On the Existential and Strong Unforgeability of Multi-Signatures in the Discrete Log Setting

Sela Navot

Submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2025

Faculty Supervisor:

Stefano Tessaro

Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

On the Existential and Strong Unforgeability of Multi-Signatures in the Discrete Log Setting

Sela Navot

Faculty Supervisor:
Stefano Tessaro
Paul G. Allen School of Computer Science & Engineering

Digital signatures are typically required to be existentially unforgeable (EUF), ensuring that no adversary can produce a valid signature on a new message that has not been signed before. A stronger notion, strong unforgeability (SUF), also ensures that adversaries cannot forge new signatures on messages that have already been signed. These notions are well understood for plain signatures, but defining them for distributed multi-signature protocols, where multiple signers jointly produce a signature via an interactive protocol, is more challenging. While EUF has been studied for multi-signatures (using multiple competing definitions), there is no general definition for SUF, even though multi-signature protocols are often used to produce strongly unforgeable plain signatures.

This thesis introduces one-more unforgeability (OMUF) as a convenient way to model SUF in distributed signing protocols, and arrives at the following conclusions:

- MuSig and Bellare-Neven multi-signatures satisfy OMUF, even when the first signing round is pre-processed before the message to sign is known, but become completely insecure if the second signing round is also pre-processed.

- MuSig2 satisfies OMUF, which is important due to its widespread use in Bitcoin.

- The HBMS and mBCJ schemes do not satisfy OMUF, despite the fact that both schemes distributively generate strongly unforgeable plain signatures. Additionally, our analysis reveals an issue with the existential unforgeability of mBCJ, which does not contradict its original security proof.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thank Professor Stefano Tessaro for his encouragement to pursue research in cryptography, and for his continuous guidance and support. His mentorship has been invaluable. I have been fortunate to work with and learn from many incredible researchers, including Professor Nirvan Tyagi, Hanjun Li, and many others inside and outside of the cryptography lab. Thank you all for the inspiration!

PREFACE

This thesis is primarily based on materials from the following works, which have been integrated and revised to form a coherent narrative.

[27] Sela Navot and Stefano Tessaro. "One-More Unforgeability for Multi- and Threshold Signatures," published in the proceedings of Asiacrypt 2024.

[26] Sela Navot. "Insecurity of MuSig and Bellare-Neven Multi-Signatures with Delayed Message Selection," Cryptology ePrint Archive, Report 2024/437.

The technical contributions presented here are drawn from these papers, except for Section 5.2, which is published here for the first time. Portions of the introduction have also been adapted from these works, but it has been expanded to align with the structure of the thesis and to incorporate new developments.

Section 1

INTRODUCTION

There has been growing interest in protocols for the distributed generation of digital signatures, in the form of *threshold signatures* [14, 15] and *multi-signatures* [20]. While distributed signing protocols have been studied for decades, their recent widespread use has been driven by applications in blockchain ecosystems, such as digital wallets [18], and to enforce the need for multiple signatures to authorize a transaction. Threshold signatures and multi-signatures are also at the center of standardization efforts by NIST [25]. In this work, we focus on the formal security definitions of multi-signatures, and analyzing the security of practical schemes.

What are multi-signatures. Multi-signature schemes are a specific type of distributed signing protocol, which allows a group of signers to provide a succinct joint signature for an agreed upon message. Each signer generates 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. Multi-signature schemes are useful if the size of the multi-signature remains constant regardless of the number of signers, allowing efficient verification and storage (or broadcasting) of signatures in settings with many signers.

Signature security: existential and strong unforgeability. Plain (non-distributed) digital signatures are typically required to be unforgeable under a chosen message attack. *Existential unforgeability (EUF)* requires that no efficient adversary can forge a valid signature on any new message (not previously signed) under the target verification key, even after seeing signatures for messages of its choice. *Strong unforgeability (SUF)* is a stronger notion, which also forbids the adversary from producing a different valid signature on a message for which it has already seen a signature. Since distributed signing protocols aim to replicate the security guarantees of their underlying signature schemes, these distinctions become important when analyzing the security of multi-signatures.

Security definitions for multi-signature. 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, which are the focus of this paper), and executions are subject to adversarial corruptions. Often, the adversary can not only corrupt a subset of the signers but also control communication between them—this is the case for a common model where inter-signer communication is mediated by a proxy. An additional source of complexity is that some schemes allow some of their signing rounds to be *pre-processed*, which means being completed before the message to be signed or the identity of all the signers have been determined. In other schemes, as we show, pre-processing of some rounds is possible but compromises the security of the scheme.

This makes it hard to define *when* a signature on a message has been issued, and, in turn, to formalize a notion of existential unforgeability. A number of works sidestep this question by considering a message signed as long as a signing session *started* on it, including, for example, all existing security proofs for Bellare-Neven multi-signatures and MuSig [7, 24, 12, 5, 36]. However, a security proof that uses such a definition does not rule out adversaries who forge a multi-signature for a message after seeing the output of some, but not all, of the signing rounds for that message. Such a definition also does not work for analyzing schemes that support pre-processing, since the message to sign is not yet determined when the signing session begins. In other works, the definition is tailored to the specific structure of the scheme (for example, in the analysis of MuSig2 [30]) or a very limited class of schemes, as in [4, 10], where Bellare et al. put forward a hierarchy of security notions for partially non-interactive threshold signatures.

Strong unforgeability for multi-signatures. This work considers a further challenge in the study of security definitions for multi-signatures, 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 [34, 32], 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.

Strong unforgeability is particularly relevant in the context of blockchain ecosystems. In Bitcoin, multi-signatures are routinely used to generate Schnorr signatures [29, 41]. This usage was made official earlier this year when the standard cryptography library maintained by the Bitcoin Core Project, *secp256k1*,[1] added an implementation of MuSig2 [30]. However, lack of strong unforgeability of plain signatures has been associated with costly transaction malleability attacks [13, 1] and was a major motivation for Bitcoin's adoption of the strongly unforgeable Schnorr Signatures [40, 22, 41]. Thus, it is prudent not to break the strong unforgeability guarantee when the signatures are issued in a distributed setting, further motivating our work. Jumping ahead again, we show that MuSig2 is indeed strongly unforgeable, and thus does not break that guarantee.

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, and it is not always clear how to do this. In the setting of threshold signatures, for example, Bellare et al. [4, 10] 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 the second signing rounds, but this is not a property we expect a protocol to have, and no general definition is known nor is it known how to extend this definition to multi-signatures.

This work: 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 Poitncheval and Stern [31, 33], 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 Definition.** We formalize the notion of one-more unforgeability for multi-signature

[1] `https://github.com/bitcoin-core/secp256k1`

schemes, and argue that it is a useful way to model strong unforgeability. Our definition is modular and can be used for schemes with different structures, including those that support pre-processing. Additionally, our definition considers a message signed only when all interactive signing rounds have been completed, and thus it implies unforgeability against adversaries who forge a multi-signature after completing some, but not all, of the interactive signing rounds for the corresponding message.

- **Proofs and Attacks for MuSig and Bellare-Neven Multi-Signatures.** Both MuSig [24] and its predecessor Bellare-Neven multi-signatures [7] employ a three-round signing process in which the message to be signed is only used in the final round. We first prove that these protocols satisfy One-More Unforgeability, even when the first signing round is pre-processed before the message to be signed is determined. In addition to establishing strong unforgeability, this is the first proof that rules out forgeries of signatures on messages that were used for some, but not all, of the signing rounds. Additionally, ours is the first security proof for the case when the first signing round is pre-processed.

  However, even though the signing protocol of these schemes seemingly allows pre-processing of the second rounds, our analysis demonstrates that this approach compromises security. In that setting, we present a practical polynomial-time attack that fully breaks both existential and strong unforgeability in less than a second on a laptop.

- **One-More Unforgeability of MuSig2.** We prove that MuSig2 [30] satisfies One-More Unforgeability. As we mentioned, this positive result is of practical interest due to widespread use of MuSig2 in Bitcoin applications, and in light of a long history of attacks resulting from lack of strong unforgeability.

- **Attacks for HBMS and mBCJ.** We show that the HBMS [5] and mBCJ [16] multi-signature schemes do not satisfy One-More Unforgeability using a polynomial time attack based on the algorithm of Benhamouda et al. [11] to solve the ROS problem [35]. This is despite the fact that both scheme generate strongly unforgeable plain signatures, which we prove.

Our analysis also reveals a subtle issue with the existential unforgeability of mBCJ: 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. While this attack does not contradict the original security proof of [16], since their security definition does not cover such forgeries, it renders the scheme not existentially unforgeable according to our and other well accepted definitions.

A summary of our analysis of existing schemes is presented in Table 1.1.

Scheme	**Existential Unforgeability (EUF)**	**One-More Unforgeability (OMUF)**
Bellare-Neven [7]	✓	✓
(with delayed message selection)	(✗)	(✗)
MuSig [24]	✓	✓
(with delayed message selection)	(✗)	(✗)
MuSig2 [30]	✓	✓
HBMS [5]	✓	✗
mBCJ [16]	✗	✗

Table 1.1: Summary of unforgeability properties of existing schemes.

Section 2

PRELIMINARIES

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 $\leftarrow\!\!\$$ for randomized assignment. In particular, $x \leftarrow\!\!\$ S$ denotes sampling an element uniformly at random from a finite set S and $x \leftarrow\!\!\$ \mathcal{A}(x_1, \dots)$ 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.

GAMES FRAMEWORK. We use the game playing framework of Bellare and Rogaway [9] 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 $\mathrm{Gm}(\mathcal{A})$, $\mathcal{A}$ 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 $\Pr[\mathrm{Gm}(\mathcal{A})]$ as a shorthand for $\Pr[\mathrm{Gm}(\mathcal{A}) = \mathsf{true}]$ 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* $\mathrm{Gm}^{\mathrm{asmp}}_{\mathrm{params}}$ *as*

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

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

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

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

The scheme S *is* DEFN*-secure if* $\mathbf{Adv}^{\text{defn}}_{\text{S}}(\mathcal{A})$ *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.

HARDNESS ASSUMPTIONS. This work focuses on schemes that their security relying on the discrete log assumption. That is, schemes that are secure when instantiated over prime ordered groups $\mathbb{G}$ satisfying the following assumption: is hard to find x given g^x, where g is a fixed generator of the group. We define the discrete logarithm assumption formally in Figure 2.1.

Historically, to prove the security of Schnorr-like schemes from the discrete log assumption, reductions made use of rewinding techniques to reduce breaking the scheme to the discrete log assumption, and then forking lemmas to analyze its success probability. Such a reduction is inherently non-tight, in the sense that given an adversary that breaks a scheme with probability p after making h hash function queries, the probability of a reduction winning the discrete log assumption is of the order of $\frac{p^k}{h^\ell}$ for some positive integers k and ℓ that are sometimes as big as 4 and 2 in the context of multi-signatures. Such reductions provide little guarantee when considering the parameters used in practice (e.g. when using 256-bit groups), and are complex and error prone.

Consequently, Bellare and Dai introduce the IDL and the XIDL assumptions (Figure 2.1) that simplify the analysis of multi-signature schemes [5]. These assumption are proven to hold when the discrete log assumption holds but are simpler to use.

Proving the security of IDL and XIDL from the hardness of the discrete log assumption requires the use of rewinding and a non-tight reduction, but once the security of the IDL and XIDL games are established then the security proofs for concrete multi-signature schemes are simpler, tighter,

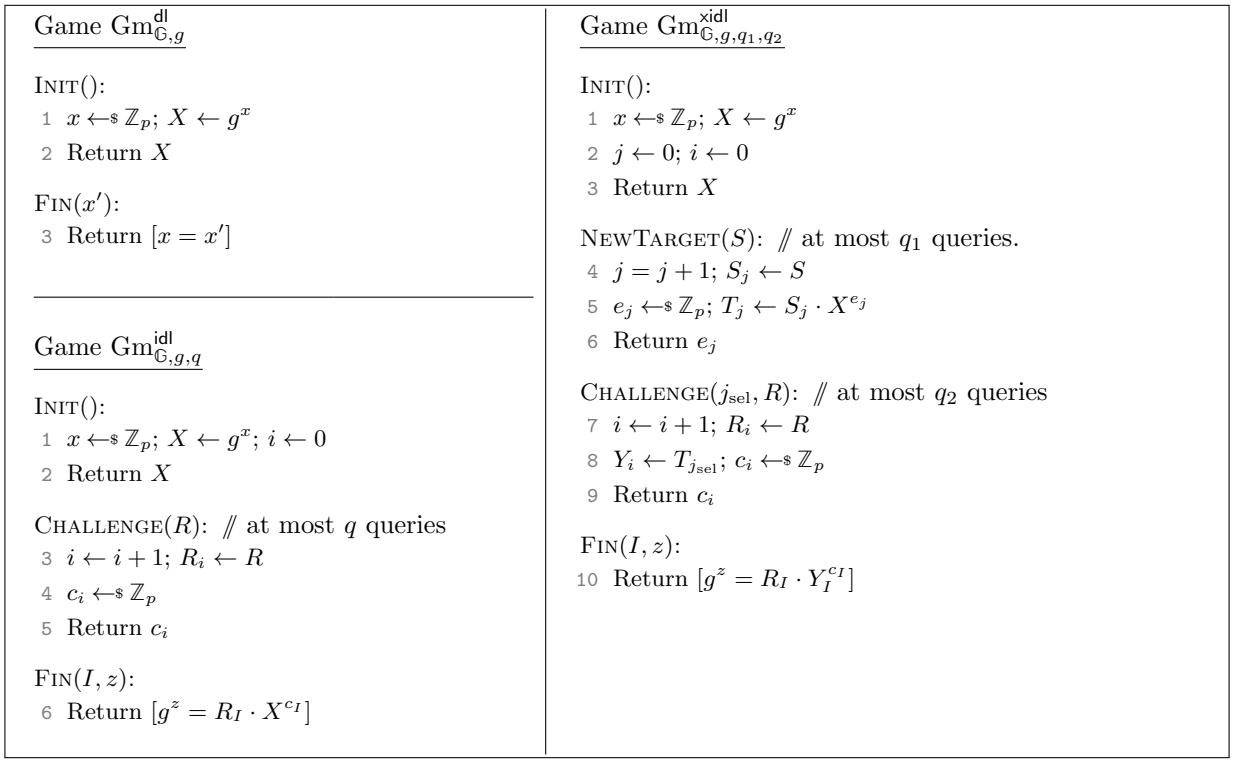

Figure 2.1: The Discrete Log (DL), Identification Discrete Log (IDL), and the Random Target Identification Logarithm (XIDL) games in a group $\mathbb{G}$ with a generator g of prime order p.

and more modular. Lemmas 2.0.1 and 2.0.2 show the concrete security of the IDL and the XIDL assumption. We do remark that Bellare and Dai provide a tight reduction of IDL and XIDL to the DL assumption which avoids rewinding in the idealized Algebraic Group Model [17], which suggests that they are hard to break in groups that are used for multi-signatures in practice, but these results are somewhat orthogonal to our work.

The only other hardness assumption we use, apart from DL or its equivalent IDL and XIDL assumptions, is in the proof of the strong unforgeability of MuSig2. Our proof, like the original existential unforgeability proof of MuSig2 [30], relies on the stronger Algebraic One-More Discrete Log assumption. We present that assumption in Section 6, and note that it is not known to be equivalent to the discrete log assumption.

Lemma 2.0.1 (DL $\rightarrow$ IDL; Theorem 3.2 of [5])**.** *Let $\mathbb{G}$ be a group of prime order p with a generator g, and let q be a positive integer. Let $\mathcal{A}_{\mathsf{idl}}$ be an adversary against $\mathrm{Gm}^{\mathsf{idl}}_{\mathbb{G},g,q}$. Then, an adversary*

$\mathcal{A}_{\mathsf{dl}}$ can be constructed so that

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

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

Lemma 2.0.2 (DL $\rightarrow$ XIDL; a combination of Theorems 3.2 and 3.4 of [5])**.** *Let $\mathbb{G}$ be a group of order p with generator g. Let q_1, q_2 be positive integers. Let $\mathcal{A}_{\mathsf{xidl}}$ be an adversary against $\mathrm{Gm}^{\mathsf{xidl}}_{\mathbb{G},g,q_1,q_2}$. Then, an adversary $\mathcal{A}_{\mathsf{dl}}$ can be constructed so that*

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

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

We omit the proofs of Lemmas 2.0.1 and 2.0.2 since they are non-trivial and can be found in the referenced paper.

The random oracle model. Our security proofs operate in the random oracle model [8], where hash functions are modeled as idealized random functions that can be queried by all parties. Although the random oracle model is an unrealistic assumption regarding the security of hash functions, it remains the only known way to prove security of Schnorr-like signatures (and all of the multi-signature schemes analyzed in this paper), and is widely accepted as a useful abstraction. A scheme that is secure in the random oracle model can be seen as secure against adversaries that do not utilize any potential weakness of the hash functions used by the scheme.

Definitions and security for plain digital signatures. A digital signature scheme Σ is a collection of algorithms $\Sigma.\mathsf{KeyGen}$, $\Sigma.\mathsf{Sign}$, and $\Sigma.\mathsf{Verify}$ with the following intent:

Key generation: The randomized algorithm $\Sigma.\mathsf{KeyGen}$ is used for key generation. It takes no input apart from the scheme parameters and outputs a secret-public key pair.

Signing: The algorithm $\Sigma.\mathsf{Sign}$ specifies the signing procedure. It takes a secret signing key and a message as input and outputs a digital signature. In some schemes this algorithms is randomized.

Verification: The algorithm $\Sigma.\mathsf{Verify}$ takes a public key, a signature, and a message as input and returns a boolean value signifying whether the signature is valid.

Correctness of a digital signature scheme requires that running the verification algorithm on a validly generated signature returns true. More formally, if $(vk, sk) \leftarrow\!\!\$\ \Sigma.\mathsf{KeyGen}$ then for every message m that is supported by the scheme it holds with probability 1 that

$$\Sigma.\mathsf{Verify}(vk, m, \Sigma.\mathsf{Sign}(sk, m)) = \mathsf{true}.$$

The standard security notion that plain (single signer) digital signatures are expected to satisfy is existential unforgeability, and often the stronger strong unforgeability. Figure 2.2 presents the game definition of these security properties.

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 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.

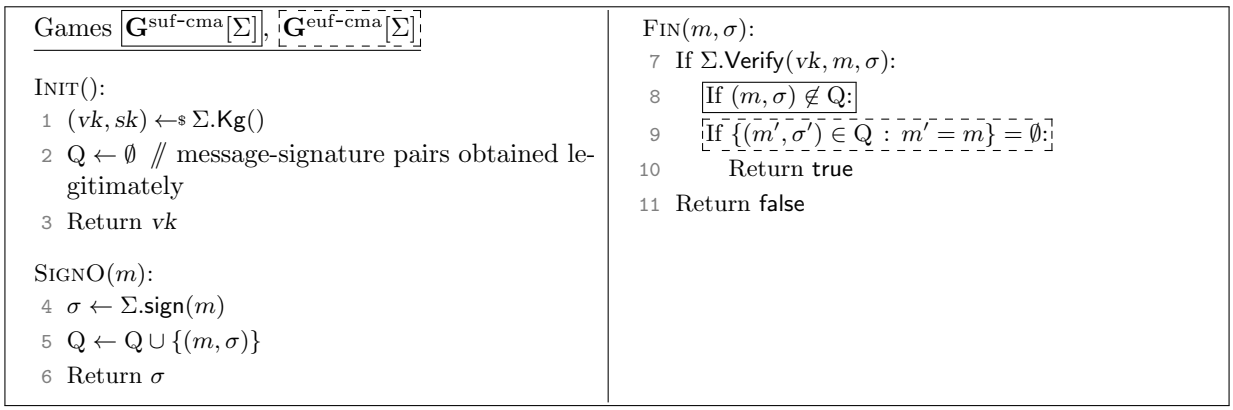

Figure 2.2: Games used to define the existential and strong unforgeability of a single signer digital signature scheme Σ. The definition for strong unforgeability, $\mathbf{G}^{\text{suf-cma}}$, contains all but the dashed box. The definition for existential unforgeability, $\mathbf{G}^{\text{euf-cma}}$, contains all but the solid box.

Section 3

SECURITY DEFINITIONS FOR MULTI-SIGNATURES

3.1 Specifications and Usage

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 [7], 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 [24] and MuSig2 [30] 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 [19, 30]) 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 $\mathsf{MS.Kg}$, $(\mathsf{MS.Sign}_r)_{r=1}^{\mathsf{MS.nr}}$, and $\mathsf{MS.Verify}$, where nr is the number of signing rounds specified by the scheme. A scheme also specifies the last interactive round $\mathsf{MS.lir}$, 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 $\mathsf{MS.KeyAgg}$ algorithm accompanied by $\mathsf{MS.AggVer}$ for key aggregation and for verification using the aggregated key. The intent of the algorithms is as follows:

Key generation: The randomized algorithm $\mathsf{MS.Kg}$ 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 $(\mathsf{MS.Sign}_r)_{r=1}^{\mathsf{MS.nr}}$ specifies the signing procedures to be run by each signing party, where $\mathsf{MS.nr}$ (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 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 $\mathsf{Sign}_{\mathsf{MS.nr}}$. These algorithms may be randomized.

Key aggregation: If the scheme supports key aggregation, the algorithm $\mathsf{MS.KeyAgg}$ 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 $\mathsf{MS.Verify}$ 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 $\mathsf{MS.AggVer}$ 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 $\mathsf{MS.Verify}((vk_i)_{i=1}^n, m, \sigma) = \mathsf{MS.AggVer}(\mathsf{MS.KeyAgg}((vk_i)_{i=1}^n), m, \sigma)$. Hence, without loss of generality, we will only consider $\mathsf{MS.Verify}$ in the correctness and security definitions.

The signers maintain a state st which may change throughout the protocol. In particular, using the convention from [5], each signer i has a long-standing secret key $i.\mathsf{st}.sk$ and public key $i.\mathsf{st}.vk$ as well as information associated with each signing session s that they participate in, denoted by $i.\mathsf{st}_s$. The session state includes $\mathsf{st}_s.n$, $(\mathsf{st}_s.vk_j)_{j=1}^{\mathsf{st}_s.n}$, $\mathsf{st}_s.m$, $\mathsf{st}_s.\mathrm{rnd}$, and $\mathsf{st}_s.\mathrm{me}$ 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

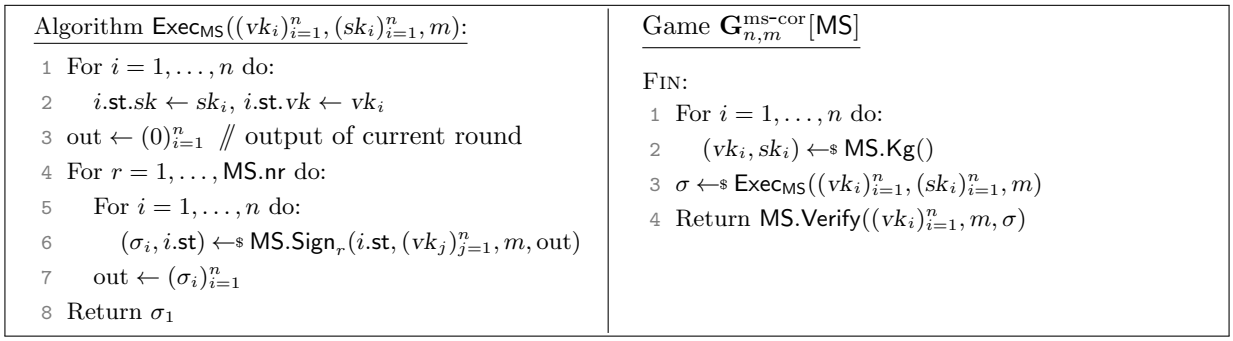

Figure 3.1: **Left:** an honest execution of the signing protocol of a multi-signature scheme MS. Note that signing rounds may only use a subset of the provided input. **Right:** a game defining the correctness of a scheme. A scheme satisfies perfect correctness for a natural number n if $\Pr[\mathbf{G}_{n,m}^{\text{ms-cor}}[\mathsf{MS}]] = 1$ for each supported message m.

---

requests to run the algorithm Sign_r for a session s if $r \neq \mathsf{st}_s.\text{rnd} + 1$. The state may also include other information such as the output of previous signing rounds or the discrete log of a nonce.

Figure 3.1 describes an honest execution of a multi-signature scheme and provides a correctness definition.

3.2 Existential and Strong Unforgeability

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 3.2, which compares it with the definition of existential unforgeability. Note that the only difference between the existential and strong unforgeability games is the 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 [7, 24, 12, 5]) 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, $\mathsf{MS.lir}$. 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, [30, 37]). 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 to help demonstrates the difference and separation between existential and strong unforgeability. 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 Toy[Σ] with a single interactive signing round as follows:

Key generation: Each party runs $(vk, sk) \leftarrow\!\!\$\ \Sigma.\mathsf{Kg}$, 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 $\sigma_k \leftarrow \Sigma.\mathsf{Sign}(sk, vk_1 \| \ldots \| vk_n \| m)$, where $\|$ denotes string concatenation. The final multi-signature is a list of all partial signatures $(\sigma_1, \ldots, \sigma_n)$.

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

It is easy to verify that Toy[Σ] satisfies perfect correctness, and we claim that Toy[Σ] 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, Toy[Σ] is existentially unforgeable if and only if it is strongly unforgeable.

To break the existential unforgeability of Toy[Σ] 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 $(vk_1\| \ldots \|vk_n\|m)$ 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 Toy[Σ] 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 Toy[Σ] by producing two different valid Σ signatures as partial signatures for some corrupt signer. In this case, however, Toy[Σ] remains existentially unforgeable, showing a separation between existential unforgeability and strong unforgeability of multi-signatures.

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.

Games $\boxed{\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]}$, $\boxed{\mathbf{G}^{\text{euf-ms}}[\mathsf{MS}]}$

INIT():
1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{MS.Kg}()$ // generates keys and initializes state
2 Q $\leftarrow$ Empty Dictionary
3 Return vk

SIGNO$_j(s,$ SOME SUBSET OF $\{m, k, (vk_i)_{i=1}^n, \text{out}\})$:
4 // An oracle for each $j \in \{1, \ldots, \mathsf{MS.nr}\}$
5 $\sigma \leftarrow\!\!\$\ \mathsf{MS.Sign}_j(\text{input of SIGNO}_j)$
6 If $\sigma = \perp$: return $\perp$
7 If $j = \mathsf{MS.lir}$: // on last interactive signing round
8 If Q$[((\mathsf{st}_s.vk_i)_{i=1}^n, \mathsf{st}_s.m)]$ uninitialized:
9 Q$[((\mathsf{st}_s.vk_i)_{i=1}^n, \mathsf{st}_s.m)] \leftarrow 1$
10 Else:
11 Q$[((\mathsf{st}_s.vk_i)_{i=1}^n, \mathsf{st}_s.m)] \leftarrow$ Q$[((\mathsf{st}_s.vk_i)_{i=1}^n, \mathsf{st}_s.m)] + 1$
12 Return σ

FIN$(k, (vk_i)_{i=1}^n, m, (\sigma_j)_{j=1}^\ell)$:
13 If $vk_k \neq vk$: return false
14 If $\sigma_i = \sigma_j$ for some $i \neq j$: return false
15 For $j = 1, \ldots, \ell$ do:
16 If not $\mathsf{MS.Verify}((vk_i)_{i=1}^n, m, \sigma_j)$:
17 Return false
18 $\boxed{\text{If Q}[(vk_i)_{i=1}^n, m] \text{ initialized and Q}[(vk_i)_{i=1}^n, m] \geq \ell\text{:}}$
19 $\boxed{\text{If Q}[(vk_i)_{i=1}^n, m] \text{ initialized:}}$
20 Return false
21 Return true

Figure 3.2: 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.

Section 4

ANALYSIS OF MUSIG

In this chapter we thoroughly analyze the unforgeability of the multi-signature scheme MuSig which consists of three interactive signing rounds and supports key aggregation, presented in [24]. First, we show that MuSig satisfies our definition of One-More Unforgeability. Then, we show that the seemingly benign shortcut of pre-processing the first two signing rounds before the message to sign has been selected renders the scheme completely insecure and not existentially unforgeable.

We emphasize that we only consider the 3-round version of MuSig, whereas the prior two-round version [23] is insecure [16, 11].

4.1 Scheme Description and Prior Security Proofs.

We now describe the scheme informally. A formal description of the scheme using our syntax for multi-signatures can be found in Figure 4.1.

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 $sk \leftarrow\!\!\$\, \mathbb{Z}_p$ and a public key $vk \leftarrow g^{sk}$. The aggregate public key for a group of n signers with public keys $vk_1, \ldots, vk_n$ is computed by

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

In the first signing rounds, each signer k chooses $r_k \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $R_k \leftarrow g^{r_k}$, and sends a commitment $t_k \leftarrow H_{\mathsf{com}}(R_k)$ to all the other signers. In the second round, each signer k receives the commitments $t_1, \ldots, t_n$ from all other signers, and sends R_k to all other signers. In the third round, the signer receives nonces $R_1, \ldots, R_n$ from all the signers and verifies the commitments by checking that $t_i = H_{\mathsf{com}}(R_i)$ 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 $c \leftarrow H_{\mathsf{sign}}(\tilde{vk}, R, m)$. Then, they output a signature share $z_k \leftarrow r_k + sk_k \cdot c \cdot H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$. Now, any of the signer can output the multi-signature

Scheme $\mathrm{MuSig}_{\mathbb{G},g,H_{\mathsf{agg}},H_{\mathsf{com}},H_{\mathsf{sign}}}$:

$\mathrm{MuSig.nr} = 4$

$\mathrm{MuSig.lir} = 3$

$\mathsf{KeyGen}()$:
1 $x \leftarrow\!\!\$\ \mathbb{Z}_p$; $X \leftarrow g^x$
2 $\mathsf{st}.sk \leftarrow x$; $\mathsf{st}.vk \leftarrow X$
3 Return $(sk = x, vk = X)$

$\mathsf{KeyAgg}(vk_1, \ldots, vk_n)$:
4 Return $\prod_{i=1}^{n} vk_i^{H_{\mathsf{agg}}(i, vk_1, \ldots, vk_n)}$

$\mathsf{AggVer}(m, \sigma, \tilde{vk})$:
5 $(R, z) \leftarrow \sigma$
6 Return $[g^z = R \cdot \tilde{vk}^{H_{\mathsf{sign}}(\tilde{vk}, R, m)}]$

$\mathsf{Sign}_1()$:
7 $\mathsf{st}.j \leftarrow \mathsf{st}.j + 1$; $j \leftarrow \mathsf{st}_j$
8 $\mathsf{st}_j.r \leftarrow\!\!\$\ \mathbb{Z}_p$; $\mathsf{st}_j.R \leftarrow g^{\mathsf{st}_j.r}$
9 $\mathsf{st}_j.t \leftarrow H_{\mathsf{com}}(\mathsf{st}_j.R)$
10 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 1$
11 Return $\mathsf{st}_j.t$

$\mathsf{Sign}_2(j, k, (t_i, vk_i)_{i=1}^n, m)$:
12 If $\mathsf{st}_j.\mathrm{rnd} \neq 1$, $vk_k \neq \mathsf{st}.vk$, or $\mathsf{st}_j.t \neq t_k$:
13 Return $\perp$
14 $\mathsf{st}_j.m \leftarrow m$; $\mathsf{st}_j.n \leftarrow n$; $\mathsf{st}_j.k \leftarrow k$
15 For i from 1 to n:
16 $\mathsf{st}_j.t_i \leftarrow t_i$; $\mathsf{st}_j.vk_i \leftarrow vk_i$
17 $\mathsf{st}.\mathrm{rnd} \leftarrow 2$
18 Return $\mathsf{st}_j.R$

$\mathsf{Sign}_3(j, R_1, \ldots, R_n)$:
19 If $\mathsf{st}_j.\mathrm{rnd} \neq 2$ or $R_{\mathsf{st}_j.k} \neq \mathsf{st}_j.R$:
20 Return $\perp$
21 If $\exists i$ such that $\mathsf{st}_j.t_i \neq H_{\mathsf{com}}(R_i)$:
22 Return $\perp$
23 $\tilde{vk} \leftarrow \mathsf{KeyAgg}(\mathsf{st}_j.vk_1, \ldots, \mathsf{st}_j.vk_n)$
24 $R \leftarrow \prod_{i=1}^{n} R_i$
25 $c \leftarrow H_{\mathsf{sign}}(R, \tilde{vk}, \mathsf{st}_j.m)$
26 $a \leftarrow H_{\mathsf{agg}}(\mathsf{st}_j.k, \mathsf{st}_j.vk_1, \ldots, \mathsf{st}_j.vk_n)$
27 $z \leftarrow \mathsf{st}_j.r + x \cdot c \cdot a$
28 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 3$
29 Return (R, z)

$\mathsf{Sign}_4(R, z_1, \ldots, z_n)$:
30 Return $(R, \sum_{i=1}^{n} z_i)$

Figure 4.1: A description of the secure version of 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.

(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

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

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 by MuSig2 [30], for example). The original MuSig paper [24] 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, as we show in section 4.3. 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). However, as our proof shows, the scheme is still secure if the first signing round is pre-processed before the message to sign has been determined.

Prior security proofs for MuSig. The existential unforgeability of MuSig is proved in [24, 12, 5]. 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 [5] by a factor of approximately q_s, where q_s denotes the maximum number of signing sessions opened by the adversary.

4.2 One-More Unforgeability

In this section we show that the three-round MuSig satisfies our definition of one-more unforgeability.

XIDL and Linking into the Reductions Chains. In [5], 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 2.1, and they show that it is hard whenever the discrete log assumption (Figure 2.1) holds, as written in Lemma 2.0.2.[1]

In Lemma 4.2.1, 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 4.2.1 (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 $\mathsf{MS} = \mathrm{MuSig}[\mathbb{G}, g]$ be the associated multi-signature scheme, with its hash functions modeled as random oracles. Let $\mathcal{A}_{\mathrm{ms}}$ be an adversary for the game $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ and assume the execution of $\mathcal{A}_{\mathrm{ms}}$ has at most q_0, q_1, q_2, q_s distinct queries to H_{com}, H_{agg}, H_{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 $\mathcal{A}_{\mathsf{xidl}}$ for the game $\mathrm{Gm}^{\mathsf{xidl}}_{\mathbb{G},g,q_1+q_s+1,q_2+\ell}$ such that*

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

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

We now describe the proof idea for Lemma 4.2.1, and then translate it into a formal proof.

Proof Idea for Lemma 4.2.1.

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 $\mathcal{A}_{\mathrm{ms}}$ 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

[1] They also achieve tighter security bounds using the algebraic group model [17], but this is orthogonal to this paper.

t_i as input to the second signing round and then an R_i for the third signing round satisfying $H_{\mathsf{com}}(R_i) = t_i$. 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 $\textsc{SignO}_2$ with input $(m, X_1, \ldots, X_n, t_1, \ldots, t_n)$, the reduction can recover all of the R_i and compute the session parameters.

Now suppose $\mathcal{A}_{\text{ms}}$ 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 $X_1, \ldots, X_n$ 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 $\mathcal{A}_{\text{ms}}$ wins the strong unforgeability game, at most $\ell - 1$ signing oracle sessions with message m and keys $X_1, \ldots, X_n$ 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 $\textsc{Init}$ 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 $vk_1, \ldots, vk_n$ with $vk_k = X$. If the key aggregation exponent $H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$ is an XIDL target[2] and $c = H(R, \tilde{vk}, m)$ is a challenge obtained from the XIDL's $\textsc{Challenge}$ oracle with input R, then (R, z) wins the XIDL game. Thus, we program the random oracle so that responses to $H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$ queries are indeed XIDL targets and c is an XIDL challenge corresponding

[2]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 $\sum_{\{k:X_k=X\}} H(k, vk_i, \ldots, vk_n)$ is an XIDL target and the hash function values appear uniformly random. See the formal proof for more details.

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 $R_k \leftarrow g^{z_k} X^{-c \cdot H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)}$, and sets $H_{\mathsf{com}}(R_k) \leftarrow t_k$ 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 $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$, 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 $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$ 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 $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$ when simulating $\textsc{SignO}_2$ and thus we cannot use it directly to win the XIDL game.[3]

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

[3]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.

can be opened by 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 $t_k \leftarrow H_{\mathsf{com}}(R_k) \leftarrow\!\!\$\ \mathbb{Z}_p$. Once the session parameters $(R, \tilde{vk}, m)$ are known at the initiation of the second round we can program the random oracle $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$ 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 $\frac{1}{q_s+1}$.

Dealing with multiple copies of X. The above technique works when all of the t_i's except one were generated by the adversary. However, recall that our security definition allows for multiple copies of the the same public key in the public key vector. In particular, it is possible that X appears multiple times in the vector, and all of the commitments and corresponding R's in the execution of the corresponding signing session were generated by the signing oracle. This can cause our approach to simulating the signing oracle fail, requiring another trick.

To illustrate the issue, consider the adversary initiates a signing session with two signers and $vk_1 = vk_2 = X$. Then, they complete the first signing round with the honest signer, obtaining a commitment t_1 and t_2. Now, the adversary attempts to continue each of the two signing sessions by querying the signing oracle twice for the second signing round with inputs $(1, 1, t_1, t_2, X, X, m)$ and $(2, 2, t_1, t_2, X, X, m)$ for some message m. This causes our reduction to fail, since it would have to

know the partial nonces for both signers when responding to the first query, but the partial nonce for the second signer has not yet been generated (as it will only be generated when the signing oracle completes the second signing round for that signer). Thus, the reduction fails, even though this adversarial behavior is allowed by our security definition.

To handle this case, we utilize the following trick: choose all of the corresponding R's for some session parameters at the same time. That is, even though not all of the R's for signers with verification key X have been selected yet, the reduction has all the information it needs in order to select all such R's the first time a query has been made to SIGNO_2 with these session parameters. Similarly to how we handle that case in random oracle queries to H_{agg}, we ensure all of the honest signer's Rs get generated at the same time, resolving this issue. To do this pedantically, we need to also consider all such queries as a part of the same session, for the sake of the guessing which session parameters are for forgery (otherwise, it might be the case that one session number is equal to ρ and some are not, even though they have the same session parameters, causing the reduction to fail). Thus, we need to assign session numbers at the second signing round as opposed to the first, and only then check if the session number equals to ρ.

Due to the notational complexity that dealing with this case involves we choose to skip it when formally describing the reduction in this thesis. However, we note that with some careful indexing it would be easy to translate the reduction to handle this edge case. It would not effect the concrete security parameters.

We now translate the described proof idea into a formal proof.

Proof of Lemma 4.2.1: Let $\mathbb{G}$ be a group of prime order p with a generator g and let $\mathsf{MS} = \text{MuSig}[\mathbb{G}, g]$ be the associated multi-signature scheme with its hash functions modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ be an adversary for the game $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ and assume the execution of $\mathcal{A}_{\text{ms}}$ makes at most q_0, q_1, q_2, q_s queries to $H_{\mathsf{com}}, H_{\mathsf{agg}}, H_{\mathsf{sign}}$, and SIGNO_1, respectively. Figure 4.2 constructs an adversary $\mathcal{A}_{\mathsf{xidl}}$ against the XIDL game that simulates the $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ for $\mathcal{A}_{\text{ms}}$, and if $\mathcal{A}_{\text{ms}}$ was successful it uses its output to win the XIDL game with high probability.

ANALYSIS OF $\mathcal{A}_{\mathsf{xidl}}$: 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 $\mathcal{A}_{\text{ms}}$. 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 $\mathcal{A}_{\text{ms}}$ 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 $H_{\mathsf{sign}}(R_j, \tilde{X}, m)$ 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 $\mathcal{A}_{\mathsf{xidl}}$ 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 $\mathcal{A}_{\text{ms}}$ 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 $H_{\mathsf{sign}}[R, \tilde{vk}, m]$ 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 $H_{\mathsf{com}}[R_k]$ or $H_{\mathsf{com}}[R_k]$ 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 $\mathcal{A}_{\mathsf{xidl}}$, 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 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

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

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

4.3 Insecurity with Delayed Message Selection

Here we present our attacks against the existential unforgeability of MuSig when used with delayed message selection. This version of the scheme is long known to be insecure, due to the sub-exponential attack published in a 2018 blog post by Jonas Nick's [28] (the attack is based on Wagner's algorithm for the generalized birthday algorithm [39]). We build on the ideas from the sub-exponential attacks and use the algorithm solving the ROS problem of Benhamouda et al. [11] to design a more efficient polynomial time attack in the same setting.

Suppose the first two rounds of MuSig and BN multi-signatures (the commitments and revealing the nonces rounds) are executed before message selection. A pseudo-code description of this insecure version of MuSig and comparison with the secure version is provided in Figure 4.3. We will describe the attacks when executed with two signers, but it is straightforward to generalize it to a setting with more signers.

A simple implementation of the attack can be found at `https://github.com/selanavot/MuSig_DMS_attack`, implemented using SageMath [38]. The attack runs in less than a second on a laptop machine.

ATTACK SETTING. The standard existential unforgeability definition allows the adversary to corrupt all but one of the signers in a group, as well as ask the honest signer for signatures with differing groups. Furthermore, to break existential unforgeability the adversary only needs to forge a signature for an arbitrary message. All of our attacks can be carried out by a weaker adversary, who can forge a signature for a message of their choice.

In particular, in the attacks against MuSig the adversary only needs to observe parallel signing sessions and control which message will be signed, but can succeed against any group of signers even if none of them is corrupt. Our attacks against BN multi-signatures also don't require the adversary to collude with signers, but the adversary needs the honest signers to complete a signing session for different messages. This is possible when the adversary mediates signer communication or corrupts all but one of the signers.

THE POLYNOMIAL TIME ATTACK. Let S_1 and S_2 be the signers with private keys x_1 and x_2 and public key X_1 and X_2, respectively. Let $\widetilde{X} = X_1^{H_{\mathsf{agg}}(1,X_1,X_2)} \cdot X_2^{H_{\mathsf{agg}}(2,X_1,X_2)}$ denote the aggregate verification key. This time, let $\ell \geq \lceil \log_2(p) \rceil$ be an integer and let $m_{\ell+1}$ be some message for which the adversary wishes to forge a signature, and for each $i \in \{1, \ldots, \ell\}$ choose distinct messages m_i^0 and m_i^1 that the signers would be willing to sign.

Now, the adversary begins ℓ signing sessions and observes the first two signing rounds to obtain an aggregate nonce $R_i = R_{i,1} \cdot R_{i,2}$ for each $i \in \{1, \ldots, \ell\}$. Then the adversary calculates the corresponding challenges $c_i^0 = H_{\mathsf{sign}}(R_i, \widetilde{X}, m_i^0)$ and $c_i^1 = H_{\mathsf{sign}}(R_i, \widetilde{X}, m_i^1)$ for each $i \in \{1, \ldots, \ell\}$. Now, define the group homomorphisms $\rho^+ : (\mathbb{Z}_p)^\ell \to \mathbb{Z}_p$ and $\rho^\times : (\mathbb{G})^\ell \to \mathbb{G}$ as follows:

$$\rho^+(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} \frac{2^{i-1} x_i}{c_i^1 - c_i^0},$$

and

$$\rho^\times(g_1, \ldots, g_\ell) = \prod_{i=1}^{\ell} g_i^{\frac{2^{i-1}}{c_i^1 - c_i^0}}.$$

Let $R_{\ell+1} = \rho^\times(R_1, \ldots, R_\ell)$, and let $c_{\ell+1} = H_{\mathsf{sign}}(R_{\ell+1}, \widetilde{X}, m_{\ell+1})$. Let $d = c_{\ell+1} - \rho^+(c_1^0, \ldots, c_\ell^0)$, and

write d in binary as $\sum_{i=1}^{\ell} 2^{i-1} b_i$ for some $b_1, \ldots, b_\ell \in \{0,1\}$, which is possible since $\ell \geq \lceil \log_2(p) \rceil$. Now, for each $i \in \{1, \ldots, \ell\}$, complete the signing session i with the message $m_i^{b_i}$ to obtain a multi-signature (R_i, z_i). We claim that $\sigma = (R_{\ell+1}, \rho^+(z_1, \ldots, z_\ell))$ is a valid multi-signature for the message $m_{\ell+1}$ under the aggregate verification key $\widetilde{X}$, and is thus a forgery that breaks the unforgeability of the scheme.

VALIDITY OF FORGED SIGNATURE. We wish to verify that $\sigma = (R_{\ell+1}, \rho^+(z_1, \ldots, z_\ell))$ is a valid signature for $m_{\ell+1}$ under the aggregate verification key $\widetilde{X}$. Thus, we must show that

$$g^{\rho^+(z_1,\ldots,z_\ell)} = R_{\ell+1} \widetilde{X}^{c_{\ell+1}}.$$

Note that (R_i, z_i) is a valid Schnorr signature for the message $m_i^{b_i}$ under the verification key $\widetilde{X}$ for each i, and thus $g^{z_i} = R_i \widetilde{X}^{c_i^{b_i}}$. Hence,

$$\rho^\times(g^{z_1}, \ldots, g^{z_\ell}) = \rho^\times(R_1 \widetilde{X}^{c_1^{b_1}}, \ldots, R_\ell \widetilde{X}^{c_\ell^{b_\ell}}),$$

or equivalently,

$$g^{\rho^+(z_1,\ldots,z_\ell)} = \rho^\times(R_1, \ldots, R_\ell) \cdot \widetilde{X}^{\rho^+(c_1^{b_1},\ldots,c_\ell^{b_\ell})}.$$

But $R_{\ell+1} = \rho^\times(R_1, \ldots, R_\ell)$, and Lemma 4.3.1 shows that $\rho^+(c_1^{b_1}, \ldots, c_\ell^{b_\ell}) = c_{\ell+1}$. Hence,

$$g^{\rho^+(z_1,\ldots,z_\ell)} = R_{\ell+1} \cdot \widetilde{X}^{c_{\ell+1}},$$

which is what we wanted to prove.

Lemma 4.3.1. *By the construction above,* $\rho^+(c_1^{b_1}, \ldots, c_\ell^{b_\ell}) = c_{\ell+1}$.

This lemma is at the heart of the attack, and is precisely the idea that allows Benhamouda et al. to solve the ROS problem [11].

Proof of Lemma 4.3.1. By definition, $\sum_{i=1}^{\ell} 2^{i-1} b_i = c_{\ell-1} - \rho^+(c_1^0, \ldots, c_\ell^0)$. Hence, to prove the lemma it is sufficient to show that $\sum_{i=1}^{\ell} 2^{i-1} b_i = \rho^+(c_1^{b_1}, \ldots, c_\ell^{b_\ell}) - \rho^+(c_1^0, \ldots, c_\ell^0)$.

Starting from the right-hand side, we have that

$$\begin{aligned}\rho^+(c_1^{b_1},\ldots,c_\ell^{b_\ell}) - \rho^+(c_1^0,\ldots,c_\ell^0) &= \rho^+(c_1^{b_1}-c_1^0,\ldots,c_\ell^{b_\ell}-c_\ell^0)\\ &= \sum_{i=1}^{\ell} \frac{2^{i-1}(c_i^{b_i}-c_i^0)}{c_i^1-c_i^0}.\end{aligned}$$

However, for each i it holds that $\frac{2^{i-1}(c_i^{b_i}-c_i^0)}{c_i^1-c_i^0}$ is 0 whenever b_i is 0 and is 2^{i-1} whenever b_i is 1. Hence, for each i it holds that $\frac{2^{i-1}(c_i^{b_i}-c_i^0)}{c_i^1-c_i^0} = 2^{i-1}b_i$, and thus the right-hand side of the equation above simplifies to $\sum_{i=1}^{\ell} 2^{i-1}b_i$, which completes the proof.

$\mathcal{A}_{\mathsf{xidl}}^{\text{NewTarget,Challenge}}(X)$:

$\mathsf{bad} \leftarrow \mathsf{false}$ // abort whenever $\mathsf{bad} \leftarrow \mathsf{true}$
$H_{\mathsf{agg}}, H_{\mathsf{sign}}, H_{\mathsf{com}} \leftarrow$ empty dictionary
$K \leftarrow \emptyset$ // tracks aggregated keys used
$\mathrm{ctrs}, \mathrm{ctrt}, \mathrm{ctrc} \leftarrow 0$ // tracks signing oracle queries, XIDL targets, XIDL challenges
$T_{\mathsf{com}} T_{\mathsf{tar}}, T_{\mathsf{chal}} \leftarrow$ empty dictionary // tracks H_{com} queries, XIDL targets, XIDL challenges
$\mathsf{st} \leftarrow$ empty list
$\rho \leftarrow\!\!\$\ \{1, \ldots, q_s + 1\}$ // guess SignO_2 query for Case 2 forgery
$\mathrm{Sim} \leftarrow \{(\widetilde{\text{SignO}}_i)_{i=1}^3, \widetilde{H_{\mathsf{agg}}}, \widetilde{H_{\mathsf{sign}}}, \widetilde{H_{\mathsf{com}}}\}$
$(m, (X_i)_{i=1}^n, (R_j, z_j)_{j=1}^\ell) \leftarrow \mathcal{A}_{\mathrm{ms}}^{\mathrm{Sim}}(X)$
$\tilde{X} \leftarrow \prod_{i=1}^n X_i^{\widetilde{H_{\mathsf{agg}}}(i, X_1, \ldots, X_n)}$
For $j = 1, \ldots, \ell$:
  $\widetilde{H_{\mathsf{sign}}}(R_j, \tilde{X}, m)$ // ensure initialization
  If $T_{\mathsf{chal}}[R_j, \tilde{X}, m]$ initialized:
    Return $(T_{\mathsf{chal}}[R_j, \tilde{X}, m], z_j)$
Return $\perp$

$\widetilde{\text{SignO}}_1()$:

$t \leftarrow\!\!\$\ \mathbb{Z}_p$
$\mathrm{ctrs} \leftarrow \mathrm{ctrs} + 1$, $\mathsf{st}_{\mathrm{ctrs}}.t \leftarrow t$
Return t

$\widetilde{\text{SignO}}_2(j, k, (vk_i, t_i)_{i=1}^n, m)$:

$\tilde{vk} \leftarrow \prod_{i=1}^n vk_i^{\widetilde{H_{\mathsf{agg}}}(i, vk_1, \ldots, vk_n)}$
If $j = \rho$:
  $R_k \leftarrow\!\!\$\ \mathbb{G}$
Else:
  $z, c \leftarrow\!\!\$\ \mathbb{Z}_p$; $\mathsf{st}_j.z \leftarrow z$
  $R_k \leftarrow g^z X^{-c}$
  For $i \in \{1, \ldots, n\} \setminus \{k\}$:
    If $T_{\mathsf{com}}[t_i]$ initialized: $R_i \leftarrow T_{\mathsf{com}}(t_i)$
    Else:
      $R_i \leftarrow\!\!\$\ \mathbb{Z}_p$
      If $H_{\mathsf{com}}[R_i]$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
      $H_{\mathsf{com}}[R_i] \leftarrow t_i$; $T_{\mathsf{com}}[t_i] \leftarrow R_i$
  $R \leftarrow \prod_{i=1}^n R_i$; $\mathsf{st}_j.R \leftarrow R$
  $\mathsf{st}_j.\tilde{vk} \leftarrow \tilde{vk}$
  If $H_{\mathsf{sign}}[R, \tilde{vk}, m]$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
  $H_{\mathsf{sign}}[R, \tilde{vk}, m] \leftarrow c$
If $H_{\mathsf{com}}[R_k]$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
$H_{\mathsf{com}}[R_k] \leftarrow \mathsf{st}_j.t$
Return R_k

$\widetilde{\text{SignO}}_3(j, R_1, \ldots, R_n)$:

If $j = \rho$: $\mathsf{bad} \leftarrow \mathsf{true}$
Return $\mathsf{st}_j.z$

$\widetilde{H_{\mathsf{agg}}}(k, X_1, \ldots, X_n)$:

If $H_{\mathsf{agg}}[k, X_1, \ldots, X_n]$ uninitialized:
  For $i = 1, \ldots, n$:
    If $X_i \neq X$: $H_{\mathsf{agg}}[i, X_1, \ldots, X_n] \leftarrow\!\!\$\ \mathbb{Z}_p$
  $S \leftarrow \prod_{\{i:\ X_i \neq X\}} X_i^{H_{\mathsf{agg}}[i, X_1, \ldots, X_n]}$
  $e \leftarrow \text{NewTarget}(S)$
  $J \leftarrow \{j : X_j = X\}$; $j_{\max} \leftarrow \max(J)$
  For $i \in J \setminus \{j_{\max}\}$:
    $e_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $H_{\mathsf{agg}}[i, X_1, \ldots, X_n] \leftarrow e_i$
  $H_{\mathsf{agg}}[j_{\max}, X_1, \ldots, X_n] \leftarrow e - \sum_{i \in J \setminus \{j_{\max}\}} e_i$
  $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\mathsf{agg}}[i, X_1, \ldots, X_n]}$
  If $\exists (R, m)$ such that $H_{\mathsf{sign}}[R, \tilde{X}, m]$ is initialized:
    $\mathsf{bad} \leftarrow \mathsf{true}$
  If $\tilde{X} \in K$:
    $\mathsf{bad} \leftarrow \mathsf{true}$
  $K \leftarrow K \cup \{\tilde{X}\}$
  $\mathrm{ctrt} \leftarrow \mathrm{ctrt} + 1$; $T_{\mathsf{tar}}[\tilde{X}] \leftarrow \mathrm{ctrt}$
Return $H_{\mathsf{agg}}[k, X_1, \ldots, X_n]$

$\widetilde{H_{\mathsf{com}}}(R)$:

If $H_{\mathsf{com}}[R]$ uninitialized:
  $H_{\mathsf{com}}[R] \leftarrow\!\!\$\ \mathbb{Z}_p$;
  If $T_{\mathsf{com}}[H_{\mathsf{com}}[R]]$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
  $T_{\mathsf{com}}[H_{\mathsf{com}}[R]] \leftarrow R$
Return $H_{\mathsf{com}}[R]$

$\widetilde{H_{\mathsf{sign}}}(R, \tilde{X}, m)$:

If $H_{\mathsf{sign}}[R, \tilde{X}, m]$ uninitialized:
  If $T_{\mathsf{tar}}[\tilde{X}]$ uninitialized:
    $H_{\mathsf{sign}}[R, \tilde{X}, m] \leftarrow\!\!\$\ \mathbb{Z}_p$
  Else:
    $c \leftarrow \text{Challenge}(T_{\mathsf{tar}}[\tilde{X}], R)$
    $\mathrm{ctrc} \leftarrow \mathrm{ctrc} + 1$
    $T_{\mathsf{chal}}[R, \tilde{X}, m] \leftarrow \mathrm{ctrc}$
Return $H_{\mathsf{sign}}[R, \tilde{X}, m]$

Figure 4.2: The reduction algorithm used in the proof of Lemma 4.2.1.

Scheme MuSig$[\mathbb{G}, g, p, H_{\mathsf{com}}, H_{\mathsf{agg}}, H_{\mathsf{sign}}]$:

Scheme InsecureMuSig$[\mathbb{G}, g, p, H_{\mathsf{com}}, H_{\mathsf{agg}}, H_{\mathsf{sign}}]$:

$\mathrm{MuSig.nr} = 4$

$\mathrm{MuSig.lir} = 3$

$\mathsf{KeyGen}()$:
1 $x \leftarrow\!\!\$\ \mathbb{Z}_p$; $X \leftarrow g^x$
2 $\mathsf{st}.sk \leftarrow x$; $\mathsf{st}.vk \leftarrow X$
3 Return $(sk = x, vk = X)$

$\mathsf{KeyAgg}(vk_1, \ldots, vk_n)$:
4 Return $\prod_{i=1}^{n} vk_i^{H_{\mathsf{agg}}(i, vk_1, \ldots, vk_n)}$

$\mathsf{AggVer}(m, \sigma, \tilde{vk})$:
5 $(R, z) \leftarrow \sigma$
6 Return $[g^z = R \cdot \tilde{vk}^{H_{\mathsf{sign}}(\tilde{vk}, R, m)}]$

$\mathsf{Sign}_1()$:
7 $\mathsf{st}.j \leftarrow \mathsf{st}.j + 1$; $j \leftarrow \mathsf{st}_j$
8 $\mathsf{st}_j.r \leftarrow\!\!\$\ \mathbb{Z}_p$; $\mathsf{st}_j.R \leftarrow g^{\mathsf{st}_j.r}$
9 $\mathsf{st}_j.t \leftarrow H_{\mathsf{com}}(\mathsf{st}_j.R)$
10 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 1$
11 Return $\mathsf{st}_j.t$

$\mathsf{Sign}_2(j, k, t_1, \ldots, t_n, k$ [, $vk_1, \ldots, vk_n, m$] $)$:
12 If $\mathsf{st}_j.\mathrm{rnd} \neq 1$ or $\mathsf{st}_j.t \neq t_k$:
13 Return $\perp$
14 $\mathsf{st}_j.k \leftarrow k$; $\mathsf{st}_s.n \leftarrow n$
15 For i from 1 to n: $\mathsf{st}_s.t_i \leftarrow t_i$
16 [Save_Session_Params$(j, k, vk_1, \ldots, vk_n, m)$]
17 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 2$
18 Return $\mathsf{st}_j.R$

$\mathsf{Sign}_3(j, R_1, \ldots, R_n$ [, $m, vk_1, \ldots, vk_n$] $)$:
19 If $\mathsf{st}_j.\mathrm{rnd} \neq 2$ or $R_{\mathsf{st}_j.k} \neq \mathsf{st}_j.R$:
20 Return $\perp$
21 If $\exists i$ such that $\mathsf{st}_j.t_i \neq H_{\mathsf{com}}(R_i)$:
22 Return $\perp$
23 [Save_Session_Params$(j, \mathsf{st}_j.\mathrm{me}, vk_1, \ldots, vk_n, m)$]
24 $R \leftarrow \prod_{i=1}^{n} R_i$
25 $\tilde{vk} \leftarrow \mathsf{KeyAgg}(\mathsf{st}_j.vk_1, \ldots, \mathsf{st}_j.vk_n)$
26 $c \leftarrow H_{\mathsf{sign}}(R, \tilde{vk}, \mathsf{st}_j.m)$
27 $a \leftarrow H_{\mathsf{agg}}(\mathsf{st}_j.k, \mathsf{st}_j.vk_1, \ldots, \mathsf{st}_j.vk_n)$
28 $z \leftarrow \mathsf{st}_j.r + \mathsf{st}.sk \cdot c \cdot a$
29 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 3$
30 Return (R, z)

$\mathsf{Sign}_4(R, z_1, \ldots, z_n)$:
31 Return $(R, \sum_{i=1}^{n} z_i)$

$\mathsf{Save\_Session\_Params}(j, k, vk_1, \ldots, vk_n, m)$:
32 // helper method to store session params
33 If $vk_k \neq \mathsf{st}.vk$:
34 Return $\perp$
35 $\mathsf{st}_j.m \leftarrow m$
36 For i from 1 to n: $\mathsf{st}_j.vk_i \leftarrow vk_i$

Figure 4.3: Comparison of the secure version of MuSig and the insecure version with delayed message selection. The secure version contains all but the dashed boxes, and the insecure version contains all but the solid boxes.

Section 5

ANALYSIS OF BELLARE-NEVEN MULTI-SIGNATURES

In this chapter we analyze the Bellare-Neven multi-signature scheme (abbreviated BN) [7]. It is a predecessor of MuSig and the first secure scheme in the plain public key model, meaning the setting where the signers do not need to participate in a distributed key generation protocol or prove ownership of a secret key. It is very similar to MuSig but without key aggregation, and is also proven secure in the random oracle model under the discrete log assumption when the message to sign is determined before the second signing round and the hash functions are modeled as random oracles.

First, we show that BN satisfies our definition of One-More Unforgeability. In addition to proving one-more unforgeability, this proof also shows that it is impossible to forge a signature for a message for which a signing session was initiated but not completed. Additionally, it proves the security of BN signatures when the first signing round is pre-processed before the message has been determined.

Then, we show that as is the case with MuSig, pre-processing the second signing rounds before the message to sign has been selected renders the scheme completely insecure and not existentially unforgeable. This analysis is presented after the analysis of MuSig is due to this attack building on the attack against MuSig with delayed message selection.

5.1 Scheme Description

We follow the scheme description of Bellare and Dai [5], which has minor differences from the original paper [7]. We now describe the scheme informally, and include a formal description in Figure 5.1 using our syntax.

The scheme is parameterized by a group $\mathbb{G}$ of prime order p with a generator g and two hash functions H_{com} and H_{sign} with codomain $\mathbb{Z}_p$ that are used for commitments and signing respectively.

Key generation and the first two signing rounds were left unaltered by MuSig. For key gen-

eration, each signer k generates a private key $sk_k \leftarrow\!\!\$\, \mathbb{Z}_p$ and a public key $vk_k \leftarrow g^{sk_k}$. For the first signing round, signer k chooses $r_k \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $R_k \leftarrow g^{r_k}$, and outputs a commitment $t_k \leftarrow H_{\mathsf{com}}(R_k)$ which is sent to all the other signers. In the second signing round, the signer receives the commitments from all other signers $t_1, \ldots, t_n$, and outputs R_k. In the third signing round, which is different from MuSig, the signer k receives nonces $R_1, \ldots, R_n$ from all the signers and verifies the commitments by checking that $t_i = H_{\mathsf{com}}(R_i)$ for each i. Then, they compute $R \leftarrow \prod_{i=1}^{n} R_i$, a challenge $c_k \leftarrow H_{\mathsf{sign}}(k, R, vk_1, \ldots, vk_n, m)$, and output a signature share $z_k \leftarrow r_k + sk_k \cdot c_k$. Now, any of the signer can output the multi-signature (R, z) where $R \leftarrow \prod_{i=1}^{n} R_i$ and $z \leftarrow \sum_{i=1}^{n} z_i$.

Verification requires the message m, the signature $\sigma = (R, z)$, and all the signers public keys $(vk_1, \ldots, vk_n)$. The verifier computes $c_i \leftarrow H_{\mathsf{sign}}(i, R, vk_1, \ldots, vk_n, m)$ for each $i \in \{1, \ldots, n\}$ and output true if and only if $g^z = R \prod_{i=1}^{n} vk_i^{c_i}$.

As with MuSig, the message and the keys of all the signers are not used until the third signing round. However, as our attack shows, it is needed for security that the message being signed is selected before the second signing round.

5.2 One-More Unforgeability

We now prove the one-more unforgeability of the BN-scheme using the IDL assumption, as presented in Lemma 5.2.1. Combined with Lemma 2.0.1, this proves the one-more unforgeability of BN from the discrete log assumption. The proof is very similar to that of the one-more unforgeability of MuSig, except that the lack of key aggregation allows the use of the IDL assumption as opposed to the XIDL.

Lemma 5.2.1 (IDL $\rightarrow$ SUF of BN in the ROM)**.** *Let $\mathbb{G}$ be a group of prime order p. Let g be a generator of $\mathbb{G}$. Let $\mathsf{MS} = BN[\mathbb{G}, g]$ be the associated multi-signature scheme, with its hash functions modeled as random oracles. Let $\mathcal{A}_{\mathrm{ms}}$ be an adversary for the game $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ and assume the execution of $\mathcal{A}_{\mathrm{ms}}$ has at most q_0, q_1, q_s distinct queries to H_{com}, H_{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 $q = q_0(q_0 + n \cdot q_s) + q_s(q_1 + q_s) + n \cdot q_s(q_0 + n \cdot q_s)$. Then,*

there exists an adversary $\mathcal{A}_{\mathsf{idl}}$ *for the game* $\mathrm{Gm}^{\mathsf{idl}}_{\mathbb{G},g,q_1}$ *such that*

$$\mathbf{Adv}^{\mathrm{suf-ms}}_{\mathsf{MS}}(\mathcal{A}_{\mathrm{ms}}) \leq (1+q_s)\mathbf{Adv}^{\mathsf{idl}}_{\mathbb{G},g,q_1}(\mathcal{A}_{\mathsf{idl}}) + \frac{q}{p}$$

and the running time of $\mathcal{A}_{\mathsf{idl}}$ *is similar to that of* $\mathcal{A}_{\mathrm{ms}}$.

We now describe the proof idea for Lemma 5.2.1, and then translate it into a formal proof.

Proof Idea for Lemma 5.2.1.

We describe informally how to win the IDL game with high probability given an adversary that breaks the strong unforgeability of BN in the ROM.

Session parameters and signature types. Each execution of the third signing round of the signing oracle uses a specific vector of public keys $vk_1, \ldots, vk_n$, aggregate nonce R, and message m. We refer this tuple $(R, vk_1, \ldots, vk_n, m)$ as the session parameter of this signing session.

Now consider an adversary $\mathcal{A}_{\mathrm{ms}}$ 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 $H_{\mathsf{com}}(R_i) = t_i$. 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 $\textsc{Sign}\mathrm{O}_2$ with input $(m, vk_1, \ldots, vk_n, t_1, \ldots, t_n)$, the reduction can recover all of the R_i and compute the session parameters.

Suppose $\mathcal{A}_{\mathrm{ms}}$ breaks the strong unforgeability of BN. 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 $vk_1, \ldots, vk_n$. Each of these signatures must fall into one of the following cases:

Case 1: $(R_j, vk_1, \ldots, vk_n, m)$ was the session parameters for some signing oracle signing session that executed the third signing round.

Case 2: $(R_j, vk_1, \ldots, vk_n, m)$ was the session parameters for some signing oracle signing session that executed the second signing round, but not the third.

Case 3: $(R_j, vk_1, \ldots, vk_n, m)$ was not the session parameters for any signing oracle signing session.

Since $\mathcal{A}_{\text{ms}}$ wins the strong unforgeability game, at most $\ell-1$ signing oracle sessions with message m and keys $vk_1, \ldots, vk_n$ 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 signatures that fall into Case 2 or 3 as "forgeries."

Using a forgery to win IDL. Let X denote the output of the IDL 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 $vk_1, \ldots, vk_n$ with $vk_k = X$. If $c = H(k, R, vk_1, \ldots, vk_n, m)$ is a challenge obtained from the IDL's Challenge oracle with input $R \cdot \prod_{i=1}^{n} vk_i^{c_i}$, then z wins the IDL game. Thus, we program the random oracle so that responses to $H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$ queries are indeed IDL 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 (i.e. the secrete log of X). To do that, we extend the standard technique of simulating the Schnorr signing oracle without knowledge of the secret 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 $R_k \leftarrow g^{z_k} X^{-c \cdot H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)}$, and sets $H_{\mathsf{com}}(R_k) \leftarrow t_k$ so the commitment from the first round holds. As explained before, the reduction can now recover the session parameters $(R, vk_1, \ldots, vk_n, m)$ even though R is not yet known to the adversary. Therefore, it programs the random oracle $H_{\mathsf{sign}}(R, vk_1, \ldots, vk_n, m) \leftarrow c$, 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 $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow$

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 $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$ when simulating SIGNO_2 and thus we cannot use it directly to win the XIDL game.[1]

To win the XIDL game using this type of forgery, the reduction generates an integer ρ uniformly at random from $\{1, \ldots, q_s, q_s + 1\}$ where q_s refers to the maximum number of signing sessions that can be opened by 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 $t_k \leftarrow H_{\mathsf{com}}(R_k) \leftarrow\!\!\$\ \mathbb{Z}_p$. Once the session parameters $(R, \tilde{vk}, m)$ are known at the initiation of the second round we can program the random oracle $H_{\mathsf{sign}}(R, \tilde{vk}, m) \leftarrow c$ 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 $\frac{1}{q_s+1}$.

---

[1] 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.

DEALING WITH MULTIPLE COPIES OF X. As was the case in the MuSig security proof, the reduction requires an additional trick when X appears multiple times in the list of public keys during queries to SIGNO_2. As in the MuSig security proof, this issue can be resolved by generating all of the corresponding R's at the same time. However, this solution would result in increased notational complexity and little effect on the reduction technique or the concrete result, so we choose to not use it when presenting our final proof. However, with some care our reduction can be translated to handle that case.

We now translate these proof ideas into a formal proof.

Proof of Lemma 5.2.1: Let $\mathbb{G}$ be a group of prime order p with a generator g and let $\mathsf{MS} = \text{BN}[\mathbb{G}, g]$ be the associated multi-signature scheme with its hash functions modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ be an adversary for the game $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ and assume the execution of $\mathcal{A}_{\text{ms}}$ makes at most q_0, q_1, q_s queries to $H_{\mathsf{com}}, H_{\mathsf{sign}}$, and SIGNO_1, respectively. Figure 5.2 constructs an adversary $\mathcal{A}_{\mathsf{idl}}$ against the IDL game that simulates the $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ for $\mathcal{A}_{\text{ms}}$, and if $\mathcal{A}_{\text{ms}}$ was successful it uses its output to win the IDL game with high probability.

ANALYSIS OF $\mathcal{A}_{\mathsf{idl}}$: 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 $\mathcal{A}_{\text{ms}}$. 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 $\mathcal{A}_{\text{ms}}$ wins, then it must be the case that they provided signatures $(R_j, z_j)_{j=1}^{\ell}$ for a message m and public keys $(vk_i)_{i=1}^{n}$ such that ℓ is more than the number of signing oracle sessions with the message m and public keys vector $(vk_i)_{i=1}^{n}$ that programmed the H_{sign} challenge. Hence, one of the challenges $H_{\mathsf{sign}}(k, R_j, (vk_i)_{i=1}^{n}, m)$ for $vk_k = X$ must be an IDL challenge, and thus the reduction wins the IDL 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 $\mathcal{A}_{\mathsf{idl}}$ and the reduction simulates H_{com}

q_0 times. Hence, the probability of a collision is no more than $\frac{q_0(q_0+n\cdot q_s)}{p}$.

- If in the simulation of $\textsc{SignO}_2$ the value $H_{\mathsf{sign}}[j, R, (vk_i)_{i=1}^n, m]$ is already defined (for the R chosen uniformly at random at this execution). The probability of this happening at each execution of $\widetilde{\textsc{SignO}}_2$ is at most $\frac{q_1+q_s}{p}$, 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 $\frac{q_s(q_1+q_s)}{p}$.

- In the simulation of $\textsc{SignO}_2$, the value $H_{\mathsf{com}}[R_k]$ or $H_{\mathsf{com}}[R_k]$ is already defined (for the R_k chosen uniformly at random at this execution). The probability of this happening at each execution of $\widetilde{\textsc{SignO}}$ 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 $\mathcal{A}_{\mathsf{idl}}$, 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 $\textsc{SignO}_3$. 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 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

$$\frac{q_0(q_0 + n \cdot q_s) + q_s(q_1 + q_s) + n \cdot q_s(q_0 + n \cdot q_s)}{p}.$$

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

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

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

5.3 Insecurity with Delayed Message Selection

Here we present our attacks against BN multi-signatures when used with delayed message selection, which utilizes the algorithm solving the ROS problem of Benhamouda et al. [11].

Suppose the first two rounds BN multi-signatures (the commitments and revealing the nonces rounds) are executed before message selection (see the insecure version in Figure 5.1). We will describe the attacks when executed with two signers, but it is straightforward to generalize it to a setting with more signers.

ATTACK SETTING. The standard existential unforgeability definition allows the adversary to corrupt all but one of the signers in a group, as well as ask the honest signer for signatures with differing groups. Furthermore, to break existential unforgeability the adversary only needs to forge a signature for an arbitrary message.

The attack we present here can be carried out by a weaker adversary, who can forge a signature for a message of their choice. In particular, our attack doesn't require the adversary to collude with signers, but the adversary needs the honest signers to complete a signing session for different messages. This is possible when the adversary mediates signer communication or corrupts all but one of the signers.

ATTACK DETAILS. This attack is an extension of the attack against MuSig with delayed message selection (Section 4.3). The main difference is that in contrast to MuSig, different signers in BN multi-signatures use distinct challenges when signing a message. At a high level, we now need a separate instance of the ROS attack that we used against MuSig for each signer in order to forge their partial signature.

As before, let S_1 and S_2 be the signers with private keys x_1 and x_2 and public keys X_1 and X_2, respectively. Let $\ell \geq \lceil \log_2(p) \rceil$ be an integer, let $m_{\ell+1}$ be some message for which the adversary wishes to forge a multi-signature, and for each $i \in \{1, \ldots, \ell\}$ choose distinct messages m_i^0 and m_i^1 that the signers would be willing to sign.

Now, the adversary begins ℓ signing sessions and observes the first two signing rounds to obtain nonce shares $R_{i,1} = g^{r_{i,1}}$ and $R_{i,2} = g^{r_{i,2}}$ for each $i \in \{1, \ldots, \ell\}$. Then, the adversary calculates challenges $c_{i,1}^b = H_{\mathsf{sign}}(1, R_{i,1} \cdot R_{i,2}, X_1, X_2, m_i^b)$ and $c_{i,2}^b = H_{\mathsf{sign}}(2, R_{i,1} \cdot R_{i,2}, X_1, X_2, m_i^b)$ for each $i \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$. Now, define the group homomorphisms $\rho_j^+ : (\mathbb{Z}_p)^\ell \to \mathbb{Z}_p$ and

$\rho_j^{\times} : (\mathbb{G})^{\ell} \to \mathbb{G}$ for $j \in \{1, 2\}$ as follows:

$$\rho_j^{+}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} \frac{2^{i-1} x_i}{c_{i,j}^1 - c_{i,j}^0},$$

and

$$\rho_j^{\times}(g_1, \ldots, g_\ell) = \prod_{i=1}^{\ell} g_i^{\frac{2^{i-1}}{c_{i,j}^1 - c_{i,j}^0}}.$$

Let $R_1 = \rho_1^{\times}(R_{1,1}, \ldots, R_{\ell,1})$ and $R_2 = \rho_2^{\times}(R_{1,2}, \ldots, R_{\ell,2})$. Let $R = R_1 \cdot R_2$, let $c_{\ell+1,1} = H_{\mathsf{sign}}(1, R, X_1, X_2, m_{\ell+1})$, and also let $c_{\ell+1,2} = H_{\mathsf{sign}}(2, R, X_1, X_2, m_{\ell+1})$. Let $d_1 = c_{\ell+1,1} - \rho_1^{+}(c_{1,1}^0, \ldots, c_{\ell,1}^0)$ and write it in binary as $\sum_{i=1}^{\ell} 2^{i-1} b_{i,1}$ for some $b_{1,1}, \ldots, b_{\ell,1} \in \{0, 1\}$, which is possible since $\ell \geq \lceil \log_2(p) \rceil$. Similarly, let $d_2 = c_{\ell+1,2} - \rho_2^{+}(c_{1,2}^0, \ldots, c_{\ell,2}^0)$ and write it in binary as $\sum_{i=1}^{\ell} 2^{i-1} b_{i,2}$.

Now, for each $i \in \{1, \ldots, \ell\}$ complete the third signing round of the signing session i with S_1 using the message $m_i^{b_{i,1}}$ to obtain a signature share $z_{i,1} = r_{i,1} + c_{i,1}^{b_{i,1}} \cdot x_1$. Similarly, complete the third signing round of session i with S_2 using the (potentially different) message $m_i^{b_{i,2}}$ to obtain a signature share $z_{i,2} = r_{i,2} + c_{i,2}^{b_{i,2}} \cdot x_2$. Now, they can calculate $z_{\ell+1,1} = \rho_1^{+}(z_{1,1}, \ldots, z_{\ell,1})$ and $z_{\ell+1,2} = \rho_2^{+}(z_{1,2}, \ldots, z_{\ell,2})$.

We claim that $\sigma = (R, z_{\ell+1,1} + z_{\ell+1,2})$ is a valid multi-signature for the message $m_{\ell+1}$ under the group S_1 and S_2, and thus this attack breaks the existential unforgeability of the scheme.

Validity of forged signature. We wish to verify that $\sigma = (R, z_{\ell+1,1} + z_{\ell+1,2})$ is a valid multi-signature for the message $m_{\ell+1}$ and the group of signers S_1 and S_2. Hence, we must show that

$$g^{z_{\ell+1,1} + z_{\ell+1,2}} = R \cdot X_1^{c_{\ell+1,1}} \cdot X_2^{c_{\ell+1,2}}.$$

Starting from the left-hand side, we have that

$$\begin{aligned} g^{z_{\ell+1,1} + z_{\ell+1,2}} &= g^{\rho_1^{+}(z_{1,1}, \ldots, z_{\ell,1}) + \rho_2^{+}(z_{1,2}, \ldots, z_{\ell,2})} \\ &= \rho_1^{\times}(R_{1,1}, \ldots, R_{\ell,1}) \cdot \rho_2^{\times}(R_{1,2}, \ldots, R_{\ell,2}) \cdot X_1^{\rho_1^{+}(c_{1,1}^{b_{1,1}}, \ldots, c_{\ell,1}^{b_{\ell,1}})} X_2^{\rho_2^{+}(c_{1,2}^{b_{1,2}}, \ldots, c_{\ell,2}^{b_{\ell,2}})}. \end{aligned}$$

The first two terms in the above equation are R_1 and R_2. Furthermore, $\rho_1^{+}(c_{1,1}^{b_{1,1}}, \ldots, c_{\ell,1}^{b_{\ell,1}}) = c_{\ell+1,2}$

and $\rho_2^+(c_{1,2}^{b_{1,2}}, \ldots, c_{\ell,2}^{b_{\ell,2}}) = c_{\ell+1,2}$ using the same idea as in Lemma 4.3.1. Thus, the above equation simplifies to

$$g^{z_{\ell+1,1}+z_{\ell+1,2}} = R \cdot X_1^{c_{\ell+1,1}} \cdot X_2^{c_{\ell+1,2}},$$

which is what we wanted to prove.

Scheme BN$[\mathbb{G}, g, p, H_{\mathsf{com}}, H_{\mathsf{sign}}]$:

Scheme InsecureBN$[\mathbb{G}, g, p, H_{\mathsf{com}}, H_{\mathsf{sign}}]$:

$\mathrm{BN.nr} = 4$; $\mathrm{BN.lir} = 3$

$\mathsf{KeyGen}()$:
1 $x \leftarrow\!\!\$\ \mathbb{Z}_p$; $X \leftarrow g^x$
2 $\mathsf{st}.sk \leftarrow x$; $\mathsf{st}.vk \leftarrow X$
3 Return $(sk = x, vk = X)$

$\mathsf{Verify}(m, \sigma, vk_1, \ldots, vk_n)$:
4 $(R, z) \leftarrow \sigma$
5 Return $[g^z = R \cdot \prod_{i=1}^{n} vk_i^{H_{\mathsf{sign}}(i, R, vk_1, \ldots, vk_n, m)}]$

$\mathsf{Sign}_1()$:
6 $\mathsf{st}.j \leftarrow \mathsf{st}.j + 1$; $j \leftarrow \mathsf{st}_j$
7 $\mathsf{st}_j.r \leftarrow\!\!\$\ \mathbb{Z}_p$; $\mathsf{st}_j.R \leftarrow g^{\mathsf{st}_j.r}$
8 $\mathsf{st}_j.t \leftarrow H_{\mathsf{com}}(\mathsf{st}_j.R)$
9 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 1$
10 Return $\mathsf{st}_j.t$

$\mathsf{Sign}_2(j, k, t_1, \ldots, t_n, \boxed{, vk_1, \ldots, vk_n, m})$:
11 If $\mathsf{st}_j.\mathrm{rnd} \neq 1$ or $\mathsf{st}_j.t \neq t_k$:
12 Return $\perp$
13 $\mathsf{st}_j.k \leftarrow k$; $\mathsf{st}_s.n \leftarrow n$
14 For i from 1 to n: $\mathsf{st}_s.t_i \leftarrow t_i$
15 $\boxed{\mathsf{Save\_Session\_Params}(j, k, vk_1, \ldots, vk_n, m)}$
16 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 2$
17 Return $\mathsf{st}_j.R$

$\mathsf{Sign}_3(j, R_1, \ldots, R_n \boxed{, m, vk_1, \ldots, vk_n})$:
18 If $\mathsf{st}_j.\mathrm{rnd} \neq 2$ or $R_{\mathsf{st}_j.k} \neq \mathsf{st}_j.R$:
19 Return $\perp$
20 If $\exists i$ such that $\mathsf{st}_j.t_i \neq H_{\mathsf{com}}(R_i)$:
21 Return $\perp$
22 $\boxed{\mathsf{Save\_Session\_Params}(j, \mathsf{st}_j.\mathrm{me}, vk_1, \ldots, vk_n, m)}$
23 $R \leftarrow \prod_{i=1}^{n} R_i$
24 $c \leftarrow H_{\mathsf{sign}}(\mathsf{st}_j.k, R, \mathsf{st}_j.vk_1, \ldots, \mathsf{st}_j.vk_n, \mathsf{st}_j.m)$
25 $z \leftarrow \mathsf{st}_j.r + \mathsf{st}.sk \cdot c$
26 $\mathsf{st}_j.\mathrm{rnd} \leftarrow 3$
27 Return (R, z)

$\mathsf{Sign}_4(R, z_1, \ldots, z_n)$:
28 Return $(R, \sum_{i=1}^{n} z_i)$

$\mathsf{Save\_Session\_Params}(j, k, vk_1, \ldots, vk_n, m)$:
29 // helper method to store session params
30 If $vk_k \neq \mathsf{st}.vk$:
31 Return $\perp$
32 $\mathsf{st}_j.m \leftarrow m$
33 For i from 1 to n: $\mathsf{st}_j.vk_i \leftarrow vk_i$

Figure 5.1: The secure Bellare-Neven multi-signature scheme, compared to the insecure version with delayed message selection. The secure version contains all but the dashed boxes, and the insecure version contains all but the solid boxes.

$\underline{\mathcal{A}_{\mathsf{idl}}^{\text{CHALLENGE}}(X)}$:

bad $\leftarrow$ false // abort whenever bad $\leftarrow$ true
$H_{\mathsf{sign}}, H_{\mathsf{com}} \leftarrow$ empty dictionary
$K \leftarrow \emptyset$ // tracks aggregated keys used
ctrs, ctrc $\leftarrow 0$ // tracks signing oracle queries and IDL challenges
$T_{\mathsf{com}}, T_{\mathsf{chal}} \leftarrow$ empty dictionary // tracks H_{com} queries, IDL challenges
st $\leftarrow$ empty list
$\rho \leftarrow\$ \{1, \ldots, q_s + 1\}$ // guess SIGNO_2 query for Case 2 forgery
Sim $\leftarrow \{(\widetilde{\text{SIGNO}}_i)_{i=1}^3, \widetilde{H_{\mathsf{sign}}}, \widetilde{H_{\mathsf{com}}}\}$
$(m, (X_i)_{i=1}^n, (R_j, z_j)_{j=1}^\ell) \leftarrow \mathcal{A}_{\text{ms}}^{\text{Sim}}(X)$
For $j = 1, \ldots, \ell$:
  $\widetilde{H_{\mathsf{sign}}}(R_j, (X_i)_{i=1}^n, m)$ // ensure initialization
  If $T_{\mathsf{chal}}[R_j, (X_i)_{i=1}^n, m]$ initialized:
    Return $(T_{\mathsf{chal}}[R_j, (X_i)_{i=1}^n, m], z_j)$
Return $\perp$

$\underline{\widetilde{\text{SIGNO}}_1()}$:

$t \leftarrow\$ \mathbb{Z}_p$
ctrs $\leftarrow$ ctrs $+ 1$, $\mathsf{st}_{\text{ctrs}}.t \leftarrow t$
Return t

$\underline{\widetilde{\text{SIGNO}}_2(j, k, (vk_i, t_i)_{i=1}^n, m)}$:

If $j = \rho$:
  $R_k \leftarrow\$ \mathbb{G}$
Else:
  $z, c \leftarrow\$ \mathbb{Z}_p$; $\mathsf{st}_j.z \leftarrow z$
  $R_k \leftarrow g^z X^{-c}$
  For $i \in \{1, \ldots, n\} \setminus \{k\}$:
    If $T_{\mathsf{com}}[t_i]$ initialized: $R_i \leftarrow T_{\mathsf{com}}[t_i]$
    Else:
      $R_i \leftarrow\$ \mathbb{Z}_p$
      If $H_{\mathsf{com}}[R_i]$ initialized: bad $\leftarrow$ true
      $H_{\mathsf{com}}[R_i] \leftarrow t_i$; $T_{\mathsf{com}}[t_i] \leftarrow R_i$
  $R \leftarrow \prod_{i=1}^n R_i$; $\mathsf{st}_j.R \leftarrow R$
  If $H_{\mathsf{sign}}[j, R, (vk_i)_{i=1}^n, m]$ initialized: bad $\leftarrow$ true
  $H_{\mathsf{sign}}[j, R, (vk_i)_{i=1}^n, m] \leftarrow c$
If $H_{\mathsf{com}}[R_k]$ initialized: bad $\leftarrow$ true
$H_{\mathsf{com}}[R_k] \leftarrow \mathsf{st}_j.t$; $T_{\mathsf{com}}[\mathsf{st}_j.t] \leftarrow R_k$
Return R_k

$\underline{\widetilde{\text{SIGNO}}_3(j, R_1, \ldots, R_n)}$:

If $j = \rho$: bad $\leftarrow$ true
Return $\mathsf{st}_j.z$

$\underline{\widetilde{H_{\mathsf{com}}}(R)}$:

If $H_{\mathsf{com}}[R]$ uninitialized:
  $H_{\mathsf{com}}[R] \leftarrow\$ \mathbb{Z}_p$;
  If $T_{\mathsf{com}}[H_{\mathsf{com}}[R]]$ initialized: bad $\leftarrow$ true
  $T_{\mathsf{com}}[H_{\mathsf{com}}[R]] \leftarrow R$
Return $H_{\mathsf{com}}[R]$

$\underline{\widetilde{H_{\mathsf{sign}}}(k, R, (vk_i)_{i=1}^n, m)}$:

If $H_{\mathsf{sign}}[k, R, (vk_i)_{i=1}^n, m]$ uninitialized:
  For i where $vk_i \neq X$:
    If $H_{\mathsf{sign}}[i, R, (vk_i)_{i=1}^n, m]$ uninitialized:
      $H_{\mathsf{sign}}[i, R, (vk_i)_{i=1}^n, m] \leftarrow\$ \mathbb{Z}_p$
    $c_i \leftarrow H_{\mathsf{sign}}[i, R, (vk_i)_{i=1}^n, m]$
  If $X \in \{vk_1, \ldots, vk_n\}$:
    $c \leftarrow \text{CHALLENGE}(R \cdot \prod_{i:vk_i \neq vk_k} vk_i^{c_i})$
    $J \leftarrow \{j : vk_j = X\}$; $j_{\max} \leftarrow \max(J)$
    For $i \in J \setminus \{j_{\max}\}$:
      $e_i \leftarrow\$ \mathbb{Z}_p$
      $H_{\mathsf{sign}}[i, R, (vk_i)_{i=1}^n, m] \leftarrow e_i$
    $H[j_{\max}, R, (vk_i)_{i=1}^n, m] \leftarrow c - \sum_{i \in J \setminus \{j_{\max}\}} e_i$
  ctrc $\leftarrow$ ctrc $+ 1$
  $T_{\mathsf{chal}}[R_j, (vk_i)_{i=1}^n, m] \leftarrow$ ctrc
Return $H_{\mathsf{sign}}[(k, R, vk_1, \ldots, vk_n, m)]$

Figure 5.2: The reduction algorithm used in the proof of Lemma 5.2.1.

Section 6

ANALYSIS OF MUSIG2

We prove the strong unforgeability of the multi-signature scheme MuSig2 [30] under the Algebraic One More Discrete Log assumption of [30] which we present in Figure 6.1 (a weaker falsifiable variant of the One More Discrete Log assumption [6]).

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 [30], 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.

6.1 Scheme Description.

We will describe the scheme informally. A formal description using our syntax is found in Figure 6.2.

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,[1] 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 $\tilde{vk} \leftarrow \prod_{i=1}^{n} vk_i^{H_{\mathsf{agg}}(i,X_1,\ldots,X_n)}$.

In the first signing round each signer k generates four[2] random values $r_{k,1}, \ldots, r_{k,4} \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends $R_{k,\ell} \leftarrow g^{r_{k,\ell}}$ for each $\ell \in \{1, \ldots, 4\}$ to all other signers. In the second round, on input

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

[2] There is a simpler variant of MuSig2 that uses only two nonces [30] that we do not consider in this paper. Its security proof relies on the algebraic group model.

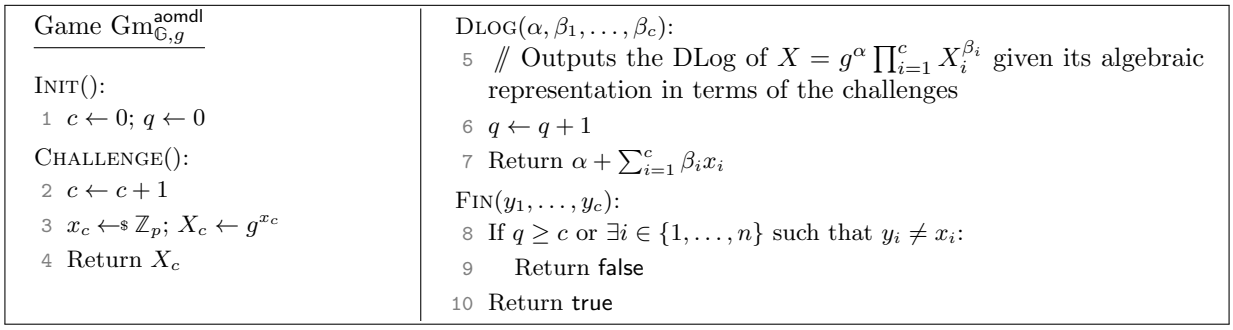

Figure 6.1: The Algebraic One More Discrete Log (AOMDL) game in a group $\mathbb{G}$ with a generator g of prime order p.

$((vk_i, R_{i,1}, \ldots, R_{i,4})_{i=1}^n, m)$, each signer k computes $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ for $\ell \in \{1, \ldots, 4\}$, the aggregate verification key $\tilde{vk}$, and $b \leftarrow H_{\mathsf{nonce}}(\tilde{vk}, R_1, \ldots, R_4, m)$. Then each signer computes the aggregate nonce $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$, the challenge $c \leftarrow H_{\mathsf{sign}}(\tilde{vk}, R, m)$, and their partial signature $z_k \leftarrow \sum_{\ell=1}^4 r_{k,\ell} \cdot b^{\ell-1} + c \cdot sk_k \cdot H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$ 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 $\tilde{vk}$ by checking that $g^z = R \cdot \tilde{vk}^{H_{\mathsf{sign}}(\tilde{vk}, R, m)}$, 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 $(R_{i,1}, \ldots, R_{i,4})_{i=1}^n$ and computes $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$ for $\ell \in \{1, \ldots, 4\}$. 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 $R_{1,\ell}, \ldots, R_{n,\ell}$ that appear uniformly random and have product R_ℓ. Therefore, without loss of generality, we do not consider this shortcut in our proof.

Scheme $\text{MuSig2}_{\mathbb{G},g,H_{\mathsf{agg}},H_{\mathsf{nonce}},H_{\mathsf{sign}}}$:

$\text{MuSig2.nr} = 3$

$\text{MuSig2.lir} = 2$

$\mathsf{KeyGen}()$:
1 $x \leftarrow\!\!\$\ \mathbb{Z}_p$; $X \leftarrow g^x$
2 $\mathsf{st}.sk \leftarrow x$; $\mathsf{st}.vk \leftarrow X$
3 Return $(sk = x, vk = X)$

$\mathsf{KeyAgg}(vk_1, \ldots, vk_n)$:
4 Return $\prod_{i=1}^{n} vk_i^{H_{\mathsf{agg}}(i, vk_1, \ldots, vk_n)}$

$\mathsf{Sign}_1()$:
5 $\mathsf{st}.j \leftarrow \mathsf{st}.j + 1$ // num of signing sessions
6 $j \leftarrow \mathsf{st}.j$, $\mathsf{st}_j.\text{rnd} \leftarrow 1$
7 For $i \in \{1, \ldots, 4\}$:
8 $\quad \mathsf{st}_j.r_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $R_i \leftarrow g^{\mathsf{st}_j.r_i}$
9 Return $(R_1, \ldots, R_4)$

$\mathsf{Sign}_2(j, k, m, (vk_i, R_{i,1}, \ldots, R_{i,4})_{i=1}^n)$:
10 If $\mathsf{st}_j.\text{rnd} \neq 1$ or $vk_k \neq \mathsf{st}.vk$:
11 $\quad$ Return $\perp$
12 $\mathsf{st}_j.\text{rnd} \leftarrow 2$
13 $\tilde{vk} \leftarrow \mathsf{KeyAgg}(vk_1, \ldots, vk_n)$
14 For $\ell = 1, \ldots, 4$:
15 $\quad R_\ell \leftarrow \prod_{i=1}^{n} R_{i,\ell}$
16 $b \leftarrow H_{\mathsf{nonce}}(\tilde{vk}, R_1, \ldots, R_4, m)$
17 $R \leftarrow \prod_{\ell=1}^{4} R_\ell^{(b^{\ell-1})}$
18 $c \leftarrow H_{\mathsf{sign}}(R, \tilde{vk}, m)$
19 $z \leftarrow \sum_{\ell=1}^{4} \mathsf{st}_j.r_\ell \cdot b^{\ell-1} + c \cdot sk_k \cdot H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$
20 Return (R, z)

$\mathsf{Sign}_3(R, z_1, \ldots, z_n)$:
21 Return $(R, \sum_{i=1}^{n} z_i)$

$\mathsf{AggVer}(m, \sigma, \tilde{vk})$:
22 $(R, z) \leftarrow \sigma$
23 Return $[g^z = R \cdot \tilde{vk}^{H_{\mathsf{sign}}(\tilde{vk}, R, m)}]$

Figure 6.2: 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.

6.2 One-More Unforgeability.

The one-more unforgeability of MuSig2 is given in the following lemma which we prove in this section.

Lemma 6.2.1 (AOMDL → SUF of MuSig2 in the ROM)**.** *Let $\mathbb{G}$ be a group with prime order p and generator g. Let $\mathsf{MS} = \text{MuSig2}[\mathbb{G}, g]$, where its hash functions are modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ be an adversary against $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ making at most q_s queries to the signing oracle SIGNO_1 and q_h queries to each random oracle. Let $q = 4q_h + 3q_s + 2$. Then, there exists an algorithm $\mathcal{D}$ such that*

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

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

PROOF IDEA: DOUBLE FORKING TECHNIQUE. The proof of the strong unforgeability of MuSig2 is nearly identical to the existential unforgeability proof of [30].

In the existential unforgeability proof, an adversary $\mathcal{A}_{\text{ms}}$ 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 $\mathcal{A}_{\text{ms}}$ outputs a forged signature (R, s) on a message m and public keys $(vk_1, \ldots, vk_n)$ where $vk_k = vk$ after making q_s signing oracle queries. First, they construct an adversary $\mathcal{B}$ that simulates $\mathbf{G}^{\text{suf-ms}}$[MuSig2] for $\mathcal{A}_{\text{ms}}$ 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 $H_{\mathsf{sign}}(R, \tilde{vk}, m)$, where $\tilde{vk}$ is the aggregate public key corresponding to $(vk_1, \ldots, vk_n)$. According to the Generalized Forking Lemma [7], 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 $H_{\mathsf{agg}}(k, vk_1, \ldots, vk_n)$. If $\mathcal{C}$ succeeds both times with the same list of public keys $vk_1, \ldots, vk_n$, 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 SIGNO_2), hence winning the AOMDL game.

We refer readers to [30] for a thorough explanation of the proof technique.

WHAT WE DO. In Lemma 6.2.2, we construct a modified adversary $\mathcal{B}$, which uses an adversary $\mathcal{A}_{\text{ms}}$ that breaks the strong unforgeability of MuSig2 (as opposed to the existential unforgeability in [30]). Our $\mathcal{B}$ takes the same input and has the same output as the algorithm $\mathcal{B}$ in the proof of [30] and uses its input in the exact same way. Thus, once our Lemma 6.2.2 is established, we can plug in our $\mathcal{B}$ to Lemma 3 and Theorem 1 of [30], and obtain a proof of the strong unforgeability of MuSig2 as stated in Lemma 6.2.1.

Lemma 6.2.2 (underlying algorithm for the rewinding based reduction)**.** *Let $\mathbb{G}$ be a group of prime order p with generator g. Let $\mathsf{MS} = \text{MuSig2}[\mathbb{G}, g]$ be the associated multi-signature scheme, where*

its hash functions are modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ *be an adversary against* $\mathbf{G}^{\text{suf-ms}}[\mathsf{MS}]$ *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* $X, U_1, \ldots, U_{4q_s}$ *are AOMDL challenges and* $h_1, \ldots, h_q \leftarrow\!\!\$\ \mathbb{Z}_p$, *then with probability*

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

the adversary $\mathcal{B}$ *defined in Figure 6.3 makes at most* q_s *queries to the AOMDL discrete log oracle and outputs a tuple* $(i_{\mathsf{sign}}, i_{\mathsf{agg}}, X_1, \ldots, X_n, R, z, a_1, \ldots, a_n)$ *where* $i_{\mathsf{sign}}, i_{\mathsf{agg}} \in \{1, \ldots, q\}$, $i_{\mathsf{sign}} < i_{\mathsf{agg}}$, $X \in \{X_1, \ldots, X_n\}$, $a_1, \ldots, a_n \in \mathbb{Z}_p$, $\sum_{\{i:\ X_i = X\}} a_i = h_{i_{\mathsf{agg}}}$, *and*

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

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

Proof of Lemma 6.2.2: We will first prove that all the requirement hold whenever $\mathcal{A}_{\text{ms}}$ and $\mathcal{B}$ succeed, and then analyze its success probability.

First, note that $\mathcal{B}$ makes a single DLOG query whenever $\mathcal{A}_{\text{ms}}$ 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 $\mathcal{A}_{\text{ms}}$ 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 $\widetilde{H_{\mathsf{nonce}}}$ 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 $\mathcal{A}_{\text{ms}}$ succeeds, then it outputs more signatures than the number of signatures for the corresponding message m and public keys $X_1, \ldots, X_n$ produced by the signing oracle. In particular, for some j, the signing oracle did not return a signature for $(m, X_1, \ldots, X_n)$ 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 $\mathcal{A}_{\text{ms}}$ succeeds then it outputs a list of public keys containing X, and hence $X \in \{X_1, \ldots, X_n\}$. Furthermore, by construction i_{agg} is the index such that

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

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

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

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

Now, we will lower bound success probability of $\mathcal{B}$. As we have shown, if $\mathcal{A}_{\mathsf{MS}}$ 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}, $\textsc{SignO}_2$, and on the output of $\mathcal{A}_{\text{ms}}$. Each time, the probability that those events happen are bounded above by $|H_{\mathsf{sign}}|/p \leq q/p$ and $|K|/p \leq q/p$, resulting in a total probability bounded above by $2q^2/p$.

Hence, $\mathcal{B}$ succeeds with probability of at least $\Pr[\mathbf{G}^{\text{suf-ms}}_{\mathsf{MS}}(\mathcal{A}_{\text{ms}})] - 2q^2/p$, 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 [30]. ∎

$\underline{\mathcal{B}^{\text{DLOG}}(X^*, U_1, \ldots, U_{4q_s}, h_1, \ldots, h_q):}$

bad $\leftarrow$ false // abort whenever bad $\leftarrow$ true
$H_{\mathsf{agg}}, H_{\mathsf{sign}}, H_{\mathsf{nonce}} \leftarrow$ empty dictionary
ctrh $\leftarrow 0$ // counter for random oracle queries
ctrs $\leftarrow 0$ // counter for signing oracle queries
$Q \leftarrow \emptyset$ // tracks signing oracle responses
$K \leftarrow \emptyset$ // tracks aggregated keys used
$T_{\mathsf{agg}}, T_{\mathsf{sign}} \leftarrow$ empty dictionary // tracks indices in h for random oracle replies
$\rho_{\mathcal{A}} \leftarrow\!\!\$\ \{0,1\}^{R_{\mathcal{A}}}$ // random tape for $\mathcal{A}$
$\text{Sim} \leftarrow \{\widetilde{\text{SIGNO}}_1, \widetilde{\text{SIGNO}}_2, \widetilde{H_{\mathsf{agg}}}, \widetilde{H_{\mathsf{sign}}}, \widetilde{H_{\mathsf{nonce}}}\}$
$(m, (X_i)_{i=1}^n, (R_j, z_j)_{j=1}^{\ell}) \leftarrow \mathcal{A}_{\text{ms}}^{\text{Sim}}(X; \rho_{\mathcal{A}})$
$\tilde{X} \leftarrow \prod_{i=1}^n X_i^{\widetilde{H_{\mathsf{agg}}}(i, X_1, \ldots, X_n)}$
If $\exists j \in \{1, \ldots, \ell\}$ such that $(R_j, \tilde{X}, m) \notin Q$:
  $\widetilde{H_{\mathsf{sign}}}(R_j, \tilde{X}, m)$ // ensure initialization
  $i_{\mathsf{sign}} \leftarrow T_{\mathsf{sign}}[R_j, \tilde{X}, m]$
  $i_{\mathsf{agg}} \leftarrow T_{\mathsf{agg}}[X_1, \ldots, X_n]$
  For $i = 1, \ldots, n$:
    $a_i \leftarrow \widetilde{H_{\mathsf{agg}}}(i, X_1, \ldots, X_n)$
  Return $(i_{\mathsf{sign}}, i_{\mathsf{agg}}, X_1, \ldots, X_n, R_j, z_j, a_1, \ldots, a_n)$
Return $\perp$

$\underline{\widetilde{\text{SIGNO}}_1():}$

ctrs $\leftarrow$ ctrs $+ 1$
$\hat{j} \leftarrow 4(\text{ctrs} - 1)$
Return $U_{\hat{j}+1}, U_{\hat{j}+2}, U_{\hat{j}+3}, U_{\hat{j}+4}$

$\underline{\widetilde{\text{SIGNO}}_2(j, k, m, (vk_i, R_{i,1}, \ldots, R_{i,4})_{i=1}^n):}$

$j' \leftarrow 4(\text{ctrs} - 1)$
$(R_{k,1}, \ldots, R_{k,4}) \leftarrow U_{\hat{j}+1}, \ldots, U_{\hat{j}+4}$
$\tilde{X} \leftarrow \prod_{i=1}^n vk_i^{\widetilde{H_{\mathsf{agg}}}(i, vk_1, \ldots, vk_n)}$
For $\ell = 1, \ldots, 4$:
  $R_\ell \leftarrow \prod_{i=1}^n R_{i,\ell}$
$b \leftarrow \widetilde{H_{\mathsf{nonce}}}(\tilde{vk}, m, R_1, \ldots, R_4)$
$R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$
$Q \leftarrow Q \cup \{(R, \tilde{vk}, m)\}$
$c \leftarrow \widetilde{H_{\mathsf{sign}}}(\tilde{vk}, R, m)$
// query dlog oracle for the partial signature:
$\alpha \leftarrow 0$; $\beta_1 \leftarrow c \cdot \widetilde{H_{\mathsf{agg}}}(k, vk_1, \ldots, vk_n)$
For $i = 2, \ldots, 4q_s + 1$:
  $\beta_i \leftarrow 0$
$(\beta_{j'+2, \ldots, j'+5}) \leftarrow (b^0, \ldots, b^3)$
$z \leftarrow \text{DLOG}(\alpha, \beta_1, \ldots, \beta_{4q_s})$
Return (R, z)

$\underline{\widetilde{H_{\mathsf{agg}}}(k, X_1, \ldots, X_n):}$

If $H_{\mathsf{agg}}[k, X_1, \ldots, X_n]$ uninitialized:
  If $X \in \{X_1, \ldots, X_n\}$:
    ctrh $\leftarrow$ ctrh $+ 1$
    $v \leftarrow \max\{i \colon X_i = X\}$
    $s \leftarrow 0$
    For $i = 1, \ldots, v-1$:
      If $X_i = X$:
        $H_{\mathsf{agg}}[i, X_1, \ldots, X_n] \leftarrow\!\!\$\ \mathbb{Z}_p$
        $s \leftarrow s + H_{\mathsf{agg}}[i, X_1, \ldots, X_n]$
    $H_{\mathsf{agg}}[v, X_1, \ldots, X_n] \leftarrow h_{\text{ctrh}} - s$
    $T_{\mathsf{agg}}[X_1, \ldots, X_n] \leftarrow$ ctrh
  For $i = 1, \ldots, n$:
    If $X_i \neq X$: $H_{\mathsf{agg}}[i, X_1, \ldots, X_n] \leftarrow\!\!\$\ \mathbb{Z}_p$
  $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_{\mathsf{agg}}[i, X_1, \ldots, X_n]}$
  If $\exists (R, m)$ such that $H_{\mathsf{sign}}[R, \tilde{X}, m]$ is initialized:
    bad $\leftarrow$ true
  If $\tilde{X} \in K$:
    bad $\leftarrow$ true
  $K \leftarrow K \cup \{\tilde{X}\}$
Return $H_{\mathsf{agg}}[k, X_1, \ldots, X_n]$

$\underline{\widetilde{H_{\mathsf{nonce}}}(\tilde{X}, R_1, \ldots, R_4, m):}$

If $H_{\mathsf{nonce}}[\tilde{X}, R_1, \ldots, R_4, m]$ uninitialized:
  ctrh $\leftarrow$ ctrh $+ 1$
  $H_{\mathsf{nonce}}[\tilde{X}, R_1, \ldots, R_4, m] \leftarrow h_{\text{ctrh}}$
  $b \leftarrow H_{\mathsf{nonce}}[\tilde{X}, R_1, \ldots, R_4, m]$
  $R \leftarrow \prod_{\ell=1}^4 R_\ell^{(b^{\ell-1})}$
  $\widetilde{H_{\mathsf{sign}}}(R, \tilde{X}, m)$ // ensure initialization
Return $H_{\mathsf{nonce}}[\tilde{X}, R_1, \ldots, R_4, m]$

$\underline{\widetilde{H_{\mathsf{sign}}}(R, \tilde{X}, m):}$

If $H_{\mathsf{sign}}[R, \tilde{X}, m]$ uninitialized:
  ctrh $\leftarrow$ ctrh $+ 1$
  $H_{\mathsf{sign}}[R, \tilde{X}, m] \leftarrow h_{\text{ctrh}}$
  $T_{\mathsf{sign}}[R, \tilde{X}, m] \leftarrow$ ctrh
Return $H_{\mathsf{sign}}[R, \tilde{X}, m]$

Figure 6.3: The algorithm from Lemma 6.2.2, which is used to prove the strong unforgeability of MuSig2. It is very similar to Figure 7 of [30].

Section 7

ANALYSIS OF HBMS

In [5] 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. [11] to solve the ROS problem [35], which broke the unforgeability of many multi-signature schemes including an older variant of MuSig [23].

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 Section 7.3 we formally define this weaker security notion (which we call non-interactive strong unforgeability), and we prove that it is satisfied by HBMS.

7.1 Scheme Description

We describe the HBMS scheme informally. A formal description of the scheme is given in Figure 9 of [5].

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, 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 $\tilde{X} \leftarrow \prod_{i=1}^{n} X_i^{H_2(i,X_1,\ldots,X_n)}$.

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 $h \leftarrow H_0(X_1, \ldots, X_n, m) \in \mathbb{G}$, 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 $(M_1, \ldots, M_n)$ 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 $z_i \leftarrow r_i + x_i \cdot c \cdot H_2(i, X_1, \ldots, X_n)$, 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 $(X_1, \ldots, X_n)$ and a message m, the verifier computes $h \leftarrow H_0(X_1, \ldots, X_n, m)$ and $\tilde{X} \leftarrow \prod_{i=1}^{n} X_i^{H_2(i,X_1,\ldots,X_n)}$, 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 [5] proves the existential unforgeability of HBMS.

7.2 Attack Against One-More Unforgeability

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 $j \in \{1, \ldots, \ell\}$, 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 $j \in \{1, \ldots, \ell\}$ and $b \in \{0, 1\}$, the adversary samples $\overline{r}_j^b$ and $\overline{s}_j^b$ uniformly at random from $\mathbb{Z}_p$ and computes $\overline{N}_j^b \leftarrow h^{\overline{s}_j^b} g^{\overline{r}_j^b}$ and $\overline{T}_j^b \leftarrow N_j \cdot \overline{N}_j^b$ as well as $\overline{c}_j^b \leftarrow H_1(\overline{T}_j^b, \tilde{X}, m)$. The adversary must also ensure that all of the $\overline{T}_j^{b_j}$ are distinct and that $\overline{c}_j^0 \neq \overline{c}_j^1$ for each j by regenerating the nonces if needed.

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

$$\rho^+(g_1, \ldots, g_\ell) = \sum_{j=1}^{\ell} \frac{2^{j-1} g_j}{\overline{c}_j^1 - \overline{c}_j^0}$$

and

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

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

Next, continue to the second round of each signing session j by sending $\overline{N}_j^{b_j}$ 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

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

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

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

We will prove below that all $\ell + 1$ signatures $(\sigma_1, \ldots, \sigma_\ell, \sigma_{\ell+1})$ 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 $\sigma_1, \ldots, \sigma_\ell$. Since all of the $\overline{T}_j^{b_j}$ are distinct, all of the σ_j are distinct for $j \in \{1, \ldots, \ell\}$. 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 $\sigma_{\ell+1} = (T_{\ell+1}, \rho^+(s_1, \ldots, s_\ell), \rho^+(z_1, \ldots, z_\ell) + c_{\ell+1} \cdot x_1 \cdot H_2(1, X_1, X_2))$ 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 $\{\overline{T}_1^{b_1}, \ldots, \overline{T}_\ell^{b_\ell}\}$ is selected uniformly at random from all subsets of $\mathbb{G}$ of cardinality ℓ, independently of $(N_1, \ldots, N_\ell)$. Hence, the probability that is contains $T_{\ell+1} = \rho^\times(N_1, \ldots, N_\ell)$ is $\frac{\ell}{|\mathbb{G}|} \approx \frac{\log_2(p)}{p}$, which is very small. Hence, with large probability, $T_{\ell+1} \neq \overline{T}_j^{b_j}$ and therefore $\sigma_{\ell+1} \neq \sigma_j$ for all $j \in \{1, \ldots, \ell\}$.

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

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

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

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

Applying Lemma 7.2.1, which states that $c_{\ell+1} = \rho^+(\overline{c}_1^{b_1}, \ldots, \overline{c}_\ell^{b_\ell})$, we can simplify the equation to

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

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

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

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

Lemma 7.2.1. *By the construction above,* $c_{\ell+1} = \rho^+(\overline{c}_1^{b_1}, \ldots, \overline{c}_\ell^{b_\ell})$.

This lemma is at the heart of the attack, and the construction allowing this lemma to hold is precisely the algorithm of [11] to solve the ROS problem.

Proof of Lemma 7.2.1 The proof of this lemma is identical to that of 4.3.1.

7.3 SUF of Underlying Plain Signature Scheme

In this section we prove that HBMS is 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 HBMS is 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 7.1 for a formal definition.

We use the XIDL game (Figure 2.1) 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 2.0.2), this proves the non-interactive strong unforgeability of HBMS under the discrete log assumption.

Lemma 7.3.1 (XIDL $\rightarrow$ NISUF of HBMS in the ROM)**.** *Let $\mathbb{G}$ be a group of prime order p with generator g. Let $\mathsf{MS} = \text{HBMS}[\mathbb{G}, g]$ be the associated multi-signature scheme, where its hash functions are modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ be an adversary against $\mathbf{G}_n^{\text{ni-suf-ms}}[\mathsf{MS}]$ that makes at most q_0, q_1, q_s queries to H_0, H_1, and* SignO *respectively. Then, we can construct an*

Games $\mathbf{G}_n^{\text{ni-suf-ms}}[\mathsf{MS}]$

INIT():
1 For $i = 1, \ldots, n$:
2 $\quad (vk_i, sk_i) \leftarrow\!\!\$\ \mathsf{MS.Kg}()$
3 $\mathrm{Q} \leftarrow \emptyset$ // message-signature pairs obtained legitimately
4 Return $(vk_i)_{i=1}^n$

SIGNO(m):
5 $\sigma \leftarrow \mathsf{Exec}_{\mathsf{MS}}((vk_i)_{i=1}^n), (sk_i)_{i=1}^n, m)$
6 $\mathrm{Q} \leftarrow \mathrm{Q} \cup \{(m, \sigma)\}$
7 Return σ

FIN(m, σ):
8 If $\mathsf{MS.Verify}((vk_i)_{i=1}^n, m, \sigma)$:
9 $\quad$ If $(m, \sigma) \notin \mathrm{Q}$:
10 $\quad\quad$ Return true
11 Return false

Figure 7.1: Game used to define the "non-interactive strong unforgeability" of a multi-signature scheme MS.

---

adversary $\mathcal{A}_{\mathsf{xidl}}$ *against* $\mathrm{Gm}^{\mathsf{xidl}}_{\mathbb{G},g,1,q_1+1}$ *such that*

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

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

PROOF IDEA FOR LEMMA 7.3.1. Suppose an adversary $\mathcal{A}_{\text{ms}}$ break the non-interactive strong unforgeability of HBMS (i.e. wins the game in Figure 7.2). 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 $h = H_0(m, X_1, \ldots, X_n)$. If $\tilde{X}$ is an XIDL target and $H_1(T, \tilde{X}, m)$ is an XIDL challenge, then the element $z + s \cdot \mathrm{DLOG}_g(h)$ wins the XIDL game, where the reduction can know $\mathrm{DLOG}_g(h)$ 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 $c, z' \leftarrow\!\!\$\ \mathbb{Z}_p$, 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 $s \leftarrow\!\!\$\ \mathbb{Z}_p$, and the triple $(s, T, z' - s\mathrm{DLOG}_g(h))$ is a valid HBMS signature (where $h = H_0(m, X_1, \ldots, X_n)$). 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 $H_0(m, X_1, \ldots, X_n)$ 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 of Lemma 7.3.1: In Figure 7.2 we construct an adversary $\mathcal{A}_{\mathsf{xidl}}$ which plays the XIDL game by executing $\mathcal{A}_{\text{ms}}$ 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 $\mathcal{A}_{\text{ms}}$ are well-formed, since otherwise we can simply simulate their responses uniformly at random. We will now analyze its success probability.

Now, suppose $\mathcal{A}_{\text{ms}}$ wins $\mathbf{G}_n^{\text{ni-suf-ms}}[\mathsf{MS}]$ by returning some forgery (m, σ) where $\sigma = (s, z, T)$. Let $h = H_0(m, X_1, \ldots, X_n)$ 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 $\mathcal{B}_{\mathsf{xidl}}$ 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 $\mathcal{A}_{\text{ms}}$ and we have a forgery of the type corresponding to this case, then $C[T]$ would be initialized in the execution of $\mathcal{A}_{\text{ms}}$ 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 $T = T'$.

Game $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{HBMS}_{\mathbb{G},g,H_0,H_1,H_2}]$

INIT():
1 For $i = 1, \ldots, n$:
2 $\quad x_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $X_i \leftarrow g^x$
3 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i,X_1,\ldots,X_n)}$
4 $\tilde{x} \leftarrow \sum_{i=1}^n x_i \cdot H_2(i, X_1, \ldots, X_n)$
5 $Q \leftarrow \emptyset$
6 Return $(X_1, \ldots, X_n)$

SIGNO(m):
7 $h \leftarrow H_0(X_1, \ldots, X_n, m)$
8 $s, r \leftarrow\!\!\$\ \mathbb{Z}_p$
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)

FIN(m, σ):
14 If $(m, \sigma) \in Q$: Return false
15 Parse σ as (s, T, z)
16 $h \leftarrow H_0(X_1, \ldots, X_n, m)$
17 Return $[g^z h^s = T \cdot \tilde{X}^{H_1(T,\tilde{X},m)}]$

$\mathcal{A}_{\mathsf{xidl}}^{\text{NEWTARGET,CHALLENGE}}(X)$:
1 // randomly selects whether to execute $\mathcal{B}_{\mathsf{xidl}}$ or $\mathcal{C}_{\mathsf{dl}}$, presented in Figures 7.3 and 7.4
2 $I \leftarrow\!\!\$\ \{0, 1, \ldots, q_0 + q_s\}$
3 If $I = 0$:
4 $\quad$ Return $\mathcal{B}_{\mathsf{xidl}}^{\text{NEWTARGET,CHALLENGE}}(X)$
5 Else:
6 $\quad x \leftarrow \mathcal{C}_{\mathsf{dl}}(X, I)$
7 $\quad$ If $x = \perp$ or $g^x \neq X$: Return $\perp$
8 $\quad e \leftarrow \text{NEWTARGET}(1_{\mathbb{G}})$
9 $\quad c \leftarrow \text{CHALLENGE}(1, 1_{\mathbb{G}})$
10 $\quad$ Return $(1, x \cdot c \cdot e)$

Figure 7.2: **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 7.3.1, where the referenced subroutines are in Figures 7.3 and 7.4.

In this case, there was a call to $\widetilde{H_0}(m, X_1, \ldots, X_n)$ by the signing oracle (or the adversary). Suppose this call was the ℓ's value programmed by $\widetilde{H_0}$. Then, if we guessed $\rho = \ell$, then $\mathcal{C}_{\mathsf{dl}}$ correctly finds the discrete log of X, allowing $\mathcal{A}_{\mathsf{xidl}}$ to win the XIDL.

Thus, in both possible cases, we have that

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

which is what we wanted to prove. ∎

$\underline{\mathcal{B}_{\mathsf{xidl}}^{\text{NewTarget,Challenge}}(X)}$:
1 $\mathsf{bad} \leftarrow \mathsf{false}$ // Abort whenever $\mathsf{bad} \leftarrow \mathsf{true}$
2 $H_0, H_1, H_2, H_0^{\mathsf{dl}}, C \leftarrow$ empty dictionaries
3 $q \leftarrow 0$ // tracks $\widetilde{H_1}$ queries
4 $X_1 \leftarrow X$
5 For $i = 2, \ldots, n$:
6 $\quad x_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $X_i \leftarrow g^x$
7 For $i = 2, \ldots, n-1$
8 $\quad \alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p$
9 $\quad H_2[i, X_1, \ldots, X_n] \leftarrow \alpha_i$
10 $H_2[1, X_1, \ldots, X_n] \leftarrow \text{NewTarget}(\prod_{i=2}^n X_i^{\alpha_i})$
11 $\tilde{X} \leftarrow \prod_{i=1}^n X_i^{H_2(i, X_1, \ldots, X_n)}$
12 $(m, (s, z, T)) \leftarrow \mathcal{A}_{\text{ms}}^{\widetilde{\text{SignO}}, \widetilde{H_0}, \widetilde{H_1}, \widetilde{H_2}}(X_1, \ldots, X_n)$
13 $\widetilde{H_0}(m, X_1, \ldots, X_n)$ // ensure initialization
14 $\widetilde{H_2}(T, \tilde{X}, m)$ // ensure initialization
15 $k \leftarrow H_0^{\mathsf{dl}}[m, X_1, \ldots, X_n]$
16 If $C[T]$ uninitialized: $\mathsf{bad} \leftarrow \mathsf{true}$
17 Return $(C[T], z + s \cdot k)$

$\underline{\widetilde{\text{SignO}}(m)}$:
18 $z', c, s \leftarrow\!\!\$\ \mathbb{Z}_p$
19 $T \leftarrow g^{z'} X^{-c}$
20 If $H_1[T, \tilde{X}, m]$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
21 $H_1[T, \tilde{X}, m] \leftarrow c$
22 $h \leftarrow \widetilde{H_0}(X_1, \ldots, X_n, m)$
23 $k \leftarrow H_0^{\mathsf{dl}}[X_1, \ldots, X_n, m]$
24 $z \leftarrow z' - k \cdot s$
25 Return (s, z, T)

$\underline{\widetilde{H_0}(X_1', \ldots, X_n', m)}$:
26 If $H_0[X_1', \ldots, X_n', m]$ uninitialized:
27 $\quad k \leftarrow\!\!\$\ \mathbb{Z}_p$
28 $\quad H_0[X_1', \ldots, X_n', m] \leftarrow g^k$
29 $\quad H_0^{\mathsf{dl}}[X_1', \ldots, X_n', m] \leftarrow k$
30 Return $H_0[X_1', \ldots, X_n', m]$

$\underline{\widetilde{H_1}(T, X', m)}$:
31 If $H_1[T, X', m]$ uninitialized:
32 $\quad$ If $X' = \tilde{X}$:
33 $\quad\quad c \leftarrow \text{Challenge}(1, T)$
34 $\quad\quad q \leftarrow q + 1$
35 $\quad\quad C[T] \leftarrow q$
36 $\quad\quad H_1[T, X', m] \leftarrow c$
37 $\quad$ Else: $H_1[T, X', m] \leftarrow\!\!\$\ \mathbb{Z}_p$
38 Return $H_1[T, X', m]$

$\underline{\widetilde{H_2}(i, X_1', \ldots, X_n')}$:
39 If $H_2[i, X_1', \ldots, X_n']$ uninitialized:
40 $\quad H_2[i, X_1', \ldots, X_n'] \leftarrow\!\!\$\ \mathbb{Z}_p$
41 Return $H_2[i, X_1', \ldots, X_n']$

Figure 7.3: Algorithms $\mathcal{B}_{\mathsf{xidl}}$, a subroutine used by $\mathcal{A}_{\mathsf{xidl}}$ of Figure 7.2 in the proof of Lemma 7.3.1.

$\underline{\mathcal{C}_{\mathsf{dl}}(X, I)}$:
$H_0, H_1, H_2, Q \leftarrow$ empty dictionaries
$q \leftarrow 0$ // tracks $\tilde{H}_1$ queries
$\ell \leftarrow 0$ // tracks signatures
For $i = 1, \ldots, n$:
  $x_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $X_i \leftarrow g^x$
$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i^{\widetilde{H_2}(i, X_1, \ldots, X_n)}$
$\tilde{x} \leftarrow \sum_{i=1}^{n} x_i \cdot \widetilde{H_2}(i, X_1, \ldots, X_n)$
$(m, (s, z, T)) \leftarrow \mathcal{A}_{\mathrm{ms}}^{\widetilde{\textsc{SignO}}, \widetilde{H_0}, \widetilde{H_1}, \widetilde{H_2}}(X_1, \ldots, X_n)$
$h \leftarrow \widetilde{H_0}(X_1, \ldots, X_n, m)$
If $h = X$:
  For $i \in \{1, \ldots, |Q|\}$:
    $(s', z', T') \leftarrow Q[i]$
    If $h^s g^z = h^{s'} g^{z'}$ and $(s, z) \neq (s', z')$:
      Return $(z - z')(s' - s)^{-1}$
Return $\perp$

$\underline{\widetilde{\textsc{SignO}}(m)}$:
$h \leftarrow \widetilde{H_0}(X_1, \ldots, X_n, m)$
$s, r \leftarrow\!\!\$\ \mathbb{Z}_p$; $T \leftarrow h^s g^r$; $c \leftarrow \widetilde{H_1}(T, \tilde{X}, m)$
$z \leftarrow r + c \cdot \tilde{x}$
$i \leftarrow i + 1$
$Q[i] \leftarrow (s, T, z)$
Return (s, T, z)

$\underline{\widetilde{H_0}(X'_1, \ldots, X'_n, m)}$:
If $H_0[X'_1, \ldots, X'_n, m]$ uninitialized:
  $q \leftarrow q + 1$
  If $q = I$: $H_0[X'_1, \ldots, X'_n, m] \leftarrow X$
  Else: $H_0[X'_1, \ldots, X'_n, m] \leftarrow\!\!\$\ \mathbb{G}$
Return $H_0[X'_1, \ldots, X'_n, m]$

$\underline{\widetilde{H_1}(T, X', m)}$:
If $H_1[T, X', m]$ uninitialized:
  $H_1[T, X', m] \leftarrow\!\!\$\ \mathbb{Z}_p$
Return $H_1[T, X', m]$

$\underline{\widetilde{H_2}(i, X'_1, \ldots, X'_n)}$:
If $H_2[i, X'_1, \ldots, X'_n]$ uninitialized:
  $H_2[i, X'_1, \ldots, X'_n] \leftarrow\!\!\$\ \mathbb{Z}_p$
Return $H_2[i, X'_1, \ldots, X'_n]$

Figure 7.4: Algorithms $\mathcal{C}_{\mathsf{dl}}$, subroutines used by $\mathcal{A}_{\mathsf{xidl}}$ of Figure 7.2 in the proof of Lemma 7.3.1.

Section 8

ANALYSIS OF MBCJ

In [2], 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 [16], using Wagner's algorithm for the generalized birthday problem [39]. This attack can be improved to polynomial time using the algorithm of Benhamouda et al. to solve the ROS problem [11]. As an alternative to the insecure BCJ, Drijvers et al. present mBCJ, a modification to the scheme that prevents the mentioned attacks [16].

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 Section 7.3, and prove it for mBCJ in Section 8.4). In other words, mBCJ produces strongly unforgeable plain signatures in a distributed way, without being strongly unforgeable itself.

8.1 Scheme Description and Security Model

Security proof of mBCJ, and our results. The existential unforgeability of mBCJ is proved under the discrete log assumption [16], but using a weaker security definition than ours. First, the scheme is presented in the key-verification model [3], 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 [7], 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 [16], 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 signer 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 $\overline{X} \leftarrow \prod_j X_j$, 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 $c \leftarrow H_0(\bar{t}_1, \bar{t}_2, \overline{X}, m)$. 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 $(\bar{t}_1, \bar{t}_2, \bar{s}, \overline{\alpha}_1, \overline{\alpha}_2)$, where $\bar{s} \leftarrow \sum_j s_j$, $\overline{\alpha}_1 \leftarrow \sum_j \alpha_{j,1}$, and $\overline{\alpha}_2 \leftarrow \sum_j \alpha_{j,2}$.

To verify a signature $(\bar{t}_1, \bar{t}_2, \bar{s}, \overline{\alpha}_1, \overline{\alpha}_2)$ with respect to a message m and keys $vk_1 = (X_1, \pi_1), \ldots, vk_n = (X_n, \pi_n)$, 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 $\overline{X} \leftarrow \prod_j X_j$, and the challenge $c \leftarrow H_0(\bar{t}_1, \bar{t}_2, \overline{X}, m)$. The verification returns true if and only if $\bar{t}_1 = g_1^{\overline{\alpha}_1} h_1^{\overline{\alpha}_2}$ and $\bar{t}_2 = g_2^{\overline{\alpha}_1} h_2^{\overline{\alpha}_2} g_1^{\bar{s}} \cdot \overline{X}^{-c}$.

8.2 One-More Unforgeability Attack.

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.

NOTATION.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.

ATTACK.

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 $j \in \{1, \ldots, \ell\}$ 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 $\rho^+ \colon (\mathbb{Z}_p)^\ell \to \mathbb{Z}_p$ and $\rho^\times \colon \mathbb{G}^\ell \to \mathbb{G}$ given by

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

and

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

and define $\bar{t}_{1,\ell+1} \leftarrow \rho^\times(t_{2,1,1}, \ldots, t_{2,1,\ell})$ and $\bar{t}_{2,\ell+1} \leftarrow \rho^\times(t_{2,2,1}, \ldots, t_{2,2,\ell})$. Let $c_{\ell+1} \leftarrow H_0(\bar{t}_{1,\ell+1}, \bar{t}_{2,\ell+1}, X_1 X_2, m)$, let $d \leftarrow c_{\ell+1} - \rho^+(c_1^0, \ldots, c_\ell^0)$, and write it in binary as $d = \sum_{j=1}^{\ell} 2^{j-1} b_j$for some $b_1, \ldots, b_\ell \in \{0,1\}$, 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

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

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

Note that $\sigma_1, \ldots, \sigma_\ell$ where 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 $\bar{t}_{1,\ell+1} = g_1^{\overline{\alpha}_{1,\ell+1}} h_1^{\overline{\alpha}_{2,\ell+1}}$ and $\bar{t}_{2,\ell+1} = g_2^{\overline{\alpha}_{1,\ell+1}} h_2^{\overline{\alpha}_{2,\ell+1}} g_1^{\overline{s}_{\ell+1}} \cdot (X_1 X_2)^{-c_{\ell+1}}$. Starting from the left hand side of the first equation, we have that

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

which is what we wanted to show.

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

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

However, by a proof identical to the proof of Lemma 4.3.1 (with slightly different notation), $c_{\ell+1} = \rho^+(c_1^{b_1}, \ldots, c_\ell^{b_\ell})$. Consequently, the exponent of X_2 cancels out, leaving us with $\bar{t}_{2,\ell+1}$, which completes the proof.

8.3 Forging a Signature with Arbitrary Signing Groups

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 $\overline{s} \leftarrow \rho^{+}(s_{2,1}, \ldots, s_{2,\ell}) + x_{\text{evil}} \cdot c_{\ell+1}$, 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.

8.4 SUF of Underlying Plain Signature Scheme

To prove the non-interactive strong unforgeability of mBCJ, we once again use the chain reductions of Bellare and Dai [5]. This time we use the Identification Discrete Logarithm game (IDL) [21, 5], presented in Figure 2.1, as the underlying assumption in our proof, which Bellare and Dai show is hard to win whenever the discrete log problem is hard.[1] In Lemma 2.0.1 we repeat their result regarding the difficulty of IDL under the DL assumption, and in Lemma 8.4.1 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 8.4.1 (IDL $\rightarrow$ NISUF of mBCJ in the ROM)**.** *Let $\mathbb{G}$ be a group of prime order p with a generator g_1. Let $\mathsf{MS} = \text{mBCJ}[\mathbb{G}, g_1]$ be the associated multi-signature scheme with its hash functions modeled as random oracles. Let $\mathcal{A}_{\text{ms}}$ be an adversary against $\mathbf{G}_n^{\text{ni-suf-ms}}[\mathsf{MS}]$ 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 $\mathcal{A}_{\mathsf{idl}}$ against $\text{Gm}^{\mathsf{idl}}_{\mathbb{G},g_1,q_0+1}$ such that*

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

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

[1] They also achieve tighter security bounds using the algebraic group model [17], but this is orthogonal to this paper.

Game $\mathbf{G}_n^{\text{ni-suf-ms}}[\text{mBCJ}_{\mathbb{G},g_1,H_0,H_1,H_2}]$

INIT():
1 For $i = 1, \ldots, n$:
2 $\quad x_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $X_i \leftarrow g^x$
3 $\quad r_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $c_i \leftarrow H_1(X, g_1^{r_i})$; $z_i \leftarrow r_i + c_i \cdot x_i$
4 $\quad \pi_i \leftarrow (c_i, z_i)$
5 $\overline{X} \leftarrow \prod_{i=1}^n X_i$; $\overline{x} \leftarrow \sum_{i=1}^n x_i$
6 $Q \leftarrow \emptyset$
7 Return $((X_1, \pi_1), \ldots, (X_n, \pi_n))$

FIN(m, σ):
8 If $(m, \sigma) \in Q$: Return false
9 Parse σ as $(t_1, t_2, s, \alpha_1, \alpha_2)$
10 $(g_2, h_1, h_2) \leftarrow H_2(m)$; $c \leftarrow H_0(t_1, t_2, \overline{X}, m)$
11 Return $[t_1 = g_1^{\alpha_1} h_1^{\alpha_2}$ and $t_2 = g_2^{\alpha_1} h_2^{\alpha_2} g_1^s \cdot \overline{X}^{-c}]$

SIGNO(m):
12 $(g_2, h_1, h_2) \leftarrow H_2(m)$
13 $r, \alpha_1, \alpha_2 \leftarrow\!\!\$\ \mathbb{Z}_p$
14 $t_1 \leftarrow g_1^{\alpha_1} h_1^{\alpha_2}$; $t_2 = g_2^{\alpha_1} h_2^{\alpha_2} g_1^r$
15 $c \leftarrow H_0(t_1, t_2, \overline{X}, m)$; $s \leftarrow r + \overline{x}c$
16 $\sigma \leftarrow (t_1, t_2, s, \alpha_1, \alpha_2)$
17 $Q \leftarrow Q \cup \{(m, \sigma)\}$
18 Return σ

Figure 8.1: The non-interactive strong unforgeability game of mBCJ over a group $\mathbb{G}$ with a generator g_1 of order p.

PROOF IDEA FOR LEMMA 8.4.1. Suppose an adversary $\mathcal{A}_{\text{ms}}$ can break the non-interactive strong unforgeability of mBCJ (i.e. wins the game in Figure 8.1 in the ROM). We will use $\mathcal{A}_{\text{ms}}$ to construct an adversary $\mathcal{A}_{\mathsf{idl}}$ that wins the IDL game (Figure 2.1) when parameterized by the group used for mBCJ and the generator g_1.

On input X, $\mathcal{A}_{\mathsf{idl}}$ 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 $i \in \{1, \ldots, n-1\}$ 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 $\mathcal{A}_{\text{ms}}$ with those public keys and proofs of possessions as its input, simulating its oracles as we describe below.

$\mathcal{A}_{\mathsf{idl}}$ 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, $\mathcal{A}_{\mathsf{idl}}$ 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 $\mathcal{A}_{\mathsf{idl}}$ 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 $\mathcal{A}_{\text{ms}}$ outputs a successful forgery $\sigma = (t_1, t_2, s, \alpha_1, \alpha_2)$ for the message m and the aggregate public 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,[2] which means that the centered equation above allows $\mathcal{A}_{\mathsf{idl}}$ to extract x using simple field operations. Once it extract x (the discrete log of X), $\mathcal{A}_{\mathsf{idl}}$ can trivially win the IDL game. We now express this idea in a more

[2]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 $\sigma = \sigma'$.

formal way.

Proof of Lemma 8.4.1: In Figure 8.2, we construct an adversary $\mathcal{A}_{\mathsf{idl}}$ which plays the IDL game (parameterized with the group $\mathbb{G}$ and the generator g_1) by executing A_{ms} and simulating the oracles that it has access to. Without loss of generality, we assume that all random oracle queries of $\mathcal{A}_{\mathrm{ms}}$ 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 $\mathcal{A}_{\mathsf{idl}}$ simulates perfectly the oracles that $\mathcal{A}_{\mathrm{ms}}$ has access to. The only places that bad can be set to true are the following:

- When $\mathcal{A}_{\mathsf{idl}}$ 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 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 $\mathcal{A}_{\mathrm{ms}}$ wins, then $\mathcal{A}_{\mathsf{idl}}$ wins too. If $\mathcal{A}_{\mathrm{ms}}$ 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 $\widetilde{\text{SIGNO}}$ 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 $\mathcal{A}_{\text{ms}}$ 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 $\mathsf{bad} \leftarrow \mathsf{true}$ in $\widetilde{H_2}$), $\mathcal{A}_{\mathsf{idl}}$ 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 $\mathcal{A}_{\text{ms}}$ wins then $\mathcal{A}_{\mathsf{idl}}$ wins. Furthermore, $\mathcal{A}_{\mathsf{idl}}$ makes at most $q_0 + 1$ queries to CHALLENGE. Hence,

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

which is what we wanted to prove. ∎

$\mathcal{A}_{\mathsf{idl}}^{\textsc{Challenge}}(X)$:

1 $\mathsf{bad} \leftarrow \mathsf{false}$ // Abort whenever $\mathsf{bad} \leftarrow \mathsf{true}$
2 $H_0, H_1, H_2, H_0^q, H_2^{\mathsf{dl}}, \Sigma \leftarrow$ empty dictionaries
3 $q \leftarrow 0$ // tracks $\widetilde{H_0}$ queries
4 For $i = 1, \ldots, n-1$:
5 $\quad X_i \leftarrow\!\!\$\ \mathbb{G}$
6 $X_n \leftarrow X \cdot \prod_{i=1}^{n-1} X_i^{-1}$
7 For $i = 1, \ldots, n$ // create proofs of possessions
8 $\quad c_i, z_i \leftarrow\!\!\$\ \mathbb{Z}_p$; $R_i \leftarrow g_1^{z_i} X_i^{-1}$
9 $\quad$ If $H_1(X_i, R_i)$ initialized: $\mathsf{bad} \leftarrow \mathsf{true}$
10 $\quad H_1[X_i, R_i] \leftarrow c_i$
11 $\quad \pi_i \leftarrow (c_i, z_i)$
12 $(m, (t_1, t_2, s, \alpha_1, \alpha_2)) \leftarrow \mathcal{A}_{\mathrm{ms}}^{\widetilde{\textsc{SignO}}, \widetilde{H_0}, \widetilde{H_1}, \widetilde{H_2}}((X_1, \pi_1), \ldots, (X_n, \pi_n))$
13 $c \leftarrow \widetilde{H_0}(t_1, t_2, X, m)$; $(g_2, h_1, h_2) \leftarrow \widetilde{H_2}(m)$
14 $(\omega_1, \omega_2, \omega_3) \leftarrow H_2^{\mathsf{dl}}[m]$
15 If $H_0^q[t_1, t_2, X, m]$ initialized:
16 $\quad I \leftarrow H_0^q[t_1, t_2, X, m]$
17 $\quad$ Return $(I, s + \alpha_1\omega_2 + \alpha_2\omega_3)$
18 Else:
19 $\quad (\alpha_1', \alpha_2', s') \leftarrow \Sigma[t_1, t_2, m]$
20 $\quad x \leftarrow (\alpha_1 - \alpha_1')(\alpha_2' - \alpha_2)^{-1}\omega_1^{-1}$
21 $\quad r \leftarrow\!\!\$\ \mathbb{Z}_p$; $R \leftarrow g_1^r$; $c' \leftarrow \textsc{Challenge}(R)$
22 $\quad$ Return $(q+1, r + x \cdot c')$

$\widetilde{\textsc{SignO}}(m)$:

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: $\mathsf{bad} \leftarrow \mathsf{true}$
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 $(t_1, t_2, s, \alpha_1, \alpha_2)$

$\widetilde{H_0}(t_1, t_2, X', m)$:

30 If $H_0[t_1, t_2, X', m]$ uninitialized:
31 $\quad$ If $X' = X$:
32 $\quad\quad q \leftarrow q+1$; $H_0^q[t_1, t_2, X, m] \leftarrow q$
33 $\quad\quad c \leftarrow \textsc{Challenge}(t_2)$
34 $\quad\quad H_0[t_1, t_2, X', m] \leftarrow c$
35 $\quad$ Else:
36 $\quad\quad H_0[t_1, t_2, X', m] \leftarrow\!\!\$\ \mathbb{Z}_p$
37 Return $H_0[t_1, t_2, X', m]$

$\widetilde{H_1}(X', R)$:

38 If $H_1[X', R]$ uninitialized:
39 $\quad H_1[X', R] \leftarrow\!\!\$\ \mathbb{Z}_p$
40 Return $H_1[X', R]$

$\widetilde{H_2}(m)$:

41 If $H_2[m]$ uninitialized:
42 $\quad \omega_1, \omega_2, \omega_3 \leftarrow\!\!\$\ \mathbb{Z}_p$
43 $\quad$ If $\omega_1 = 0$: $\mathsf{bad} \leftarrow \mathsf{true}$
44 $\quad h_1 \leftarrow X^{\omega_1}$; $g_2 \leftarrow g_1^{\omega_2}$; $h_2 \leftarrow g_1^{\omega_3}$
45 $\quad H_2[m] \leftarrow (g_2, h_1, h_2)$
46 $\quad H_2^{\mathsf{dl}}[m] \leftarrow (\omega_1, \omega_2, \omega_3)$
47 Return $H_2[m]$

Figure 8.2: Algorithms $\mathcal{A}_{\mathsf{idl}}$, the reduction used in the proof of Lemma 8.4.1.

BIBLIOGRAPHY

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] M. Bellare, C. Namprempre, 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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`.

[11] 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.

[12] 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.

[13] C. Decker and R. Wattenhofer. Bitcoin transaction malleability and mtgox. In M. Kutyłowski and J. Vaidya, editors, *Computer Security - ESORICS 2014*, pages 313–326, Cham, 2014. Springer International Publishing.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] K. Itakura, K; Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC research & development*, 1983.

[21] 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.

[22] 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`.

[23] 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`.

[24] 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.

[25] National Institute of Standards and Technology. Multi-Party Threshold Cryptography, 2018–Present. `https://csrc.nist.gov/Projects/threshold-cryptography`.

[26] S. Navot. Insecurity of MuSig and bellare-neven multi-signatures with delayed message selection. Cryptology ePrint Archive, Paper 2024/437, 2024.

[27] S. Navot and S. Tessaro. One-more unforgeability for multi - and threshold signatures. In K.-M. Chung and Y. Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 429–462, Singapore, 2025. Springer Nature Singapore.

[28] J. Nick. Insecure shortcuts in musig, 2019. `https://medium.com/blockstream/insecure-shortcuts-in-musig-2ad0d38a97da`.

[29] 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`.

[30] 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.

[31] 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.

[32] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. In *Journal of Cryptology*. Journal of Cryptology, May 1998.

[33] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[34] 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.

[35] 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.

[36] G. Segev and L. Shapira. An explicit high-moment forking lemma and its applications to the concrete security of multi-signatures. *IACR Communications in Cryptology*, 1(2), 2024.

[37] 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.

[38] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.4)*. The Sage Development Team, 2024. `https://www.sagemath.org`.

[39] 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.

[40] P. Wuille. Dealing with malleability. Bitcoin Improvement Proposal 62, 2014. `https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki`.

[41] 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`.