returned a signature (s', T', z') on m with $T = T'$. In this case, there was a call to $\tilde{H}_0(m, X_1, \dots, X_n)$ by the signing oracle (or the adversary). Suppose this call was the ℓ 's value programmed by \tilde{H}_0 . Then, if we guessed $\rho = \ell$, then \mathcal{C}_{dl} correctly finds the discrete log of X , allowing $\mathcal{A}_{\text{xidl}}$ to win the XIDL.

Thus, in both possible cases, we have that

$$\Pr[\text{Gm}_{\mathbb{G},g}^{\text{xidl}}(\mathcal{A}_{\text{xidl}})] \geq \frac{1}{q_0 + q_s + 1} \left(\Pr[\mathbf{G}_n^{\text{ni-suf-ms}}[\text{ms}](\mathcal{A}_{\text{ms}})] - \frac{q_s(q_1 + q_s)}{p} \right),$$

which is what we wanted to prove.

B.2 Underlying Scheme of mBCJ

To prove the non-interactive strong unforgeability of mBCJ, we once again use the chain reductions of Bellare and Dai [7]. This time we use the Identification Discrete Logarithm game (IDL) [30, 7], presented in Figure 14, as the underlying assumption in our proof, which Bellare and Dai show is hard to win whenever the discrete log problem is hard.⁹ In Lemma 8 we repeat their result regarding the difficulty of IDL under the

⁹ They also achieve tighter security bounds using the algebraic group model [25], but this is orthogonal to this paper.

DL assumption, and in Lemma 9 we prove the non-interactive strong unforgeability of mBCJ in the random oracle model assuming that IDL is hard. The combination of these lemmas is a proof of the non-interactive strong unforgeability of mBCJ in the ROM under the DL assumption.

Lemma 8 (DL \rightarrow IDL; Theorem 3.2 of [7]). *Let \mathbb{G} be a group of prime order p with a generator g , and let q be a positive integer. Let \mathcal{A}_{idl} be an adversary against $\text{Gm}_{\mathbb{G},g,q}^{\text{idl}}$. Then, an adversary \mathcal{A}_{dl} can be constructed so that*

$$\text{Adv}_{\mathbb{G},g,q}^{\text{idl}}(\mathcal{A}_{\text{idl}}) \leq \sqrt{q \cdot \text{Adv}_{\mathbb{G},g}^{\text{dl}}(\mathcal{A}_{\text{dl}})} + \frac{q}{p}.$$

Furthermore, the running time of \mathcal{A}_{dl} is approximately twice the running time of \mathcal{A}_{idl} .

We omit the proofs of Lemma 8 since it is non trivial and can be found in the referenced paper.

Lemma 9 (IDL \rightarrow NISUF of mBCJ in the ROM). *Let \mathbb{G} be a group of prime order p with a generator g_1 . Let $\text{MS} = \text{mBCJ}[\mathbb{G}, g_1]$ be the associated multi-signature scheme with its hash functions modeled as random oracles. Let \mathcal{A}_{ms} be an adversary against $\text{G}_n^{\text{ni-suf-ms}}[\text{MS}]$ that makes at most q_0, q_1, q_2, q_s queries to H_0, H_1, H_2 , and the signing oracle respectively. Then, we construct an adversary \mathcal{A}_{idl} against $\text{Gm}_{\mathbb{G},g_1,q_0+1}^{\text{idl}}$ such that*

$$\text{Adv}_{\text{MS}}^{\text{ni-suf-ms}}(\mathcal{A}_{\text{ms}}) \leq \text{Adv}_{\mathbb{G},g_1,q_0+1}^{\text{idl}}(\mathcal{A}_{\text{idl}}) + \frac{n(n-1) + (q_0 + q_s + 1)(q_0 + q_s)}{p^2} + \frac{q_2 + q_s + 1}{p}.$$

Furthermore, the runtime of \mathcal{A}_{idl} is approximately that of \mathcal{A}_{ms} .

PROOF IDEA FOR LEMMA 9. Suppose an adversary \mathcal{A}_{ms} can break the non-interactive strong unforgeability of mBCJ (i.e. wins the game in Figure 14 in the ROM). We will use \mathcal{A}_{ms} to construct an adversary \mathcal{A}_{idl} that wins the IDL game (Figure 14) when parameterized by the group used for mBCJ and the generator g_1 .

On input X , \mathcal{A}_{idl} sets up n random looking mBCJ public keys so that their aggregate public key is X . More specifically, it chooses $X_i \leftarrow \mathbb{G}$ for $i \in \{1, \dots, n-1\}$ and sets $X_n \leftarrow X \cdot \prod_{i=1}^{n-1} X_i^{-1}$. Now, it generates a secret key proof-of-possession for each X_i by choosing $c_i, z_i \leftarrow \mathbb{Z}_p$, $R_i \leftarrow g_1^{z_i} X_i^{-c}$, and programming the random oracle $H_1(X_i, R_i) \leftarrow c_i$. It now executes \mathcal{A}_{ms} with those public keys and proofs of possessions as its input, simulating its oracles as we describe below.

\mathcal{A}_{idl} simulates random oracle queries for $H_0(t_1, t_2, X, m)$ by querying the IDL CHALLENGE oracle with input t_2 , and returning the obtained challenge. It simulates H_1 random oracle queries by choosing an output uniformly at random. For $H_2(m)$ signing oracle queries, \mathcal{A}_{idl} also replies with uniformly random output $(g_{2,m}, h_{1,m}, h_{2,m})$, but it generates them in such a way that it knows the discrete log of $g_{2,m}$ and $h_{2,m}$ with respect to g_1 and the discrete log of $h_{1,m}$ with respect to X . More specifically, it chooses $\omega_{1,m}, \omega_{2,m}, \omega_{3,m} \leftarrow \mathbb{Z}_p$ and sets $h_{1,m} \leftarrow X^{\omega_{1,m}}$, $g_{2,m} \leftarrow g_1^{\omega_{2,m}}$ and $h_{2,m} \leftarrow g_1^{\omega_{3,m}}$, and stores the ω values. As we explain below, knowing those discrete logs allows \mathcal{A}_{idl} to use an mBCJ forgery to win the IDL game.

We simulate the signing oracle in a similar manner to how we simulated the proofs-of-possessions. On input m , we first generate the commitment parameters $(h_{1,m}, g_{2,m}, h_{2,m})$ that correspond to m . Next, we choose $\alpha_1, \alpha_2, s, c \leftarrow \mathbb{Z}_p$, set $t_1 \leftarrow g_1^{\alpha_1} h_{1,m}^{\alpha_2}$, and set $t_2 \leftarrow g_{2,m}^{\alpha_1} h_{2,m}^{\alpha_2} g_1^s \cdot X^{-c}$. Lastly we program the random oracle $H_0(t_1, t_2, X, m) \leftarrow c$, and output $(t_1, t_2, s, \alpha_1, \alpha_2)$. It is easy to verify that the outputted signature is valid with respect to m and the aggregate public key X , and that its distribution is identical to that of an honestly generated signature.

We are left to explain how to win the IDL game given a successful mBCJ forgery. Suppose \mathcal{A}_{ms} outputs a successful forgery $\sigma = (t_1, t_2, s, \alpha_1, \alpha_2)$ for the message m and the aggregate public key X . This means that

$$t_1 = g_1^{\alpha_1} h_{1,m}^{\alpha_2},$$

and

$$t_2 = g_{2,m}^{\alpha_1} h_{2,m}^{\alpha_2} g_1^s \cdot X^{-H_0(t_1, t_2, X, m)}.$$

Without loss of generality, assume that the random oracle value for $H_0(t_1, t_2, X, m)$ has already been set. If it was set by a random oracle query, then $H_0(t_1, t_2, X, m)$ is an IDL challenge that was generated on input t_2 . Thus, if we let $z \leftarrow \omega_{2,m} \alpha_1 + \omega_{3,m} \alpha_2 + s$, by the validity equation of t_2 it holds that $g_1^z = t_2 \cdot X^{H_0(t_1, t_2, X, m)}$, which allows the adversary to win the IDL game. Otherwise, the random oracle value for $H_0(t_1, t_2, X, m)$ was set by the signing oracle, which means that the signing oracle returned another valid signature $\sigma' =$

$(t_1, t_2, s', \alpha'_1, \alpha'_2)$ for the message m . Thus, since $g_1^{\alpha_1} h_{1,m}^{\alpha_2} = g_1^{\alpha'_1} h_{1,m}^{\alpha'_2}$ (by the validity equation of t_1), we know that

$$\alpha_1 + x\omega_{1,m}\alpha_2 = \alpha'_1 + x\omega_{1,m}\alpha'_2.$$

It is easy to verify that $\alpha_2 \neq \alpha'_2$ are distinct since σ and σ' are distinct,¹⁰ which means that the centered equation above allows \mathcal{A}_{idl} to extract x using simple field operations. Once it extract x (the discrete log of X), \mathcal{A}_{idl} can trivially win the IDL game. We now express this idea in a more formal way.

Proof (Lemma 9). In Figure 15, we construct an adversary \mathcal{A}_{idl} which plays the IDL game (parameterized with the group \mathbb{G} and the generator g_1) by executing \mathcal{A}_{ms} and simulating the oracles that it has access to. Without loss of generality, we assume that all random oracle queries of \mathcal{A}_{ms} are well-formed, since otherwise we can simply simulate their responses uniformly at random. We will now analyze its success probability.

First, note that if `bad` is not set to `true`, then \mathcal{A}_{idl} simulates perfectly the oracles that \mathcal{A}_{ms} has access to. The only places that `bad` can be set to true are the following:

- When \mathcal{A}_{idl} generates proofs-of-possession for the secret key, if there is a collision $(X_i, R_i) = (X_j, R_j)$ for some $i \neq j$. Since the choices of X_i and R_i are uniformly random in \mathbb{G}^2 (when not conditioned the value of X), the probability of collision is bounded above by $\frac{n(n-1)}{p^2}$.
- In `SIGNO`, if we need to program the random oracle for \widetilde{H}_0 for an input for which it has already been programmed. This can only happen if we choose t_1 and t_2 for which a random oracle value has been set. Since the maximum number of H_0 random oracle values set throughout the protocol is $q_0 + q_s + 1$, the probability of this happening is bounded above by $\frac{(q_0+q_s+1)(q_0+q_s)}{p^2}$.
- In \widetilde{H}_2 , if we choose $\omega_1 = 0$. Since ω_1 is chosen uniformly at random for each \widetilde{H}_2 execution, and \widetilde{H}_2 is executed at most $q_2 + q_s + 1$ times, the probability of this happening is bounded above by $\frac{q_2+q_s+1}{p}$.

Hence, the probability that `bad` is set to true is bounded above by $\frac{n(n-1)+(q_0+q_s+1)(q_0+q_s)}{p^2} + \frac{q_2+q_s+1}{p}$.

We are left to prove that if \mathcal{A}_{ms} wins, then \mathcal{A}_{idl} wins too. If \mathcal{A}_{ms} wins it outputs $(m, (t_1, t_2, s, \alpha_1, \alpha_2))$ satisfying

$$t_1 = g_1^{\alpha_1} h_1^{\alpha_2}$$

and

$$t_2 = g_2^{\alpha_1} h_2^{\alpha_2} g_1^s \cdot X^{-H_0[t_1, t_2, X, m]},$$

where $(g_2, g_1, h_2) = H_2[m]$. If $H_0[t_1, t_2, X, m]$ was first initialized on a \widetilde{H}_2 query, then it initialized $H_0^q[t_1, t_2, X, m]$. In this case $H_0[t_1, t_2, X, m]$ was the IDL challenge query number $H_0^q[t_1, t_2, X, m]$, and therefore returning $(H_0^q[t_1, t_2, X, m], s + \alpha_1\omega_2 + \alpha_2\omega_3)$ wins the IDL game.

If $H_0[t_1, t_2, X, m]$ was not initialized on a \widetilde{H}_2 query, then it was initialized on a `SIGNO` query. In this case, $\Sigma[t_1, t_2, m]$ has been set with $(\alpha'_1, \alpha'_2, s')$ such $(t_1, t_2, s', \alpha'_1, \alpha'_2)$ is a valid signature for the message m and aggregate key X . However, if \mathcal{A}_{ms} wins, the signature it outputted must be different then $(t_1, t_2, s', \alpha'_1, \alpha'_2)$ and consequently $\alpha_2 \neq \alpha'_2$. Furthermore, since $\omega_1 \neq 0$ (otherwise we would have set `bad` \leftarrow `true` in \widetilde{H}_2), \mathcal{A}_{idl} successfully extract x , the discrete log of X , using which it trivially wins the IDL game.

We have shown that the probability that `bad` is set to true is bounded above by $\frac{n(n-1)+(q_0+q_s+1)(q_0+q_s)}{p^2} + \frac{q_2+q_s+1}{p}$, and that if `bad` is not set to `true` and \mathcal{A}_{ms} wins then \mathcal{A}_{idl} wins. Furthermore, \mathcal{A}_{idl} makes at most $q_0 + 1$ queries to `CHALLENGE`. Hence,

$$\Pr[\text{Gm}_{\mathbb{G}, g_1, q+1}^{\text{idl}}(\mathcal{A}_{\text{idl}})] \geq \Pr[\mathbf{G}_n^{\text{ni-suf-ms}}[\text{ms}](\mathcal{A}_{\text{ms}})] - \frac{n(n-1) + (q_0 + q_s + 1)(q_0 + q_s)}{p^2} - \frac{q_2 + q_s + 1}{p},$$

which is what we wanted to prove.

C Proof of Lemma 5

Proof (Lemma 5). Let \mathbb{G} be a group of prime order p with a generator g and let `MS` = `MuSig` $[\mathbb{G}, g]$ be the associated multi-signature scheme with its hash functions modeled as random oracles. Let \mathcal{A}_{ms} be an adversary for the game $\mathbf{G}_n^{\text{suf-ms}}[\text{MS}]$ and assume the execution of \mathcal{A}_{ms} makes at most q_0, q_1, q_2, q_s queries to

¹⁰ If $\alpha_2 = \alpha'_2$, then by the validity equation of t_1 we know that $\alpha_1 = \alpha'_1$, and therefore by the validity equation of t_2 we know that $s = s'$ and thus $\sigma = \sigma'$.

$H_{\text{com}}, H_{\text{agg}}, H_{\text{sign}}$, and SIGNO_1 , respectively. Figure 16 constructs an adversary $\mathcal{A}_{\text{xidl}}$ against the XIDL game that simulates the $\mathbf{G}^{\text{suf-ms}}[\text{MS}]$ for \mathcal{A}_{ms} , and if \mathcal{A}_{ms} was successful it uses its output to win the XIDL game with high probability.

ANALYSIS OF $\mathcal{A}_{\text{xidl}}$: As we explained in the proof idea portion, in order to win the reduction must “guess right” when selecting ρ , which happens with probability of at least $\frac{1}{q_s+1}$, independently of the execution of \mathcal{A}_{ms} . Now, suppose the reduction picked the right value of ρ .

First, note that unless **bad** is set to **true**, the simulation is perfect. Furthermore, if \mathcal{A}_{ms} wins, then it must be the case that they provided signatures $(R_j, z_z)_{j=1}^\ell$ for a message m and public key \tilde{X} such that ℓ is more than the number of signing oracle sessions with the message m and aggregated key \tilde{X} that programmed the H_{sign} challenge. Hence, one of the challenges $H_{\text{sign}}(R_j, \tilde{X}, m)$ must be an XIDL challenge, and thus the reduction wins the XIDL game with the corresponding z_j .

Thus, the reduction may only fail if **bad** is set to **true**. To upper bound the probability that **bad** is set to **true**, we enumerate all the possible cases where it may happen and bound their probability conditioned on the fact that the reduction guessed ρ correctly:

- In the simulation of H_{com} , if the adversary finds a hash collision. At most $q_0 + n \cdot q_s$ values are added to the H_{com} dictionary over the execution of $\mathcal{A}_{\text{xidl}}$ and the reduction simulates H_{com} q_0 times. Hence, the probability of a collision is no more than $\frac{q_0(q_0+n \cdot q_s)}{p}$.
- In the simulation of H_{agg} , if the adversary finds a collision of aggregated keys or queries H_{sign} with an aggregated key before constructing it. An aggregated key is determined at most once for each SIGNO query, at most once for each H_{agg} query, and at most once after the execution of \mathcal{A}_{ms} when using the forgery to win the XIDL. Thus, the probability of finding a key collision is bounded above by $\frac{(q_s+q_1+1)^2}{p}$. Similarly, since H_{sign} can be initialized with an aggregated key that is not yet computed by the reduction only on calls to H_{sign} , we have that the probability that H_{sign} with an aggregated key before it is added to the key set is bounded above by $\frac{q_2(q_s+q_1+1)}{p}$.
- If in the simulation of SIGNO_2 the value $H_{\text{sign}}[R, \tilde{vk}, m]$ is already defined (for the R chosen uniformly at random at this execution). The probability of this happening at each execution of $\widetilde{\text{SIGNO}}_2$ is at most $\frac{q_2+q_s}{p}$, since the H_{sign} dictionary contains at most $q_s + q_2$ elements at each point in the execution, and thus the probability of this happening during the reduction is no more than $\frac{q_s(q_2+q_s)}{p}$.
- In the simulation of SIGNO_2 , the value $H_{\text{com}}[R_k]$ or $H_{\text{com}}[R_k]$ is already defined (for the R_k chosen uniformly at random at this execution). The probability of this happening at each execution of $\widetilde{\text{SIGNO}}$ is at most $n \cdot \frac{q_0+n \cdot q_s}{p}$ since at most $q_0 + n \cdot q_s$ values are added to the H_{com} dictionary over the execution of $\mathcal{A}_{\text{xidl}}$, and thus the probability of this happening during the reduction is at most $\frac{n \cdot q_s(q_0+n \cdot q_s)}{p}$.
- Lastly, note that if it guesses ρ correctly, then the reduction never has to set **bad** to **true** when simulating SIGNO_3 . This is because if we guessed $\rho = q_s + 1$, then we can simulate all signing oracle queries and **bad** would never be set at this point. Otherwise, to guess ρ correctly means that \mathcal{A}_{ms} outputs a forgery that falls into case 2 with session parameters corresponding to the ρ_{th} signing session, and therefore the third round of the ρ_{th} signing session is never executed.

Hence, the probability that **bad** is set to **true** if the reduction guessed ρ correctly is at most

$$\frac{q_0(q_0 + n \cdot q_s) + (q_s + q_1 + 1)^2 + q_2(q_s + q_1 + 1) + q_s(q_2 + q_s) + n \cdot q_s(q_0 + n \cdot q_s)}{p}.$$

Thus, if q is as defined in the lemma statement, we obtain that

$$\mathbf{Adv}_{\mathbb{G}, g, q_1, q_2}^{\text{xidl}}(\mathcal{A}_{\text{xidl}}) \geq \frac{\mathbf{Adv}_{\text{MS}}^{\text{suf-ms}}(\mathcal{A}_{\text{ms}}) - q/p}{q_s + 1}.$$

Rearranging the equation leads to the statement we wished to prove.

<p>$\mathcal{B}_{\text{xidl}}^{\text{NEWTARGET, CHALLENGE}}(X)$:</p> <ol style="list-style-type: none"> 1 $\text{bad} \leftarrow \text{false}$ // Abort whenever $\text{bad} \leftarrow \text{true}$ 2 $H_0, H_1, H_2, H_0^{\text{dl}}, C \leftarrow$ empty dictionaries 3 $q \leftarrow 0$ // tracks \widetilde{H}_1 queries 4 $X_1 \leftarrow X$ 5 For $i = 2, \dots, n$: 6 $x_i \leftarrow_{\\$} \mathbb{Z}_p$; $X_i \leftarrow g^x$ 7 For $i = 2, \dots, n-1$ 8 $\alpha_i \leftarrow_{\\$} \mathbb{Z}_p$ 9 $H_2[i, X_1, \dots, X_n] \leftarrow \alpha_i$ 10 $H_2[1, X_1, \dots, X_n] \leftarrow \text{NEWTARGET}(\prod_{i=2}^n X_i^{\alpha_i})$ 11 $\widetilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i, X_1, \dots, X_n)}$ 12 $(m, (s, z, T)) \leftarrow \mathcal{A}_{\text{ms}}^{\text{SIGNO}, \widetilde{H}_0, \widetilde{H}_1, \widetilde{H}_2}(X_1, \dots, X_n)$ 13 $\widetilde{H}_0(m, X_1, \dots, X_n)$ // ensure initialization 14 $\widetilde{H}_2(T, \widetilde{X}, m)$ // ensure initialization 15 $k \leftarrow H_0^{\text{dl}}[m, X_1, \dots, X_n]$ 16 If $C[T]$ uninitialized: $\text{bad} \leftarrow \text{true}$ 17 Return $(C[T], z + s \cdot k)$ <p>$\widetilde{\text{SIGNO}}(m)$:</p> <ol style="list-style-type: none"> 18 $z', c, s \leftarrow_{\\$} \mathbb{Z}_p$ 19 $T \leftarrow g^{z'} X^{-c}$ 20 If $H_1[T, \widetilde{X}, m]$ initialized: $\text{bad} \leftarrow \text{true}$ 21 $H_1[T, \widetilde{X}, m] \leftarrow c$ 22 $h \leftarrow H_0(X_1, \dots, X_n, m)$ 23 $k \leftarrow H_0^{\text{dl}}[X_1, \dots, X_n, m]$ 24 $z \leftarrow z' - k \cdot s$ 25 Return (s, z, T) <p>$\widetilde{H}_0(X'_1, \dots, X'_n, m)$:</p> <ol style="list-style-type: none"> 26 If $H_0[X'_1, \dots, X'_n, m]$ uninitialized: 27 $k \leftarrow_{\\$} \mathbb{Z}_p$ 28 $H_0[X'_1, \dots, X'_n, m] \leftarrow g^k$ 29 $H_0^{\text{dl}}[X'_1, \dots, X'_n, m] \leftarrow k$ 30 Return $H_0[X'_1, \dots, X'_n, m]$ <p>$\widetilde{H}_1(T, X', m)$:</p> <ol style="list-style-type: none"> 31 If $H_1[T, X', m]$ uninitialized: 32 If $X' = \widetilde{X}$: 33 $c \leftarrow \text{CHALLENGE}(1, T)$ 34 $q \leftarrow q + 1$ 35 $C[T] \leftarrow q$ 36 $H_1[T, X', m] \leftarrow c$ 37 Else: $H_1[T, X', m] \leftarrow_{\\$} \mathbb{Z}_p$ 38 Return $H_1[T, X', m]$ <p>$\widetilde{H}_2(i, X'_1, \dots, X'_n)$:</p> <ol style="list-style-type: none"> 39 If $H_2[i, X'_1, \dots, X'_n]$ uninitialized: 40 $H_2[i, X'_1, \dots, X'_n] \leftarrow_{\\$} \mathbb{Z}_p$ 41 Return $H_2[i, X'_1, \dots, X'_n]$ 	<p>$\mathcal{C}_{\text{dl}}(X, I)$:</p> <ol style="list-style-type: none"> 1 $H_0, H_1, H_2, Q \leftarrow$ empty dictionaries 2 $q \leftarrow 0$ // tracks \widetilde{H}_1 queries 3 $\ell \leftarrow 0$ // tracks signatures 4 For $i = 1, \dots, n$: 5 $x_i \leftarrow_{\\$} \mathbb{Z}_p$; $X_i \leftarrow g^x$ 6 $\widetilde{X} \leftarrow \prod_{i=1}^n X_i^{\widetilde{H}_2(i, X_1, \dots, X_n)}$ 7 $\widetilde{x} \leftarrow \sum_{i=1}^n x_i \cdot \widetilde{H}_2(i, X_1, \dots, X_n)$ 8 $(m, (s, z, T)) \leftarrow \mathcal{A}_{\text{ms}}^{\text{SIGNO}, \widetilde{H}_0, \widetilde{H}_1, \widetilde{H}_2}(X_1, \dots, X_n)$ 9 $h \leftarrow H_0(X_1, \dots, X_n, m)$ 10 If $h = X$: 11 For $i \in \{1, \dots, Q \}$: 12 $(s', z', T') \leftarrow Q[i]$ 13 If $h^s g^z = h^{s'} g^{z'}$ and $(s, z) \neq (s', z')$: 14 Return $(z - z')(s' - s)^{-1}$ 15 Return \perp <p>$\widetilde{\text{SIGNO}}(m)$:</p> <ol style="list-style-type: none"> 16 $h \leftarrow H_0(X_1, \dots, X_n, m)$ 17 $s, r \leftarrow_{\\$} \mathbb{Z}_p$; $T \leftarrow h^s g^r$; $c \leftarrow \widetilde{H}_1(T, \widetilde{X}, m)$ 18 $z \leftarrow r + c \cdot \widetilde{x}$ 19 $i \leftarrow i + 1$ 20 $Q[i] \leftarrow (s, T, z)$ 21 Return (s, T, z) <p>$\widetilde{H}_0(X'_1, \dots, X'_n, m)$:</p> <ol style="list-style-type: none"> 22 If $H_0[X'_1, \dots, X'_n, m]$ uninitialized: 23 $q \leftarrow q + 1$ 24 If $q = I$: $H_0[X'_1, \dots, X'_n, m] \leftarrow X$ 25 Else: $H_0[X'_1, \dots, X'_n, m] \leftarrow_{\\$} \mathbb{G}$ 26 Return $H_0[X'_1, \dots, X'_n, m]$ <p>$\widetilde{H}_1(T, X', m)$:</p> <ol style="list-style-type: none"> 27 If $H_1[T, X', m]$ uninitialized: 28 $H_1[T, X', m] \leftarrow_{\\$} \mathbb{Z}_p$ 29 Return $H_1[T, X', m]$ <p>$\widetilde{H}_2(i, X'_1, \dots, X'_n)$:</p> <ol style="list-style-type: none"> 30 If $H_2[i, X'_1, \dots, X'_n]$ uninitialized: 31 $H_2[i, X'_1, \dots, X'_n] \leftarrow_{\\$} \mathbb{Z}_p$ 32 Return $H_2[i, X'_1, \dots, X'_n]$
--	---

Fig. 13. Algorithms $\mathcal{B}_{\text{xidl}}$ and \mathcal{C}_{dl} , subroutines used by $\mathcal{A}_{\text{xidl}}$ of Figure 12 in the proof of Lemma 7.

Game $\text{Gm}_{\mathbb{G},g,q}^{\text{idl}}$	Game $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{mBCJ}_{\mathbb{G},g_1,H_0,H_1,H_2}]$
<p>INIT():</p> <ol style="list-style-type: none"> 1 $x \leftarrow_{\\$} \mathbb{Z}_p; X \leftarrow g^x; i \leftarrow 0$ 2 Return X <p>CHALLENGE(R): // at most q queries</p> <ol style="list-style-type: none"> 3 $i \leftarrow i + 1; R_i \leftarrow R$ 4 $c_i \leftarrow_{\\$} \mathbb{Z}_p$ 5 Return c_i <p>FIN(I, z):</p> <ol style="list-style-type: none"> 6 Return $[g^z = R_I \cdot X^{c_I}]$ 	<p>INIT():</p> <ol style="list-style-type: none"> 1 For $i = 1, \dots, n$: 2 $x_i \leftarrow_{\\$} \mathbb{Z}_p; X_i \leftarrow g^{x_i}$ 3 $r_i \leftarrow_{\\$} \mathbb{Z}_p; c_i \leftarrow H_1(X, g_1^{r_i}); z_i \leftarrow r_i + c_i \cdot x_i$ 4 $\pi_i \leftarrow (c_i, z_i)$ 5 $\bar{X} \leftarrow \prod_{i=1}^n X_i; \bar{x} \leftarrow \sum_{i=1}^n x_i$ 6 $Q \leftarrow \emptyset$ 7 Return $((X_1, \pi_1), \dots, (X_n, \pi_n))$ <p>SIGNO(m):</p> <ol style="list-style-type: none"> 8 $(g_2, h_1, h_2) \leftarrow H_2(m)$ 9 $r, \alpha_1, \alpha_2 \leftarrow_{\\$} \mathbb{Z}_p$ 10 $t_1 \leftarrow g_1^{\alpha_1} h_1^{\alpha_2}; t_2 \leftarrow g_2^{\alpha_1} h_2^{\alpha_2} g_1^r$ 11 $c \leftarrow H_0(t_1, t_2, \bar{X}, m); s \leftarrow r + \bar{x}c$ 12 $\sigma \leftarrow (t_1, t_2, s, \alpha_1, \alpha_2)$ 13 $Q \leftarrow Q \cup \{(m, \sigma)\}$ 14 Return σ <p>FIN(m, σ):</p> <ol style="list-style-type: none"> 15 If $(m, \sigma) \in Q$: Return false 16 Parse σ as $(t_1, t_2, s, \alpha_1, \alpha_2)$ 17 $(g_2, h_1, h_2) \leftarrow H_2(m); c \leftarrow H_0(t_1, t_2, \bar{X}, m)$ 18 Return $[t_1 = g_1^{\alpha_1} h_1^{\alpha_2} \text{ and } t_2 = g_2^{\alpha_1} h_2^{\alpha_2} g_1^s \cdot \bar{X}^{-c}]$

Fig. 14. Left: the Identification Discrete Logarithm (IDL) game in a group \mathbb{G} with a generator g of prime order p . **Right:** the non-interactive strong unforgeability game of mBCJ over a group \mathbb{G} with a generator g_1 of order p .

<p><u>$\mathcal{A}_{\text{idl}}^{\text{CHALLENGE}}(X)$:</u></p> <ol style="list-style-type: none"> 1 bad \leftarrow false // Abort whenever bad \leftarrow true 2 $H_0, H_1, H_2, H_0^q, H_2^{\text{dl}}, \Sigma \leftarrow$ empty dictionaries 3 $q \leftarrow 0$ // tracks \widetilde{H}_0 queries 4 For $i = 1, \dots, n - 1$: 5 $X_i \leftarrow \mathbb{G}$ 6 $X_n \leftarrow X \cdot \prod_{i=1}^{n-1} X_i^{-1}$ 7 For $i = 1, \dots, n$ // create proofs of possessions 8 $c_i, z_i \leftarrow \mathbb{Z}_p$; $R_i \leftarrow g_1^{z_i} X_i^{-1}$ 9 If $H_1(X_i, R_i)$ initialized: bad \leftarrow true 10 $H_1[X_i, R_i] \leftarrow c_i$ 11 $\pi_i \leftarrow (c_i, z_i)$ 12 $(m, (t_1, t_2, s, \alpha_1, \alpha_2)) \leftarrow \widetilde{\mathcal{A}}_{\text{ms}}^{\text{SIGNO}, \widetilde{H}_0, \widetilde{H}_1, \widetilde{H}_2}((X_1, \pi_1), \dots, (X_n, \pi_n))$ 13 $c \leftarrow H_0(t_1, t_2, X, m)$; $(g_2, h_1, h_2) \leftarrow \widetilde{H}_2(m)$ 14 $(\omega_1, \omega_2, \omega_3) \leftarrow H_2^{\text{dl}}[m]$ 15 If $H_0^q[t_1, t_2, X, m]$ initialized: 16 $I \leftarrow H_0^q[t_1, t_2, X, m]$ 17 Return $(I, s + \alpha_1 \omega_2 + \alpha_2 \omega_3)$ 18 Else: 19 $(\alpha'_1, \alpha'_2, s') \leftarrow \Sigma[t_1, t_2, m]$ 20 $x \leftarrow (\alpha_1 - \alpha'_1)(\alpha'_2 - \alpha_2)^{-1} \omega_1^{-1}$ 21 $r \leftarrow \mathbb{Z}_p$; $R \leftarrow g_1^r$; $c' \leftarrow \text{CHALLENGE}(R)$ 22 Return $(q + 1, r + x \cdot c')$ <p><u>$\widetilde{\text{SIGNO}}(m)$:</u></p> <ol style="list-style-type: none"> 23 $(h_1, g_2, h_2) \leftarrow \widetilde{H}_2(m)$ 24 $\alpha_1, \alpha_2, c, s \leftarrow \mathbb{Z}_p$ 25 $t_1 \leftarrow g_1^{\alpha_1} h_1^{\alpha_2}$; $t_2 \leftarrow g_2^{\alpha_1} h_2^{\alpha_2} g_1^s X^{-c}$ 26 If $H_0[t_1, t_2, X, m]$ initialized: bad \leftarrow true 27 $H_0[t_1, t_2, X, m] \leftarrow c$ 28 $\Sigma[t_1, t_2, m] \leftarrow (\alpha_1, \alpha_2, s)$ 29 Return $(t_1, t_2, s, \alpha_1, \alpha_2)$ 	<p><u>$\widetilde{H}_0(t_1, t_2, X', m)$:</u></p> <ol style="list-style-type: none"> 30 If $H_0[t_1, t_2, X', m]$ uninitialized: 31 If $X' = X$: 32 $q \leftarrow q + 1$; $H_0^q[t_1, t_2, X, m] \leftarrow q$ 33 $c \leftarrow \text{CHALLENGE}(t_2)$ 34 $H_0[t_1, t_2, X', m] \leftarrow c$ 35 Else: 36 $H_0[t_1, t_2, X', m] \leftarrow \mathbb{Z}_p$ 37 Return $H_0[t_1, t_2, X', m]$ <p><u>$\widetilde{H}_1(X', R)$:</u></p> <ol style="list-style-type: none"> 38 If $H_1[X', R]$ uninitialized: 39 $H_1[X', R] \leftarrow \mathbb{Z}_p$ 40 Return $H_1[X', R]$ <p><u>$\widetilde{H}_2(m)$:</u></p> <ol style="list-style-type: none"> 41 If $H_2[m]$ uninitialized: 42 $\omega_1, \omega_2, \omega_3 \leftarrow \mathbb{Z}_p$ 43 If $\omega_1 = 0$: bad \leftarrow true 44 $h_1 \leftarrow X^{\omega_1}$; $g_2 \leftarrow g_1^{\omega_2}$; $h_2 \leftarrow g_1^{\omega_3}$ 45 $H_2[m] \leftarrow (g_2, h_1, h_2)$ 46 $H_2^{\text{dl}}[m] \leftarrow (\omega_1, \omega_2, \omega_3)$ 47 Return $H_2[m]$
--	---

Fig. 15. Algorithms \mathcal{A}_{idl} , the reduction used in the proof of Lemma 9.

<pre> $\mathcal{A}_{\text{xidl}}^{\text{NEWTARGET, CHALLENGE}}(X)$: 1 bad \leftarrow false // abort whenever bad \leftarrow true 2 $H_{\text{agg}}, H_{\text{sign}}, H_{\text{com}} \leftarrow$ empty dictionary 3 $K \leftarrow \emptyset$ // tracks aggregated keys used 4 $\text{ctrs}, \text{ctr}_t, \text{ctrc} \leftarrow 0$ // tracks signing oracle queries, XIDL targets, XIDL challenges 5 $T_{\text{com}}, T_{\text{tar}}, T_{\text{chal}} \leftarrow$ empty dictionary // tracks H_{com} queries, XIDL targets, XIDL challenges 6 st \leftarrow empty list 7 $\rho \leftarrow_{\\$} \{1, \dots, q_s + 1\}$ // guess SIGNO_2 query for Case 2 forgery 8 $\text{Sim} \leftarrow \{(\text{SIGNO}_i)_{i=1}^3, \widetilde{H}_{\text{agg}}, \widetilde{H}_{\text{sign}}, \widetilde{H}_{\text{com}}\}$ 9 $(m, (X_i)_{i=1}^n, (R_j, z_j)_{j=1}^\ell) \leftarrow \mathcal{A}_{\text{ms}}^{\text{Sim}}(X)$ 10 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\text{agg}}(i, X_1, \dots, X_n)}$ 11 For $j = 1, \dots, \ell$: 12 $H_{\text{sign}}(R_j, \tilde{X}, m)$ // ensure initialization 13 If $T_{\text{chal}}[R_j, \tilde{X}, m]$ initialized: 14 Return $(T_{\text{chal}}[R_j, \tilde{X}, m], z_j)$ 15 Return \perp $\widetilde{\text{SIGNO}}_1()$: 16 $t \leftarrow_{\\$} \mathbb{Z}_p$ 17 $\text{ctrs} \leftarrow \text{ctrs} + 1, \text{st}_{\text{ctrs}.t} \leftarrow t$ 18 Return t $\widetilde{\text{SIGNO}}_2(j, k, (vk_i, t_i)_{i=1}^n, m)$: 19 $\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{\widetilde{H}_{\text{agg}}(i, vk_1, \dots, vk_n)}$ 20 If $j = \rho$: 21 $R_k \leftarrow_{\\$} \mathbb{G}$ 22 Else: 23 $z, c \leftarrow_{\\$} \mathbb{Z}_p; \text{st}_j.z \leftarrow z$ 24 $R_k \leftarrow g^z X^{-c}$ 25 For $i \in \{1, \dots, n\} \setminus \{k\}$: 26 If $T_{\text{com}}[t_i]$ initialized: $R_i \leftarrow T_{\text{com}}(t_i)$ 27 Else: 28 $R_i \leftarrow_{\\$} \mathbb{Z}_p$ 29 If $H_{\text{com}}[R_i]$ initialized: bad \leftarrow true 30 $H_{\text{com}}[R_i] \leftarrow t_i; T_{\text{com}}[t_i] \leftarrow R_i$ 31 $R \leftarrow \prod_{i=1}^n R_i; \text{st}_j.R \leftarrow R$ 32 $\text{st}_j.\tilde{vk} \leftarrow \tilde{vk}$ 33 If $H_{\text{sign}}[R, \tilde{vk}, m]$ initialized: bad \leftarrow true 34 $H_{\text{sign}}[R, \tilde{vk}, m] \leftarrow c$ 35 If $H_{\text{com}}[R_k]$ initialized: bad \leftarrow true 36 $H_{\text{com}}[R_k] \leftarrow \text{st}_j.t$ 37 Return R_k </pre>	<pre> $\widetilde{\text{SIGNO}}_3(j, R_1, \dots, R_n)$: 38 If $j = \rho$: bad \leftarrow true 39 Return $\text{st}_j.z$ $\widetilde{H}_{\text{agg}}(k, X_1, \dots, X_n)$: 40 If $H_{\text{agg}}[k, X_1, \dots, X_n]$ uninitialized: 41 For $i = 1, \dots, n$: 42 If $X_i \neq X$: $H_{\text{agg}}[i, X_1, \dots, X_n] \leftarrow_{\\$} \mathbb{Z}_p$ 43 $S \leftarrow \prod_{\{i: X_i \neq X\}} X_i^{H_{\text{agg}}[i, X_1, \dots, X_n]}$ 44 $e \leftarrow \text{NEWTARGET}(S)$ 45 $J \leftarrow \{j : X_j = X\}; j_{\text{max}} \leftarrow \max(J)$ 46 For $i \in J \setminus \{j_{\text{max}}\}$: 47 $e_i \leftarrow_{\\$} \mathbb{Z}_p; H_{\text{agg}}[i, X_1, \dots, X_n] \leftarrow e_i$ 48 $H_{\text{agg}}[j_{\text{max}}, X_1, \dots, X_n] \leftarrow e - \sum_{i \in J \setminus \{j_{\text{max}}\}} e_i$ 49 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\text{agg}}[i, X_1, \dots, X_n]}$ 50 If $\exists (R, m)$ such that $H_{\text{sign}}[R, \tilde{X}, m]$ is initialized: 51 bad \leftarrow true 52 If $\tilde{X} \in K$: 53 bad \leftarrow true 54 $K \leftarrow K \cup \{\tilde{X}\}$ 55 $\text{ctr}_t \leftarrow \text{ctr}_t + 1; T_{\text{tar}}[\tilde{X}] \leftarrow \text{ctr}_t$ 56 Return $H_{\text{agg}}[k, X_1, \dots, X_n]$ $\widetilde{H}_{\text{com}}(R)$: 57 If $H_{\text{com}}[R]$ uninitialized: 58 $H_{\text{com}}[R] \leftarrow_{\\$} \mathbb{Z}_p$; 59 If $T_{\text{com}}[H_{\text{com}}[R]]$ initialized: bad \leftarrow true 60 $T_{\text{com}}[H_{\text{com}}[R]] \leftarrow R$ 61 Return $H_{\text{com}}[R]$ $\widetilde{H}_{\text{sign}}(R, \tilde{X}, m)$: 62 If $H_{\text{sign}}[R, \tilde{X}, m]$ uninitialized: 63 If $T_{\text{tar}}[\tilde{X}]$ uninitialized: 64 $H_{\text{sign}}[R, \tilde{X}, m] \leftarrow_{\\$} \mathbb{Z}_p$ 65 Else: 66 $c \leftarrow \text{CHALLENGE}(T_{\text{tar}}[\tilde{X}], R)$ 67 $\text{ctrc} \leftarrow \text{ctrc} + 1$ 68 $T_{\text{chal}}[R, \tilde{X}, m] \leftarrow \text{ctrc}$ 69 Return $H_{\text{sign}}[R, \tilde{X}, m]$ </pre>
---	---

Fig. 16. The reduction algorithm used in the proof of Lemma 5.

D Statement and Proof of Lemma 10

Lemma 10 (underlying algorithm for the rewinding based reduction). *Let \mathbb{G} be a group of prime order p with generator g . Let $\text{MS} = \text{MuSig2}[\mathbb{G}, g]$ be the associated multi-signature scheme, where its hash functions are modeled as random oracles. Let \mathcal{A}_{ms} be an adversary against $\mathbf{G}^{\text{suf-ms}}[\text{MS}]$ that makes at most q_s queries to SIGNO_1 , q_h queries to each hash function, and uses a random tape of length $\rho_{\mathcal{A}}$. Let $q = 4q_h + 3q_s + 2$. If X, U_1, \dots, U_{4q_s} are AOMDL challenges and $h_1, \dots, h_q \leftarrow_{\$} \mathbb{Z}_p$, then with probability*

$$\text{acc}(\mathcal{B}) \geq \text{Adv}_{\text{MS}}^{\text{suf-ms}}(\mathcal{A}_{\text{ms}}) - \frac{4q^2}{p}$$

the adversary \mathcal{B} defined in Figure 17 makes at most q_s queries to the AOMDL discrete log oracle and outputs a tuple $(i_{\text{sign}}, i_{\text{agg}}, X_1, \dots, X_n, R, z, a_1, \dots, a_n)$ where $i_{\text{sign}}, i_{\text{agg}} \in \{1, \dots, q\}$, $i_{\text{sign}} < i_{\text{agg}}$, $X \in \{X_1, \dots, X_n\}$, $a_1, \dots, a_n \in \mathbb{Z}_p$, $\sum_{\{i: X_i=X\}} a_i = h_{i_{\text{agg}}}$, and

$$g^z = R \prod_{i=1}^n X_i^{a_i h_{i_{\text{sign}}}}.$$

Furthermore, the runtime of \mathcal{B} is approximately that of \mathcal{A}_{ms} .

Proof (Lemma 10). We will first prove that all the requirements hold whenever \mathcal{A}_{ms} and \mathcal{B} succeed, and then analyze its success probability.

First, note that \mathcal{B} makes a single DLOG query whenever \mathcal{A}_{ms} makes a query to the second round of the signing oracle SIGNO_2 , and does not query DLOG at any other time. Hence, it makes at most q_s queries to DLOG, as desired.

Now, we will ensure that ctrh does not exceed q . On each query \mathcal{A}_{ms} makes to H_{agg} and H_{sign} , ctrh is incremented by 1, and it is incremented by at most two on each H_{sign} query, resulting in at most $4 \cdot q_h$ incrementations from handling the adversary's random oracle queries. Additionally, for each SIGNO_2 query the adversary makes \mathcal{B} computes the aggregated key (incrementing the counter once) and b (which makes $\widetilde{H}_{\text{nonce}}$ increment the counter twice), resulting in at most $3 \cdot q_s$ incrementations to handle signing oracle queries. Lastly, to use a forgery \mathcal{B} computes the aggregated key, which may increment ctrh by one. Thus, ctrh does not exceed $4q_h + 3q_s + 1$, as desired.

If \mathcal{A}_{ms} succeeds, then it outputs more signatures than the number of signatures for the corresponding message m and public keys X_1, \dots, X_n produced by the signing oracle. In particular, for some j , the signing oracle did not return a signature for (m, X_1, \dots, X_n) with R_j as the R value. If bad is not set to **true**, then the adversary did not find a collision of aggregated keys for different lists of public keys, and therefore the signing oracle did not return a signature for (m, \tilde{X}) with R_j as the R value, and thus \mathcal{B} returns an output based on the forgery σ_j and not \perp .

For the other requirements, note that if \mathcal{A}_{ms} succeeds then it outputs a list of public keys containing X , and hence $X \in \{X_1, \dots, X_n\}$. Furthermore, by construction i_{agg} is the index such that

$$h_{i_{\text{agg}}} = \sum_{\{i: X_i=X\}} H_{\text{agg}}(i, X_1, \dots, X_n) = \sum_{\{i: X_i=X\}} a_i$$

as desired, and by construction $h_{i_{\text{sign}}} = H_{\text{sign}}(R_j, \tilde{X}, m)$. Since each of the signatures that \mathcal{A}_{ms} outputs is valid for the message m , it holds that

$$g^{s_j} = R \prod_{i=1}^n X_i^{a_i h_{i_{\text{sign}}}}.$$

Lastly, note that if $i_{\text{agg}} > i_{\text{sign}}$ then $H_{\text{sign}}[R_j, \tilde{X}, m]$ was set before $H_{\text{agg}}[k, X_1, \dots, X_n]$ was initialized for any k , and therefore bad is set to **true** when $H_{\text{sign}}[R_j, \tilde{X}, m]$ is initialized, meaning that \mathcal{B} does not succeed. Thus, if \mathcal{B} succeeds then $i_{\text{agg}} < i_{\text{sign}}$.

Now, we will lower bound success probability of \mathcal{B} . As we have shown, if \mathcal{A}_{MS} succeeds and bad is not set to **true**, then \mathcal{B} succeeds. Note that bad is set to **false** in two cases: when a new aggregated key has already been used in an H_{sign} query and when two aggregated keys collide. Each of those bad events can only happen when we initialize a new H_{agg} value, which happens at most once in calls to H_{agg} , SIGNO_2 , and on the output of \mathcal{A}_{ms} . Each time, the probability that those events happen are bounded above by $|H_{\text{sign}}|/p \leq q/p$ and $|K|/p \leq q/p$, resulting in a total probability bounded above by $2q^2/p$.

$\mathcal{B}^{\text{DLOG}}(X^*, U_1, \dots, U_{4q_s}, h_1, \dots, h_q):$ 1 $\text{bad} \leftarrow \text{false}$ // abort whenever $\text{bad} \leftarrow \text{true}$ 2 $H_{\text{agg}}, H_{\text{sign}}, H_{\text{nonce}} \leftarrow$ empty dictionary 3 $\text{ctrh} \leftarrow 0$ // counter for random oracle queries 4 $\text{ctrs} \leftarrow 0$ // counter for signing oracle queries 5 $Q \leftarrow \emptyset$ // tracks signing oracle responses 6 $K \leftarrow \emptyset$ // tracks aggregated keys used 7 $T_{\text{agg}}, T_{\text{sign}} \leftarrow$ empty dictionary // tracks indices in h for random oracle replies 8 $\rho_{\mathcal{A}} \leftarrow_{\$} \{0, 1\}^{R_{\mathcal{A}}}$ // random tape for \mathcal{A} 9 $\text{Sim} \leftarrow \{\text{SIGNO}_1, \text{SIGNO}_2, \widetilde{H}_{\text{agg}}, \widetilde{H}_{\text{sign}}, \widetilde{H}_{\text{nonce}}\}$ 10 $(m, (X_i)_{i=1}^n, (R_j, z_j)_{j=1}^\ell) \leftarrow \mathcal{A}_{\text{ms}}^{\text{sim}}(X; \rho_{\mathcal{A}})$ 11 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\text{agg}}(i, X_1, \dots, X_n)}$ 12 If $\exists j \in \{1, \dots, \ell\}$ such that $(R_j, \tilde{X}, m) \notin Q:$ 13 $\widetilde{H}_{\text{sign}}(R_j, \tilde{X}, m)$ // ensure initialization 14 $i_{\text{sign}} \leftarrow T_{\text{sign}}[R_j, \tilde{X}, m]$ 15 $i_{\text{agg}} \leftarrow T_{\text{agg}}[X_1, \dots, X_n]$ 16 For $i = 1, \dots, n:$ 17 $a_i \leftarrow H_{\text{agg}}(i, X_1, \dots, X_n)$ 18 Return $(i_{\text{sign}}, i_{\text{agg}}, X_1, \dots, X_n, R_j, z_j, a_1, \dots, a_n)$ 19 Return \perp $\widetilde{\text{SIGNO}}_1():$ 20 $\text{ctrs} \leftarrow \text{ctrs} + 1$ 21 $\hat{j} \leftarrow 4(\text{ctrs} - 1)$ 22 Return $U_{\hat{j}+1}, U_{\hat{j}+2}, U_{\hat{j}+3}, U_{\hat{j}+4}$ $\widetilde{\text{SIGNO}}_2(j, k, m, (vk_i, R_{i,1}, \dots, R_{i,4})_{i=1}^n):$ 23 $j' \leftarrow 4(\text{ctrs} - 1)$ 24 $(R_{k,1}, \dots, R_{k,4}) \leftarrow U_{\hat{j}+1}, \dots, U_{\hat{j}+4}$ 25 $\tilde{X} \leftarrow \prod_{i=1}^n vk_i^{H_{\text{agg}}(i, vk_1, \dots, vk_n)}$ 26 For $\ell = 1, \dots, 4:$ 27 $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ 28 $b \leftarrow \widetilde{H}_{\text{nonce}}(\tilde{X}, m, R_1, \dots, R_4)$ 29 $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$ 30 $Q \leftarrow Q \cup \{(R, \tilde{X}, m)\}$ 31 $c \leftarrow \widetilde{H}_{\text{sign}}(\tilde{X}, R, m)$ 32 // query dlog oracle for the partial signature: 33 $\alpha \leftarrow 0; \beta_1 \leftarrow c \cdot H_{\text{agg}}(k, vk_1, \dots, vk_n)$ 34 For $i = 2, \dots, 4q_s + 1:$ 35 $\beta_i \leftarrow 0$ 36 $(\beta_{j'+2}, \dots, \beta_{j'+5}) \leftarrow (b^0, \dots, b^3)$ 37 $z \leftarrow \text{DLOG}(\alpha, \beta_1, \dots, \beta_{4q_s})$ 38 Return (R, z)	$\widetilde{H}_{\text{agg}}(k, X_1, \dots, X_n):$ 39 If $H_{\text{agg}}[k, X_1, \dots, X_n]$ uninitialized: 40 If $X \in \{X_1, \dots, X_n\}:$ 41 $\text{ctrh} \leftarrow \text{ctrh} + 1$ 42 $v \leftarrow \max\{i: X_i = X\}$ 43 $s \leftarrow 0$ 44 For $i = 1, \dots, v - 1:$ 45 If $X_i = X:$ 46 $H_{\text{agg}}[i, X_1, \dots, X_n] \leftarrow_{\$} \mathbb{Z}_p$ 47 $s \leftarrow s + H_{\text{agg}}[i, X_1, \dots, X_n]$ 48 $H_{\text{agg}}[v, X_1, \dots, X_n] \leftarrow h_{\text{ctrh}} - s$ 49 $T_{\text{agg}}[X_1, \dots, X_n] \leftarrow \text{ctrh}$ 50 For $i = 1, \dots, n:$ 51 If $X_i \neq X: H_{\text{agg}}[i, X_1, \dots, X_n] \leftarrow_{\$} \mathbb{Z}_p$ 52 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\text{agg}}[i, X_1, \dots, X_n]}$ 53 If $\exists (R, m)$ such that $H_{\text{sign}}[R, \tilde{X}, m]$ is initialized: 54 $\text{bad} \leftarrow \text{true}$ 55 If $\tilde{X} \in K:$ 56 $\text{bad} \leftarrow \text{true}$ 57 $K \leftarrow K \cup \{\tilde{X}\}$ 58 Return $H_{\text{agg}}[k, X_1, \dots, X_n]$ $\widetilde{H}_{\text{nonce}}(\tilde{X}, R_1, \dots, R_4, m):$ 59 If $H_{\text{nonce}}[\tilde{X}, R_1, \dots, R_4, m]$ uninitialized: 60 $\text{ctrh} \leftarrow \text{ctrh} + 1$ 61 $H_{\text{nonce}}[\tilde{X}, R_1, \dots, R_4, m] \leftarrow h_{\text{ctrh}}$ 62 $b \leftarrow H_{\text{nonce}}[\tilde{X}, R_1, \dots, R_4, m]$ 63 $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$ 64 $\widetilde{H}_{\text{sign}}(R, \tilde{X}, m)$ // ensure initialization 65 Return $H_{\text{nonce}}[\tilde{X}, R_1, \dots, R_4, m]$ $\widetilde{H}_{\text{sign}}(R, \tilde{X}, m):$ 66 If $H_{\text{sign}}[R, \tilde{X}, m]$ uninitialized: 67 $\text{ctrh} \leftarrow \text{ctrh} + 1$ 68 $H_{\text{sign}}[R, \tilde{X}, m] \leftarrow h_{\text{ctrh}}$ 69 $T_{\text{sign}}[R, \tilde{X}, m] \leftarrow \text{ctrh}$ 70 Return $H_{\text{sign}}[R, \tilde{X}, m]$
---	--

Fig. 17. The algorithm from Lemma 10, which is used to prove the strong unforgeability of MuSig2. It is very similar to Figure 7 of [42].

Hence, \mathcal{B} succeeds with probability of at least $\Pr[\mathbf{G}_{\text{MS}}^{\text{suf-ms}}(\mathcal{A}_{\text{ms}})] - 2q^2/p$, which implies the probability bound in the lemma. We use the bound in the lemma as opposed to the tighter bound we obtained for compatibility with the proof of [42].