Schnorr Signatures and MuSig2 Are Jointly Secure, Even in Deterministic Wallets

Renas Bacho[1], Yanbo Chen[2], Poulami Das[3], Julian Loss[1], Tim Ruffing[4], and Benedikt Wagner[5]

[1] Ruhr University Bochum
{renas.bacho,julian.loss}@rub.de
[2] University of Ottawa
ychen918@uottawa.ca
[3] Least Authority
poulamidas22@gmail.com
[4] Blockstream Research
me@real-or-random.org
[5] Ethereum Foundation
benedikt.wagner@ethereum.org

Abstract. Modern cryptocurrency wallets use Schnorr signatures together with public, deterministic key derivation: by repeatedly rerandomizing a single master public key, an unlimited number of public keys for incoming payments can be derived without access to secret data. Moreover, some wallets support multi-signature schemes such as MuSig2 for optionally aggregating derived public keys from n distinct parties into "n-of-n" public keys, so that funds received on such aggregated keys can be spent only when authorized by all n parties. However, these advanced key-management techniques stretch the underlying cryptographic schemes beyond the guarantees provided by existing security proofs. Although deterministic Schnorr wallets and MuSig2 have each been proven secure in isolation, the provable security of their composition—despite its deployment in wallets—has not been established thus far.

The goal of this work is to narrow this gap between theory and practice. We provide the first formal security analysis of the joint use of single-signer Schnorr signatures and MuSig2, both under rerandomization and in deterministic wallets. First, we introduce the notion of a *(rerandomizable) joint-signature scheme*, in which an honest signer uses the same secret key for single-signer signatures and multi-signatures. Within this model, we prove tight security of Schnorr signatures plus MuSig2 under the algebraic one-more discrete logarithm (AOMDL) assumption in the combination of the algebraic group model (AGM) and the random oracle model (ROM). Finally, we provide a formal model of deterministic wallets with joint-signature schemes and show that Schnorr plus MuSig2 remains secure under public, deterministic key derivation.

Table of Contents

Schnorr Signatures and MuSig2 Are Jointly Secure, Even in Deterministic Wallets 1
Renas Bacho, Yanbo Chen, Poulami Das, Julian Loss, Tim Ruffing, and Benedikt Wagner
1 Introduction 3
1.1 Our Contributions 4
1.2 Related Work 4
1.3 Open Questions 5
2 Technical Overview 5
3 Preliminaries 7
3.1 Assumptions and Idealizations 7
3.2 Signature and Multi-Signature Schemes 8
4 Joint-Signature Schemes 9
5 Joint-Signatures with Rerandomizable Keys 10
5.1 Construction and Security Analysis 11
6 Proofs of Lemmas 15
6.1 Proof of Lemma 4 ($\mathbf{G}_6 \to \mathbf{G}_7$) 15
6.2 Proof of Lemma 5 (the case $b^* = 0$) 18
6.3 Proof of Lemma 6 (the case $b^* = 1$) 21
7 Application to Deterministic Wallets 28
7.1 Joint Deterministic Wallets 28
7.2 Construction and Security Analysis 31
A Postponed Definitions 34
A.1 MuSig2 Multi-Signature 34
A.2 EUF-CMA Security for Joint-Signatures 35

1 Introduction

Cryptocurrencies such as Bitcoin use a blockchain to maintain a publicly verifiable ledger of all transactions. Every transaction associates a set of funds to their new owner represented by a public key. This new owner, i.e., the recipient of the transaction, can authorize further spending of the funds by signing another transaction with the corresponding secret key.

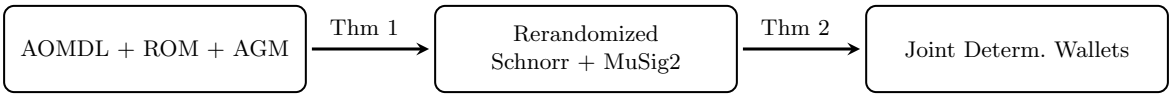


Fig. 1. Summary of our contributions: from assumptions to rerandomized joint signatures and wallets.

If a user reuses the same public key for incoming transactions, an observer of the public blockchain can trivially link all these transactions. This linkage exposes sensitive information about the user's assets and payment history, and may make their wallet an attractive target for theft. Consequently, public key reuse has already been discouraged since Bitcoin's inception [Nak08]. A naive way to avoid trivial linkability is to generate a fresh, independent key pair for every incoming transaction. However, storing and backing up a large number of unrelated secret keys is impractical.

Instead, essentially all contemporary Bitcoin wallets adopt a more compact and flexible approach through *Bitcoin Improvement Proposal 32* (BIP32) [Wui12], a standard for *deterministic wallets*. A deterministic wallet uses a single master key pair $(\mathsf{msk}, \mathsf{mpk})$ together with a bitstring ch called the *chaincode*, from which an unbounded sequence of pseudorandom key pairs (indexed by some identifier ID) can be deterministically derived. Crucially, public keys can be derived without knowledge of the master secret key msk. This makes it possible to store msk in a trusted and offline component of the wallet called the *cold wallet*, e.g., a secure hardware token.

The corresponding untrusted and online component called the *hot wallet*, e.g., a smartphone app connected to the Bitcoin network, has access only to mpk and ch, from which it can derive an unlimited number of single-use public keys for receiving and managing incoming transactions. All derived public keys remain unlinkable to any observer not knowing ch. Moreover, since all keys can be rederived from $(\mathsf{msk}, \mathsf{mpk})$ and ch, and these values can be backed up once at wallet creation time, users are freed from the need to back up their wallet again each time a new secret key is generated.

At the cryptographic level, BIP32 wallets derive new key pairs by rerandomizing—also known as *tweaking* among practitioners— the master key pair $(\mathsf{msk}, \mathsf{mpk})$, using "randomness" derived deterministically via a hash function from ch, mpk, and ID. This introduces an important subtlety: all derived signing keys are *correlated* since they originate from the same master secret key msk and the same chaincode ch, where the latter is assumed to be known to the adversary due to its presence in the untrusted hot wallet. Despite this subtlety, prior work [DFL19, DEF+21] has shown that it suffices for the underlying digital signature scheme to remain unforgeable under key rerandomization to securely instantiate BIP32 wallets.

In addition to the widespread deployment of BIP32 in wallets, Bitcoin's signature layer itself has evolved. While Bitcoin traditionally relied on ECDSA, the *Taproot* upgrade in November 2021 [WNR20, WNT20a, WNT20b] added support for authorizing transactions using Schnorr signatures (standardized in BIP340 [WNR20]). The main benefit of Schnorr signatures is that they allow for much simpler and much more efficient constructions of advanced signing protocols such as "n-of-n" *multi-signatures*, which make it possible for n users to jointly own a set of funds and can spend it only if they all agree.

The two-round scheme MuSig2 [NRS21] has emerged as the state-of-the-art solution for Schnorr multi-signatures in the Bitcoin ecosystem. Signatures produced by MuSig2 are fully compatible with the ordinary Schnorr verification algorithm as supported by Bitcoin. That is, they look as if a single signer had produced them, and they are valid under a single *aggregated public key* computed from the n individual public keys of the involved signers. MuSig2 has consequently been standardized in BIP327 [NRJ22] and deployed in popular Bitcoin wallets [IS24, Bla23] including the *Bitcoin Core* wallet [Cho24, Nic24], which evolved from the original code by Nakamoto and is widely considered the reference implementation of the Bitcoin protocol.

Despite the individual maturity of BIP32 and MuSig2, the security of their composition as deployed in wallets is not well understood. In particular, existing analyses of MuSig2 do not account for signing with rerandomized keys as used in BIP32-style wallets. The real-world relevance of this question is emphasized by a concrete vulnerability in a draft version of BIP327 [TR22, Nic22a]. This vulnerability

emerged precisely from the combination of MuSig2 with rerandomization of individual signers' key pairs. In more detail, if the adversary can choose the randomness used for rerandomization adaptively after the first round of the signing protocol, then it can mount a polynomial-time attack by opening many sessions concurrently [Nic22a]. While the draft did not explicitly accommodate the use of rerandomized signing keys, nothing prevents an application from providing a signing key which it has obtained via rerandomization, and this is a realistic concern given the prevalence of BIP32. As a result, the BIP327 draft had to be amended, and the final version mandates implementations to take as input the final rerandomized signing key already before the first round of a signing session [Nic22b].

Moreover, wallet implementations support mixing the interactive MuSig2 signing protocol with the non-interactive single-signer Schnorr signing algorithm within the same user's wallet, i.e., implementations will pick one or the other spontaneously depending on whether the task is to generate a multi-signature or an ordinary, single-signer Schnorr signature. Yet, even in the degenerate case of a multi-signature involving only a single signer ($n = 1$), MuSig2 internally shifts the signer's key pair $(\mathsf{sk}, \mathsf{pk})$ by applying a hash-derived scalar $a \neq 1$, so that a MuSig2 signature for the singleton signer set $\mathcal{P} = \{\mathsf{pk}\}$ (with $n = |\mathcal{P}| = 1$) will be valid under a shifted aggregated public key pk^a rather than the original public key pk. As a result, security of MuSig2 does not trivially imply security when key pairs are also used for standard Schnorr signing.

These observations motivate the following central question:

Does MuSig2 remain unforgeable when (1) jointly used with ordinary Schnorr signatures and, simultaneously, (2) combined with BIP32-style deterministic key derivation?

1.1 Our Contributions

In this work, we resolve the above question in the positive by providing the first formal security analysis of the MuSig2 Schnorr multi-signature scheme in a setting that accounts for deterministic key derivation and interoperability with single-signer Schnorr signatures. Our contributions are summarized as follows (see also Figure 1):

- We introduce and formalize the notion of a *joint-signature scheme* (with and without rerandomizable keys). This abstraction unifies a standard digital signature scheme and any of its multi-signature variants within a single framework, in which users employ the same long-term keys for both single-party and multi-party signing.
- We prove *tight security* of the rerandomizable joint-signature construction $\mathsf{Schnorr} + \mathsf{MuSig2}$ in the algebraic group model (AGM) and the random oracle model (ROM), under the hardness of the *algebraic one-more discrete logarithm (AOMDL)* problem. Our proof relies on the same underlying assumptions as the original security analysis of plain $\mathsf{MuSig2}$, while strictly strengthening it to cover rerandomized keys and joint use with single-signer Schnorr signatures.
- We present a model of *deterministic wallets for joint-signature schemes* and in this model, we consider the concrete construction of $\mathsf{Schnorr} + \mathsf{MuSig2}$. This allows us to rigorously show that deterministic wallets can be securely combined with Schnorr signatures and MuSig2 simultaneously.

1.2 Related Work

In this section, we provide an overview of related work and explain how it differs from our work.

Schnorr Multi-Signatures. A series of works have proposed Schnorr multi-signature schemes [NKDM03, MPSW19, NRSW20, AB21]. MuSig2 as well as the essentially equivalent DWMS scheme [AB21] are the successors of the original three-round MuSig scheme by Maxwell et al. [MPSW19]. The schemes by Garillot et al. [GKMN21] (restricted to $n = 2$) and MuSig-DN by Nick et al. [NRSW20] feature deterministic signing protocols to avoid leaking secret keys through an accidental reuse of randomness. This requires expensive zero-knowledge proofs and multi-party computation, respectively, and it is orthogonal to the setting considered in this work: these schemes derive the *signature* deterministically whereas deterministic wallets derive the *keys* deterministically.

Rerandomizable Keys and Deterministic Wallets. Deterministic wallets have been considered in many different settings [GS15, DFL19, AADE$^+$20, LFA20, GS22, DEF$^+$25]. Foundational to our work, Das et al. [DFL19] show that deterministic wallets can be constructed generically from signature schemes with rerandomizable keys as first introduced by Fleischhacker et al. [FKM$^+$16]. Rerandomizable signature schemes continue to be a useful building block in constructions of deterministic wallets [DFL19, DEF$^+$21, ER22].

Other Schnorr-based Signatures with Rerandomizable Keys. Whereas we focus on key rerandomization in the context of Schnorr multi-signatures, other recent works have considered key randomization in the context of multi-party variants of Schnorr signatures other than multi-signatures: Gouvea and Komlo [GK24] prove that FROST, a two-round threshold signature scheme, which can be viewed as a threshold analogue of MuSig2, is secure with rerandomizable keys. Similarly, Nick et al. [NRS26] present a two-round aggregate signature scheme with rerandomizable keys that is also similar in spirit to MuSig2. Both of these works provide security proofs based on the AOMDL assumption in the ROM.

1.3 Open Questions

Our work makes significant progress towards closing the gap between the provable security of Schnorr signatures and MuSig2 and their real-world use in Bitcoin wallets. However, it does not cover the following use cases and technical optimizations, a formal treatment of which we leave to future work.

- BIP32 wallets support a *hierarchy* of key derivations, i.e., a derived key pair can effectively act as master key pair, from which further key pairs can be derived. Our model covers only one level of derivation.
- Real-world wallets rerandomize not only individual signers' key pairs, but also the MuSig2 *aggregated public key*. This is necessary for full compatibility with Bitcoin's Taproot upgrade [WNT20a].[6] Moreover, rerandomization of the aggregated public key can be used to derive further public keys from the aggregated public key (as touched upon in the previous item). Accordingly, BIP327 describes an extension to the MuSig2 signing protocol that explicitly supports signing for rerandomized public keys [NRJ22]. In our model, the individual signers' public keys provided as input to the key aggregation algorithm can be rerandomized, but the resulting aggregated public key cannot.
- BIP327 relies on a slightly more efficient but also more complex key aggregation method [NRS21, Appendix A] that saves a single exponentiation over the original key aggregation method. Our work covers only the original method.

2 Technical Overview

In this section, we recall the existing cryptographic schemes underlying our work, and we provide an overview on how we extend existing formalizations. Moreover, we explain the attack hinted at in Section 1 in more detail.

Schnorr and MuSig2 Signatures. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p with generator g. A user holds a key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_p$ and $\mathsf{pk} := g^{\mathsf{sk}}$. We first recall the standard Schnorr signature scheme. To sign a message m, the signer samples $r \leftarrow_\$ \mathbb{Z}_p$ and computes

$$R := g^r, \quad c := \mathsf{H}_c(\mathsf{pk}, R, m), \quad s := c \cdot \mathsf{sk} + r.$$

The signature is $\sigma := (R, s)$. Here, $\mathsf{H}_c : \{0,1\}^* \to \mathbb{Z}_p$ is modeled as a random oracle. Verification of (R, s) proceeds by checking whether $g^s = R \cdot \mathsf{pk}^c$.

We now switch to the multi-party setting in which n users with public keys $\mathcal{P} := \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ aim to jointly create a Schnorr multi-signature σ on message m. The MuSig2 scheme [NRS21] makes use of two additional hash functions $\mathsf{H}_a, \mathsf{H}_b : \{0,1\}^* \to \mathbb{Z}_p$ (modeled as random oracles). We describe the protocol from the perspective of user 1 with key pair $(\mathsf{sk}_1, \mathsf{pk}_1)$. First, it computes its key aggregation coefficient $a_1 := \mathsf{H}_a(\mathcal{P}, \mathsf{pk}_1) \in \mathbb{Z}_p$.

In the first round, the signer samples nonces $r_{1,j} \leftarrow_\$ \mathbb{Z}_p$ and computes $R_{1,j} := g^{r_{1,j}}$ for $j \in [2]$.[7] After receiving all other nonces $\{(R_{i,1}, R_{i,2})\}_{i \in [n]}$, the parties compute the aggregated nonces

$$R_j := \prod_{i \in [n]} R_{i,j} \quad \text{for } j \in [2].$$

In the second round, once the message m is fixed, the users compute the aggregated public key

$$\tilde{\mathsf{pk}} := \prod_{i \in [n]} \mathsf{pk}_i^{a_i}, \quad \text{where } a_i := \mathsf{H}_a(\mathcal{P}, \mathsf{pk}_i),$$

[6] We refer the interested reader to Ruffing [Ruf25] for a detailed explanation of Taproot aimed at a cryptographic audience.

[7] We describe the protocol with $\nu = 2$ nonces as standardized in BIP327 [NRJ22].

and the scalar $b := \mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$. Then, the users set $R := R_1 R_2^b$ and compute

$$c := \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m), \qquad s_1 := ca_1\mathsf{sk}_1 + (r_{1,1} + br_{1,2}).$$

The user's partial signature is s_1, and the final signature is $\sigma := (R, s)$, where $s := \sum_{i\in[n]} s_i$.

One can see that this is an ordinary Schnorr signature for the aggregated public key $\prod_{i\in[n]} \mathsf{pk}_i^{a_i}$. Intuitively, each user i signs with the *shifted* key pair $(a_i\mathsf{sk}_i, \mathsf{pk}_i^{a_i})$ instead of its registered pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. But at the same time, it uses $(\mathsf{sk}_i, \mathsf{pk}_i)$ for ordinary (single-signer) Schnorr signing. Thus, when users employ their keys on Bitcoin to produce both Schnorr and MuSig2 signatures, it becomes unclear what the stand-alone security proof of MuSig2 guarantees in this setting. This issue is further complicated when users derive their individual keys (which are provided as input to key aggregation) deterministically from a master key. In more detail, deterministic wallets use *additive* key rerandomization, i.e., a master key pair $(\mathsf{msk}_i, \mathsf{mpk}_i)$ of user i is rerandomized to $(\mathsf{sk}_i, \mathsf{pk}_i) = (\mathsf{msk}_i + \rho, \mathsf{mpk}_i \cdot g^\rho)$ given some randomness ρ.[8]

Towards an Improved Model. To address these issues, we introduce the model of a *joint-signature scheme with rerandomizable keys*. A joint-signature (JS) scheme combines a standard digital signature (DS) scheme with one of its multi-signature (MS) variants, sharing the same key generation and verification algorithms. Formally, a JS scheme consists of algorithms

$$\mathsf{JS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAggr}, \mathsf{SSign}, \mathsf{MSign}, \mathsf{Verify}),$$

where $\mathsf{DS} := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SSign}, \mathsf{Verify})$ denotes a standard digital signature scheme and $\mathsf{MS} := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAggr}, \mathsf{MSign}, \mathsf{Verify})$ denotes a multi-signature scheme. Moreover, the scheme supports *rerandomizable keys* if it is augmented with algorithms $(\mathsf{RandSK}, \mathsf{RandPK})$ over randomness space $\mathcal{R}$, such that rerandomized keys

$$\hat{\mathsf{sk}} \leftarrow \mathsf{RandSK}(\mathsf{sk}, \rho), \qquad \hat{\mathsf{pk}} \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho)$$

for random $\rho \leftarrow_\$ \mathcal{R}$ follow the same distribution as freshly generated keys $(\bar{\mathsf{sk}}, \bar{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}$.

In the corresponding security game, the adversary $\mathcal{A}$ has access to signing oracles for both the single-signer and multi-signer functionalities. That is, it may request standard signatures from an honest signer with key pair $(\mathsf{sk}_1, \mathsf{pk}_1)$, while simultaneously interacting with multi-signature oracles. For each oracle query, the adversary may also specify arbitrary randomness ρ under which the honest signer must sign (i.e., using the corresponding rerandomized key). This models that honest users use their (rerandomized) keys for both Schnorr and MuSig2 signing. It wins the game if it outputs a forgery $(\mathcal{P}^*, m^*, \sigma^*, \rho^*, b^*)$, where σ^* verifies as a valid signature on message m^* under the public key set $\mathcal{P}^*$, which includes the rerandomized key $\hat{\mathsf{pk}} \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho^*)$. The bit b^* indicates whether the forgery is to be verified as a multi-signature or as a single-signer signature (note that the difference lies solely in how the aggregated public key is computed during verification). Moreover, $(\mathcal{P}^*, m^*)$ must not have been used in any previously completed signing session.

A Subtlety and an ROS-type Attack. Our objective is to prove the security of $\mathsf{Schnorr} + \mathsf{MuSig2}$ under rerandomized keys. A natural attempt to model security would be to allow the adversary to specify the randomness ρ only in the second round of the signing protocol, because it is not used before. However, as we have already hinted at in Section 1, there is an attack in this model, which has been discovered in the context of BIP327 [TR22, Nic22a]. Therefore, our model forces the adversary to specify ρ already in the first round, when initiating a signing session, (Analogously, the final version of BIP327 forces the signer to provide the final rerandomized signing key already in the first round.) In this model, our proof goes through. This subtlety showcases the need for provable security.

Let us now explain this subtlety in more detail. We first consider security without rerandomization. Recall that the adversary can interact with the honest signer in multi-signature sessions. In the first round of a multi-signature session, the adversary receives $R_{1,1} := g^{r_{1,1}}$ and $R_{1,2} := g^{r_{1,2}}$ from the honest signer. In the second round, the adversary receives $s_1 := ca_1\mathsf{sk}_1 + (r_{1,1} + br_{1,2})$. Here, we have $a_1 := \mathsf{H}_a(\mathcal{P}, \mathsf{pk})$, $b := \mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$, and $c := \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$, where $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ and $R := R_1 R_2^b$. These hash values depend on the inputs $\mathcal{P}$, m, (R_1, R_2) for the second round chosen by the adversary.

We briefly review the conditions under which an ROS (**R**andom inhomogeneities in an **O**verdetermined **S**olvable system of linear equations) attack [Wag02, BLL+22] is applicable. The ROS attack applies if the adversary could efficiently find two possible inputs for the second round that yield the same b but different values of $d := ca_1$. In the attack, the adversary concurrently opens $\log p$ signing sessions, finds

[8] The shifting of a key pair that occurs as part of key aggregation is also a form of rerandomization, namely multiplicative rerandomization. For simplicity however, we reserve the term "rerandomization" in this work for the additive kind.

two potential inputs with the above property for each session, and strategically computes a forged nonce R^*. Given $c^* := \mathsf{H}_c(\mathsf{pk}, R^*, m^*)$ with an arbitrary m^*, it can select one of the two potential inputs to complete each session, such that it can combine the outputs of all the sessions to obtain a forged signature (R^*, s^*) under pk.

MuSig2 resists this attack in the fixed-key model because all the second-round inputs to the signing oracle are effectively hashed by $\mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$. To find two possible inputs that yield the same b, the adversary needs to find a collision in H_b or in $\mathsf{KeyAggr}$.

Now let us consider security with rerandomizable keys, where the honest signer uses key pair $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$ and $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho)$. Suppose that the adversary can choose the randomness ρ for rerandomizing the key in the second round. Crucially, ρ is also an input of the second signing round, but it is not an input to H_b. Moreover, fixing other inputs $\mathcal{P}$, m, and (R_1, R_2), different values of $\rho \neq \rho'$ of the randomness are sufficient to yield different values $a_1 \neq a_1'$. The adversary can let $\mathcal{P} := \{\mathsf{RandPK}(\mathsf{pk}, \rho), \mathsf{RandPK}(\mathsf{pk}, \rho')\} = \{\hat{\mathsf{pk}}, \hat{\mathsf{pk}}'\}$. If it chooses ρ as the randomness, then the honest signer signs under $\hat{\mathsf{pk}}$ with $a_1 := \mathsf{H}_a(\mathcal{P}, \hat{\mathsf{pk}})$; if it chooses ρ', then the honest signer signs under $\hat{\mathsf{pk}}'$ with $a_1' := \mathsf{H}_a(\mathcal{P}, \hat{\mathsf{pk}}')$. Meanwhile, b and c are fixed, since $\mathcal{P}$, m, and (R_1, R_2) are fixed. Therefore, the adversary can choose between two second-round inputs that yield the same b but different $d := ca_1$, which enables the ROS attack.

3 Preliminaries

We denote the security parameter by $\lambda \in \mathbb{N}$ and assume that all algorithms get 1^λ implicitly as input. For a given finite set S, we assume a canonical ordering (e.g., lexicographically) and write $\langle S \rangle$ to denote a unique encoding of this set. We use standard cryptographic terminology such as negligible, overwhelming, and probabilistic polynomial-time (PPT). We write $x \leftarrow_\$ S$ to denote that x is sampled uniformly at random from a set S. We write $x \leftarrow \mathcal{D}$ to denote that x is sampled according to a distribution $\mathcal{D}$. We write $y \leftarrow \mathsf{A}(x)$ to denote that the (probabilistic) algorithm A is run with uniformly sampled random coins on input x and assign its output to y. If A is deterministic, we instead write $y := \mathsf{A}(x)$. We write $\mathbf{T}(\mathsf{A})$ for the running time of A. All our algorithms are probabilistic, unless stated otherwise. For a positive integer $k \in \mathbb{N}$, we define $[k] := \{1, \ldots, k\}$ and $[\![k]\!] := \{0, \ldots, k\}$.

3.1 Assumptions and Idealizations

We rely on the following assumptions and idealizations.

Idealized Models. We work in the *random oracle model (ROM)* [BR93], where a hash function H is treated as an idealized random function. The security of all existing Schnorr-based signature schemes relies on this model. Furthermore, we work in the *algebraic group model (AGM)* [FKL18], where the adversary is assumed to be algebraic. An algorithm $\mathcal{A}$ is algebraic (over a group $\mathbb{G}$) if for every group element $\zeta \in \mathbb{G}$ that it outputs, it also outputs a vector $\mathbf{z} = (z_1, \ldots, z_k)$ of integers such that $\zeta = \prod_{i \in [k]} g_i^{z_i}$, where $(g_1, \ldots, g_k)$ denotes the list of group elements it has observed so far.

AOMDL Assumption. We assume the hardness of the algebraic one-more discrete logarithm (AOMDL) problem [NRS21]. In the respective security game, the adversary receives a group description $(\mathbb{G}, p, g)$. Its goal is to output the discrete logarithms $x_1, \ldots, x_\ell \in \mathbb{Z}_p$ (to base g) of ℓ challenge elements $X_1, \ldots, X_\ell \in \mathbb{G}$, while making at most $\ell - 1$ queries to a discrete logarithm oracle $\texttt{DLOG}$. A crucial restriction is that every $\texttt{DLOG}$ query must be made algebraically, i.e., the adversary must provide an algebraic representation of the queried element $X \in \mathbb{G}$ in terms of the group elements it has previously obtained (i.e., $g, X_1, \ldots, X_\ell$). We formally define this as follows.

Definition 1 (AOMDL Problem). *Let* GrGen *be a* group generation algorithm *that, on input* 1^λ*, outputs the description of a (multiplicative) group* $\mathbb{G}$ *of prime order* p *with generator* g*. Consider the security game* ℓ-$\mathsf{AOMDL}^{\mathcal{A}}_{\mathsf{GrGen}}$ *defined in Figure 2. We say that the* ℓ*-AOMDL problem with* $\ell \in \mathbb{N}$ *is hard for* GrGen *if for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{A},\mathsf{GrGen}}(\lambda) := \Pr\left[\ell\text{-}\mathsf{AOMDL}^{\mathcal{A}}_{\mathsf{GrGen}}(\lambda) \Rightarrow 1\right].$$

Game ℓ-AOMDL$^{\mathcal{A}}_{\mathsf{GrGen}}(\lambda)$	**Oracle** $\mathtt{DLOG}\left((\beta_i)_{i\in[\ell]}\right)$
00 $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$	06 **if** $\exists i \in [\ell] : \beta_i \notin \mathbb{Z}_p$: **return** $\perp$
01 $\mathsf{cnt} := 0,\ \mathsf{par} := (\mathbb{G}, p, g)$	07 $\mathsf{cnt} := \mathsf{cnt} + 1$
02 **for** $i \in [\ell] : x_i \leftarrow_\$ \mathbb{Z}_p,\ X_i := g^{x_i}$	08 **return** $\sum_{i\in[\ell]} \beta_i x_i$
03 $(x'_1, \ldots, x'_\ell) \leftarrow \mathcal{A}^{\mathtt{DLOG}}(\mathsf{par}, X_1, \ldots, X_\ell)$	
04 **if** $(\forall i \in [\ell] : g^{x'_i} = X_i) \wedge (\mathsf{cnt} < \ell)$: **return** 1	
05 **return** 0	

Fig. 2. The game **ℓ-AOMDL** for group generation algorithm GrGen and adversary $\mathcal{A}$.

3.2 Signature and Multi-Signature Schemes

We briefly recall the syntax for a standard digital signature scheme and for multi-signatures. For multi-signatures, we focus on two-round protocols with key aggregation, where the first signing round is independent of the message to be signed, adopting the definitions of Nick et al. [NRS21]. For security, we require the standard notion of existential unforgeability under chosen-message attacks (EUF-CMA). As we do not formally analyze such schemes in isolation, we omit a formal security definition and instead refer to [Kat10, NRS21]. We will give a security definition for the joint setting later.

Definition 2 (Digital Signature Scheme). *A digital signature scheme is a tuple of PPT algorithms* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SSign}, \mathsf{Verify})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$*: The* setup algorithm *takes as input the security parameter* 1^λ *and outputs global system parameters* par*. We assume that all algorithms related to* DS *implicitly take* par *as input.*
- $\mathsf{KeyGen}(\mathsf{par}) \to (\mathsf{sk}, \mathsf{pk})$*: The* key generation algorithm *takes as input the system parameters* par*, and outputs a tuple consisting of a secret key* sk *and a public key* pk*.*
- $\mathsf{SSign}(\mathsf{sk}, m) \to \sigma$*: The* signing algorithm *takes as input a secret key* sk *and a message* m*, and outputs a signature* σ*.*
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 0/1$*: The deterministic* verification algorithm *takes as input a public key* pk*, a message* m*, a signature* σ*, and outputs a bit* 0 *(reject) or* 1 *(accept).*

We require DS *to be correct: for all* $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$*, all* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(\mathsf{par})$*, and all* $m \in \{0,1\}^*$*, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1 \mid \sigma \leftarrow \mathsf{SSign}(\mathsf{sk}, m)\right] = 1.$$

Definition 3 (Multi-Signature Scheme). *A multi-signature scheme is a tuple of PPT algorithms* $\mathsf{MS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAggr}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$*: The* setup algorithm *takes as input the security parameter* 1^λ *and outputs global system parameters* par*. We assume that* par *implicitly defines sets of public keys, secret keys, messages and signatures, and all algorithms related to* MS *implicitly take* par *as input.*
- $\mathsf{KeyGen}(\mathsf{par}) \to (\mathsf{sk}, \mathsf{pk})$*: The* key generation algorithm *takes as input the system parameters* par*, and outputs a pair consisting of a secret key* sk *and a public key* pk*.*
- $\mathsf{KeyAggr}(\mathcal{P}) \to \tilde{\mathsf{pk}}$*: The deterministic* key aggregation algorithm *takes as input a multiset of public keys* $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ *and outputs an aggregated public key* $\tilde{\mathsf{pk}}$*.*
- $\mathsf{Sign} = (\mathsf{Sign}_0, \mathsf{SignAggr}_0, \mathsf{Sign}_1, \mathsf{SignAggr}_1, \mathsf{Sign}_2)$*: The interactive* signing protocol *is formally given by five algorithms:*
  - $\mathsf{Sign}_0() \to (\mathsf{pm}_1, \mathsf{St}_1)$*: This algorithm takes no explicit input, and outputs a first-round protocol message* pm_1 *and a first-round private state* St_1*.*
  - $\mathsf{SignAggr}_0(\mathcal{M}_1) \to \tilde{\mathsf{pm}}_1$*: This algorithm is deterministic and takes as input a tuple of first-round protocol messages* $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,n})$*, and outputs an aggregated first-round protocol message* $\tilde{\mathsf{pm}}_1$*.*
  - $\mathsf{Sign}_1(\mathcal{P}, \mathsf{sk}, m, \mathsf{St}_1, \tilde{\mathsf{pm}}_1) \to (\mathsf{pm}_2, \mathsf{St}_2)$*: This algorithm takes as input a multiset of public keys* $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$*, a secret key* sk*, a message* m*, a first-round private state* St_1*, an aggregated first-round protocol message* $\tilde{\mathsf{pm}}_1$*, and outputs a second-round protocol message* pm_2 *and a second-round private state* St_2*.*
  - $\mathsf{SignAggr}_1(\mathcal{M}_2) \to \tilde{\mathsf{pm}}_2$*: This algorithm is deterministic and takes as input a tuple of second-round protocol messages* $\mathcal{M}_2 = (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,n})$*, and outputs an aggregated second-round protocol message* $\tilde{\mathsf{pm}}_2$*.*

- $\mathsf{Sign}_2(\mathsf{St}_2, \tilde{\mathsf{pm}}_2) \to \sigma$*: This algorithm takes as input a second-round private state* St_2 *and an aggregated second-round protocol message* $\tilde{\mathsf{pm}}_2$*, and outputs a signature* σ*.*

- $\mathsf{Verify}(\tilde{\mathsf{pk}}, m, \sigma) \to 0/1$*: The deterministic* verification algorithm *takes as input an aggregated public key* $\tilde{\mathsf{pk}}$*, a message* m*, a signature* σ*, and outputs a bit* 0 *(reject) or* 1 *(accept).*

We require MS *to be correct: for all* $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$*, all* $n = poly(\lambda)$*, all* $(\mathsf{sk}_i, \mathsf{pk}_i) \in \mathsf{KeyGen}(\mathsf{par})$ *for* $i \in [n]$*, and all* $m \in \{0,1\}^*$*, we have*

$$\Pr\left[\mathsf{Verify}(\tilde{\mathsf{pk}}, m, \sigma) = 1 \;\middle|\; \begin{array}{c} \tilde{\mathsf{pk}} \leftarrow \mathsf{KeyAggr}(\mathcal{P}), \\ \sigma \leftarrow \mathsf{MS.Exec}(\mathcal{P}, \mathcal{S}, m) \end{array}\right] = 1,$$

where $\mathcal{P} := \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$*,* $\mathcal{S} := \{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$*, and algorithm* $\mathsf{MS.Exec}(\mathcal{P}, \mathcal{S}, m)$ *models an honest execution of the signing protocol* Sign *for* MS *as defined in Figure 3.*

Algorithm $\mathsf{MS.Exec}(\mathcal{P}, \mathcal{S}, m)$

00 **let** $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, $\mathcal{S} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$
01 **for** $i \in [n] : (\mathsf{pm}_{1,i}, \mathsf{St}_{1,i}) \leftarrow \mathsf{Sign}_0()$
02 **let** $\mathcal{M}_1 := (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,n})$
03 $\tilde{\mathsf{pm}}_1 := \mathsf{SignAggr}_0(\mathcal{M}_1)$
04 **for** $i \in [n] : (\mathsf{pm}_{2,i}, \mathsf{St}_{2,i}) \leftarrow \mathsf{Sign}_1(\mathcal{P}, \mathsf{sk}_i, m, \mathsf{St}_{1,i}, \tilde{\mathsf{pm}}_1)$
05 **let** $\mathcal{M}_2 := (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,n})$
06 $\tilde{\mathsf{pm}}_2 := \mathsf{SignAggr}_1(\mathcal{M}_2)$
07 **for** $i \in [n] : \sigma_i \leftarrow \mathsf{Sign}_2(\mathsf{St}_{2,i}, \tilde{\mathsf{pm}}_2)$
08 **if** $\exists i \neq j \in [n]$ s.t $\sigma_i \neq \sigma_j$: **return** $\perp$
09 **return** $\sigma := \sigma_1$

Fig. 3. Algorithm MS.Exec for a multi-signature scheme MS. This models an honest execution of the signing protocol Sign for public key multiset $\mathcal{P}$, secret keys $\mathcal{S}$, and message m.

Schnorr and MuSig2. Let GrGen be a *group generation algorithm* that, on input 1^λ, outputs the description of a (multiplicative) group $\mathbb{G}$ of prime order p with generator g. Let $\nu \in \mathbb{N}$ be a positive integer. We present the description of the Schnorr and MuSig2 schemes formally in Figure 4. In Appendix A.1, we additionally provide a verbal description of MuSig2.

4 Joint-Signature Schemes

In this section, we introduce our model of *joint-signature schemes*. In a joint-signature scheme, parties can use their keys to produce either a standard signature or a multi-signature, such that both types of signatures can be verified using the same verification algorithm. A natural class of joint-signature schemes arises from combining a digital signature scheme DS with any of its corresponding multi-signature variants MS – examples include DS = Schnorr and MS = MuSig2 (see Figure 4 for the pseudocode description of this joint-signature scheme), or DS = BLS [BLS01] and MS = Multi-BLS [Bol03].

Definition 4 (Joint-Signature Scheme). *We say that* JS = (Setup, KeyGen, KeyAggr, SSign, MSign, Verify) *is a* joint-signature scheme*, where the tuple* (Setup, KeyGen, SSign, Verify) *constitutes a standard digital signature scheme, whereas the tuple* (Setup, KeyGen, KeyAggr, MSign, Verify) *defines a multi-signature scheme.*

Henceforth, we will sometimes write JS = DS + MS to denote a joint-signature scheme with underlying digital signature scheme DS and multi-signature scheme MS.

Security. We model the security of a joint-signature scheme as follows. The adversary is given a public key pk, and it has access to two types of signing oracles: one for standard digital signatures and another for multi-signatures. When querying the oracle SSIGN, the adversary receives a standard signature under the key pk. For multi-signatures, the adversary controls $n-1$ signers, while pk corresponds to the only honest signer. To obtain a multi-signature, the adversary may query the interactive oracle MSIGN, which is decomposed into three oracles ($\mathtt{SIGN}_0, \mathtt{SIGN}_1, \mathtt{SIGN}_2$) that can be invoked separately. Finally, the adversary outputs a forgery with respect to either a standard signature or a multi-signature.

We call this notion EUF-CMA security and give a formal definition of it in Appendix A.2, as it is implied by EUF-CMA-RK, which we define in the next section and show for Schnorr + MuSig2.

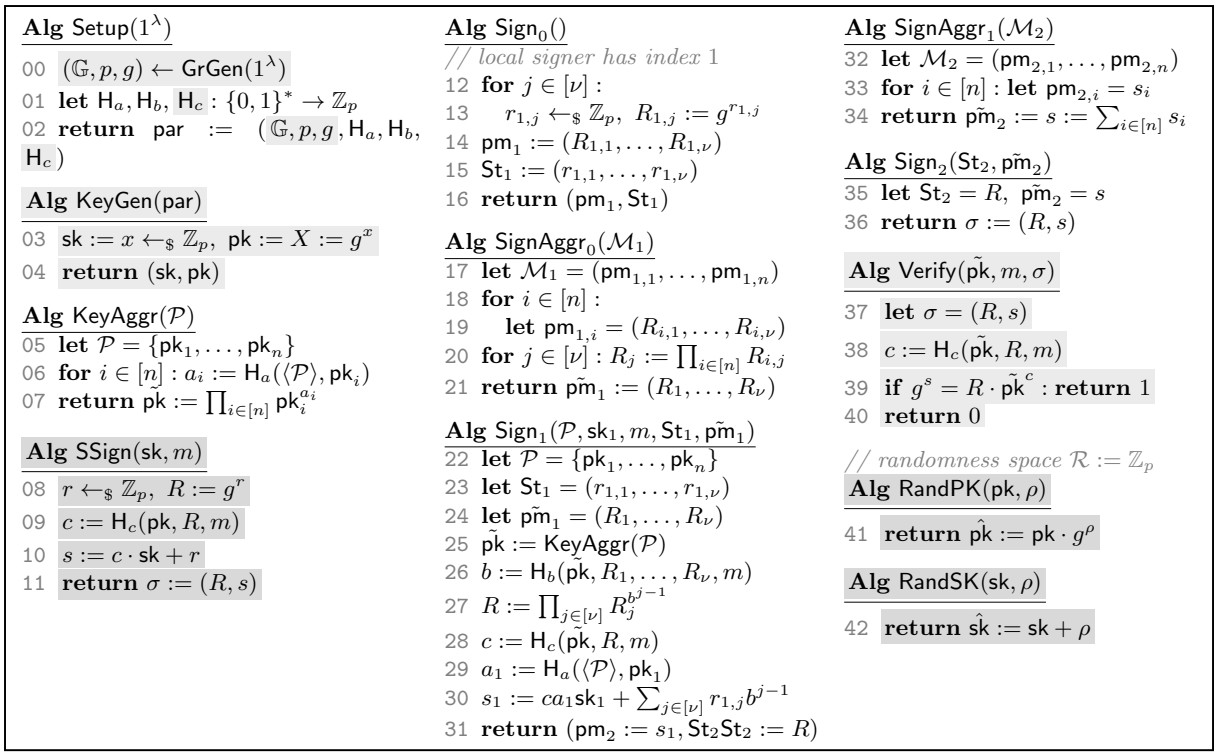


Fig. 4. The algorithms of the joint-signature scheme $\mathsf{JS} = \mathsf{Schnorr} + \mathsf{MuSig2}$. Public parameters par output by Setup are implicitly given as input to all other algorithms. We highlight common algorithms for Schnorr and MuSig2 in lightgray boxes, and the differences to MuSig2 in gray boxes (i.e., the Schnorr signing algorithm and the rerandomizable key algorithms).

5 Joint-Signatures with Rerandomizable Keys

In this section, we introduce our model of *joint-signatures with rerandomizable keys*. We obtain a joint-signature scheme with rerandomizable keys by simply augmenting the joint-signature scheme with key rerandomization algorithms. This is essentially an adaptation of the definition of prior works [DFL19, NRS26] to our setting of joint-signature schemes.

Definition 5 (Joint-Signature Scheme with Rerandomizable Keys). *Let* $\mathsf{JS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAggr}, \mathsf{SSign}, \mathsf{MSign}, \mathsf{Verify})$ *be a* joint-signature scheme *as specified in Definition 4. Further, we assume that the public parameters* par *define a randomness space* $\mathcal{R} := \mathcal{R}(\mathsf{par})$*. Then, we say that* JS *has* rerandomizable keys *if it is augmented with a pair of PPT algorithms* $(\mathsf{RandSK}, \mathsf{RandPK})$ *with the following syntax and security properties:*

- $\mathsf{RandSK}(\mathsf{sk}, \rho) \to \hat{\mathsf{sk}}$*: The deterministic* secret key rerandomization algorithm *takes as input the secret key* sk *and some randomness* $\rho \in \mathcal{R}$*, and outputs a rerandomized secret key* $\hat{\mathsf{sk}}$*.*
- $\mathsf{RandPK}(\mathsf{pk}, \rho) \to \hat{\mathsf{pk}}$*: The deterministic* public key rerandomization algorithm *takes as input the public key* pk *and some randomness* $\rho \in \mathcal{R}$*, and outputs a rerandomized public key* $\hat{\mathsf{pk}}$*.*

We require (perfect) rerandomizability of keys*: for all* $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$*, all* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(\mathsf{par})$*, and all* $\rho \in \mathcal{R}$*, the following two distributions are identical:*

$$\{(\hat{\mathsf{sk}}, \hat{\mathsf{pk}}) \mid \hat{\mathsf{sk}} \leftarrow \mathsf{RandSK}(\mathsf{sk}, \rho),\ \hat{\mathsf{pk}} \leftarrow \mathsf{RandPK}(\mathsf{pk}, \rho)\} \quad \textit{and} \quad \{(\bar{\mathsf{sk}}, \bar{\mathsf{pk}}) \mid (\bar{\mathsf{sk}}, \bar{\mathsf{pk}}) \leftarrow_\$ \mathsf{KeyGen}(\mathsf{par})\}.$$

Henceforth, we will write $\hat{\mathsf{JS}}$ to denote a joint-signature scheme JS augmented with key rerandomization algorithms, as defined above.

Security. We naturally extend the security definition of a joint-signature scheme to one with rerandomizable keys. That is, when the adversary calls its signing oracles, it can decide how the honest user's key is rerandomized by providing ρ.

Definition 6 (EUF-CMA-RK Security). *Let* $\hat{\mathsf{JS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RandSK}, \mathsf{RandPK}, \mathsf{KeyAggr}, \mathsf{SSign}, \mathsf{MSign}, \mathsf{Verify})$ *be a joint-signature scheme with rerandomizable keys. Consider the* **EUF-CMA-RK** *game defined in Figure 5. We say that* $\hat{\mathsf{JS}}$ *is EUF-CMA-RK secure if for all PPT adversaries* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\text{EUF-CMA-RK}}_{\mathcal{A},\hat{\mathsf{JS}}}(\lambda) := \Pr\left[\mathbf{EUF\text{-}CMA\text{-}RK}^{\mathcal{A}}_{\hat{\mathsf{JS}}}(\lambda) \Rightarrow 1\right].$$

Game EUF-CMA-RK$^{\mathcal{A}}_{\hat{\mathsf{JS}}}(\lambda)$

00 $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
01 $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{par})$
02 $\mathsf{sid} := 0,\ S_0 := \emptyset$
03 $S_1 := \emptyset,\ Q_0 := \emptyset,\ Q_1 := \emptyset$
04 $\mathtt{MSIGN} := (\mathtt{SIGN}_0, \mathtt{SIGN}_1, \mathtt{SIGN}_2)$
05 $(\mathcal{P}^*, m^*, \sigma^*, b^*, \rho^*) \leftarrow \mathcal{A}^{\mathtt{SSIGN},\mathtt{MSIGN}}(\mathsf{par}, \mathsf{pk})$
06 $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho^*)$
07 **if** $(\hat{\mathsf{pk}} \notin \mathcal{P}^*) \vee (b^* \notin \{0,1\})$:
08 **return** 0
09 **if** $b^* = 0$:
10 **if** $(\mathcal{P}^* \neq \{\hat{\mathsf{pk}}\}) \vee ((\hat{\mathsf{pk}}, m^*) \in Q_0)$:
11 **return** 0
12 **return** $\mathsf{Verify}(\hat{\mathsf{pk}}, m^*, \sigma^*)$
13 **else** :
14 **if** $(\mathcal{P}^*, m^*) \in Q_1$: **return** 0
15 **return** $\mathsf{Verify}(\mathsf{KeyAggr}(\mathcal{P}^*), m^*, \sigma^*)$

Oracle $\mathtt{SIGN}_0(\rho)$

16 $\mathsf{sid} := \mathsf{sid} + 1$
17 $k := \mathsf{sid}$
18 $S_0 := S_0 \cup \{k\}$
19 $\rho_k := \rho$
20 $(\mathsf{pm}_1, \mathsf{St}_{1,k}) \leftarrow \mathsf{Sign}_0()$
21 **return** pm_1

Oracle $\mathtt{SIGN}_1(k, \mathcal{P}, m, \tilde{\mathsf{pm}}_1)$

22 **let** $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$
23 $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho_k)$
24 **if** $(\hat{\mathsf{pk}} \notin \mathcal{P}) \vee (k \notin S_0)$:
25 **return** $\perp$
26 $S_0 := S_0 \setminus \{k\}$
27 $S_1 := S_1 \cup \{k\}$
28 $Q_1 := Q_1 \cup \{(\mathcal{P}, m)\}$
29 $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho_k)$
30 $(\mathsf{pm}_2, \mathsf{St}_{2,k}) \leftarrow \mathsf{Sign}_1(\mathcal{P}, \hat{\mathsf{sk}}, m, \mathsf{St}_{1,k}, \tilde{\mathsf{pm}}_1)$
31 **return** pm_2

Oracle $\mathtt{SIGN}_2(k, \tilde{\mathsf{pm}}_2)$

32 **if** $k \notin S_1$: **return** $\perp$
33 $S_1 := S_1 \setminus \{k\}$
34 $\sigma \leftarrow \mathsf{Sign}_2(\mathsf{St}_{2,k}, \tilde{\mathsf{pm}}_2)$
35 **return** σ

Oracle $\mathtt{SSIGN}(m, \rho)$

36 $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho)$
37 $Q_0 := Q_0 \cup \{(\hat{\mathsf{pk}}, m)\}$
38 $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$
39 $\sigma \leftarrow \mathsf{SSign}(\hat{\mathsf{sk}}, m)$
40 **return** σ

Fig. 5. The game **EUF-CMA-RK** for a joint-signature scheme $\hat{\mathsf{JS}}$ with rerandomizable keys and adversary $\mathcal{A}$. We highlight the differences to the **EUF-CMA** game for a joint-signature scheme (see Figure 9) in gray boxes.

5.1 Construction and Security Analysis

In Figure 4, we define a joint-signature scheme with rerandomizable keys that models the joint use of Schnorr and Musig2. We analyze its security here.

Theorem 1. *Let* $\hat{\mathsf{JS}}$ *denote the joint-signature scheme with rerandomizable keys as defined in Figure 4 (i.e., where* $\mathsf{JS} = \mathsf{Schnorr} + \mathsf{MuSig2}$*). Then, for any algebraic PPT algorithm* $\mathcal{A}$ *that makes at most* Q_{MS} *queries in total to signing oracles* $\mathtt{SIGN}_0, \mathtt{SIGN}_1, \mathtt{SIGN}_2$*, at most* Q_S *queries to signing oracle* $\mathtt{SSIGN}$*, and at most* $Q_{\mathsf{H}_a}, Q_{\mathsf{H}_b}, Q_{\mathsf{H}_c}$ *queries to random oracles* $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c$*, respectively, there are PPT algorithms* $\mathcal{B}$ *and* $\mathcal{C}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{C})$ *and*

$$\mathsf{Adv}^{\text{EUF-CMA-RK}}_{\mathcal{A},\hat{\mathsf{JS}}}(\lambda) \leq \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{B},\mathsf{GrGen}}(\lambda) + \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{C},\mathsf{GrGen}}(\lambda) + \frac{4Q^2 + 2NQ + (N+1)Q^2}{p}.$$

where $\ell = 2Q_{MS} + 1$*,* $Q = 3Q_{MS} + Q_S + Q_{\mathsf{H}_a} + 2Q_{\mathsf{H}_b} + Q_{\mathsf{H}_c} + 2$*, and* N *is the sum of sizes of all the public key multisets* $\mathcal{P}$ *that* $\mathcal{A}$ *has output or queried.*

Proof. Let $\mathcal{A}$ be an algebraic adversary against the EUF-CMA-RK security of $\hat{\mathsf{JS}}$. We show the statement via a sequence of games and reductions to the AOMDL assumption. This is done in the following sections.

Preparation: Hybrid Games We start by giving a sequence of games.

Game $\mathbf{G}_0$ (Real game). This is the real game **EUF-CMA-RK$^{\mathcal{A}}_{\hat{\mathsf{JS}}}$** for joint-signatures with rerandomizable keys. We recall the game to fix some notation. First, the game samples parameters $\mathsf{par} := (\mathbb{G}, p, g, \mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c) \leftarrow \mathsf{Setup}(1^\lambda)$, which consists of a group $\mathbb{G}$ of prime order p with generator g and random oracles $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p$. Here, H_c is used for the underlying standard signature Schnorr, while $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c$ are used for the multi-signature MuSig2. Note that the random oracle H_c to derive the challenge scalars is the same for both Schnorr and MuSig2. Further, the game samples a secret key

$\mathsf{sk} \leftarrow_\$ \mathbb{Z}_p$ and computes the public key $\mathsf{pk} := g^{\mathsf{sk}} \in \mathbb{G}$ for the only honest signer 1. This key is used for both standard Schnorr signatures and interactive MuSig2 multi-signatures. Henceforth, we may also write $(x, X) := (\mathsf{sk}, \mathsf{pk})$ for the key pair of signer 1. Then, the game runs $\mathcal{A}$ on input par and pk with access to the following oracles:

- *Signing oracle* $\mathtt{SSIGN}$ *for* Schnorr *with rerandomizable keys.* This oracle executes the signing algorithm SSign on rerandomized secret key $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$ for the standard signature Schnorr. Here, $\mathcal{A}$ can submit a query $\mathtt{SSIGN}(m, \rho)$ to obtain a Schnorr signature σ on the message m signed with the secret key $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$. The oracle then adds the pair $(\hat{\mathsf{pk}}, m)$ to a list Q_0, where $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho)$. Concretely, SSign samples $r \leftarrow_\$ \mathbb{Z}_p$, computes $R := g^r$, makes an internal hash query to obtain challenge $c := \mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$, computes $s := r + c(x + \rho)$, and returns the signature $\sigma := (R, s)$. The game stores the above variables ρ, $\hat{\mathsf{pk}}$ R, c, s involved in the j-th $\mathtt{SIGN}$ query as global variables $\bar{\rho}_j$, $\bar{X}_j$ $\bar{R}_j$, $\bar{c}_j$, $\bar{s}_j$, respectively.
- *Signing oracles* $\mathtt{SIGN}_0, \mathtt{SIGN}_1, \mathtt{SIGN}_2$ *for* MuSig2 *with rerandomizable keys.* These oracles run algorithms $\mathsf{Sign}_0, \mathsf{Sign}_1, \mathsf{Sign}_2$ on rerandomized secret key $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$ for the multi-signature MuSig2, respectively.
  Concretely, $\mathcal{A}$ can submit a query $\mathtt{SIGN}_0(\rho)$ to start a new signing session. The game runs $\mathsf{Sign}_0()$, which returns two nonces $(R_{1,1}, R_{1,2})$ and a state St_1. The game returns $(R_{1,1}, R_{1,2})$ to $\mathcal{A}$, adds the session identifier $k := \mathsf{sid}$ to a list S_0, and records $\rho_k := \rho$ and $\mathsf{St}_{1,k} := \mathsf{St}_1$. Specifically, Sign_0 samples $r_{1,1}, r_{1,2} \leftarrow_\$ \mathbb{Z}_p$, computes $R_{1,1} := g^{r_{1,1}}$, $R_{1,2} := g^{r_{1,2}}$, and returns $(R_{1,1}, R_{1,2})$ and $\mathsf{St}_1 := (r_{1,1}, r_{1,2})$. Further, $\mathcal{A}$ can submit a query $\mathtt{SIGN}_1(k, \mathcal{P}, m, \tilde{\mathsf{pm}}_1)$ with session identifier k, two aggregated nonces $\tilde{\mathsf{pm}}_1 = (R_1, R_2)$, and message m to be signed for public keys $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$. In that case, the game runs $\mathsf{Sign}_1(\mathcal{P}, \hat{\mathsf{sk}}, m, \mathsf{St}_{1,k}, \tilde{\mathsf{pm}}_1)$ with $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho_k)$ to obtain a signature share s_1 and returns it to $\mathcal{A}$. The game then removes the identifier k from S_0, adds it to another list S_1, and also adds the pair $(\mathcal{P}, m)$ to the list Q_1.
  Specifically, algorithm Sign_1 first computes the aggregate key $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$, for which it needs to query the scalar $a_1 := \mathsf{H}_a(\langle \mathcal{P} \rangle, \hat{\mathsf{pk}})$ with $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho_k) = \mathsf{pk} \cdot g^{\rho_k} = X \cdot g^{\rho_k}$. Recall that we require $\hat{\mathsf{pk}} \in \mathcal{P}$ and assume its index is 1. Then the algorithm makes hash queries to obtain $b := \mathsf{H}_b(\tilde{\mathsf{pk}}, \tilde{\mathsf{pm}}_1, m)$ and computes $R := R_1 R_2^b$ with $(R_1, R_2) := \tilde{\mathsf{pm}}_1$. Finally, it queries $c := \mathsf{H}(\tilde{\mathsf{pk}}, R, m)$ and returns $s_1 := r_{1,1} + b r_{1,2} + a_1 c(x + \rho_k)$.
  We ignore the oracle $\mathtt{SIGN}_2$, which would compute the final signature $\sigma := (R, s)$ with $s = \sum_{i \in [n]} s_i$. This is without loss of generality, since algorithm Sign_2 does not make use of any secret state and can be publicly run using the messages output in Sign_0 and Sign_1. Thus, one could always build a wrapper adversary that internally runs $\mathcal{A}$ and simulates the oracle $\mathtt{SIGN}_2$ for itself. Clearly, the wrapper adversary has the same running time and advantage as $\mathcal{A}$.
  The game stores the above variables $\hat{\mathsf{pk}}$, $R_{1,1}$, $R_{1,2}$, a_1, b, c, s_1 involved in the k-th multi-signature signing session as global variables X_k, $R_{k,1,1}$, $R_{k,1,2}$, $a_{k,1}$, b_k, c_k, $s_{k,1}$, respectively. Note that the input randomness was already stored as ρ_k.
- *Random oracles* $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c$. The random oracles H_a (used to derive the aggregation coefficients for the aggregated public key $\tilde{\mathsf{pk}}$ in MuSig2), H_b (used to derive the randomization coefficient for the aggregated nonce R in MuSig2), and H_c (used to derive the challenge in Schnorr and MuSig2) are implemented via lazy sampling.

At the end of the game, the adversary outputs a forgery $(\mathcal{P}^*, m^*, \sigma^*, b^*, \rho^*)$. Then, the game computes the rerandomized public key $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho^*)$ and outputs 1 if and only if (i) $\mathcal{P}^* = \{\hat{\mathsf{pk}}\}$, $(\hat{\mathsf{pk}}, m^*) \notin Q_0$ and $\mathsf{Verify}(\hat{\mathsf{pk}}, m^*, \sigma^*) = 1$ or (ii) $\hat{\mathsf{pk}} \in \mathcal{P}^*$, $(\mathcal{P}^*, m^*) \notin Q_1$, and $\mathsf{Verify}(\mathsf{KeyAggr}(\mathcal{P}^*), m^*, \sigma^*) = 1$. Here, condition (i) is checked if $b^* = 0$ and indicates that the adversary succeeded in forging a signature for Schnorr, while (ii) is checked if $b^* = 1$ and indicates that it succeeded in forging a signature for MuSig2. Further, we also write $\sigma^* = (R^*, s^*)$ for the forgery signature. Clearly,

$$\mathsf{Adv}^{\text{EUF-CMA-RK}}_{\mathcal{A}, \hat{\mathsf{JS}}}(\lambda) = \Pr[\mathbf{G}_0 \Rightarrow 1].$$

Before proceeding to the next game, we introduce the following change: We assume that all Q_{MS} multi-signature signing sessions are closed by the end of the game (i.e., immediately after the adversary outputs its forgery and terminates). This assumption can be enforced by constructing a wrapper adversary that internally runs $\mathcal{A}$ and, for every signing session with identifier k that remains open, forcefully closes it by querying $\mathtt{SIGN}_1(k, \mathcal{P}_k, m_k, \tilde{\mathsf{pm}}_1)$ for some $(\mathcal{P}_k, m_k, \tilde{\mathsf{pm}}_1)$ such that $(\mathcal{P}_k, m_k) \neq (\mathcal{P}^*, m^*)$. Moreover, if fewer than Q_{MS} signing sessions were opened in total, the wrapper adversary issues additional queries of the form $\mathtt{SIGN}_0(\rho_k)$ for some $\rho_k \neq \rho^*$ until exactly Q_{MS} sessions have been opened (and finally

closed). Clearly, the wrapper adversary has the same running time and achieves the same advantage as $\mathcal{A}$. Moreover, once and for all, we define $\ell := 2Q_{MS} + 1$.

Game $\mathbf{G}_1$ (Artificial hash queries): In this game, we introduce some artificial hash queries.

- Upon every query in the form of $\mathsf{H}_a(\langle\mathcal{P}\rangle, \cdot)$, the game queries $\mathsf{H}_a(\langle\mathcal{P}\rangle, \mathsf{pk}_i)$ for every $\mathsf{pk}_i \in \mathcal{P}$. Consequently, when a public key multiset $\mathcal{P}$ is queried to H_a for the first time, the value of $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ is immediately determined.
- Upon every query in the form of $\mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$, before returning b, the game queries $\mathsf{H}_c(\tilde{\mathsf{pk}}, R_1 R_2^b, m)$. This H_c query is the "subsequent" query of $\mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$, in the sense that if algorithm Sign_1 queries $\mathsf{H}_b(\tilde{\mathsf{pk}}, (R_1, R_2), m)$, then it must also query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R_1 R_2^b, m)$.

These changes do not affect the view of $\mathcal{A}$. We have

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \Pr[\mathbf{G}_1 \Rightarrow 1].$$

Let us count some numbers that we will use later.

- The total number of public key multisets $\mathcal{P}$ such that the aggregated key $\mathsf{KeyAggr}(\mathcal{P})$ has been determined is at most $Q_{\mathsf{H}_a} + Q_{MS} + 1$, including one for each H_a query, one for each $\mathtt{SIGN}_1$ query, and one for the verification of $\mathcal{A}$'s output.
- The total number of H_b queries is at most $Q_{\mathsf{H}_b} + Q_{MS}$, including those made by $\mathcal{A}$ and oracle $\mathtt{SIGN}_1$.
- The total number of H_c queries is at most $Q_{\mathsf{H}_b} + Q_{\mathsf{H}_c} + Q_S + Q_{MS} + 1$, including those made by $\mathcal{A}$, oracle $\mathtt{SSIGN}$, and oracle $\mathtt{SIGN}_1$, those artificial queries introduced in this game, and the query for the verification of $\mathcal{A}$'s output.

All these numbers are clearly upper-bounded by their sum $Q = 3Q_{MS} + Q_S + Q_{\mathsf{H}_a} + 2Q_{\mathsf{H}_b} + Q_{\mathsf{H}_c} + 2$.

Game $\mathbf{G}_2$ (Simulate single-signer signatures): In this game, we change the signing oracle $\mathtt{SSIGN}$. Recall that in the previous game $\mathbf{G}_1$, upon a signing query $\mathtt{SSIGN}(m, \rho)$, the oracle performs the standard $\mathsf{Schnorr}$ signing algorithm $\mathsf{SSign}(\hat{\mathsf{sk}}, m)$ using rerandomized key $\hat{\mathsf{sk}} := \mathsf{RandSK}(\mathsf{sk}, \rho)$. In this game, $\mathtt{SSIGN}$ instead performs the standard honest-verifier zero-knowledge (HVZK) simulator for $\mathsf{Schnorr}$. Specifically, the oracle uniformly samples $c, s \leftarrow_\$ \mathbb{Z}_p$, computes $R := g^s \hat{\mathsf{pk}}^{-c}$ with $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho)$, and programs the random oracle $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m) := c$. The game aborts if $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is already defined at the point of programming. The oracle returns the signature $\sigma := (R, s)$.

We argue that until the game aborts in $\mathtt{SSIGN}$, $\mathbf{G}_2$ is identical to $\mathbf{G}_1$ in the view of $\mathcal{A}$. In $\mathtt{SSIGN}$ in $\mathbf{G}_1$, when $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ has not been queried before, $c := \mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is indeed uniformly distributed over $\mathbb{Z}_p$. Also, s is uniformly distributed over $\mathbb{Z}_p$ because of the randomness of r (the discrete logarithm of R). The validity of the signature guarantees that $R = g^s \hat{\mathsf{pk}}^{-c}$. Therefore, the behavior of $\mathtt{SSIGN}$ in $\mathbf{G}_1$ is identical to that in $\mathbf{G}_2$ in the view of $\mathcal{A}$. Since R is uniformly distributed over $\mathbb{G}$, the probability that $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ has been defined is at most Q/p. By the union bound, we have

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq Q^2/p.$$

Game $\mathbf{G}_3$ (Rule out $\hat{\mathsf{pk}} = 1_\mathbb{G} := g^0$): In this game, we introduce an abort condition. The game aborts if the adversary $\mathcal{A}$ (i) issues a $\mathtt{SSIGN}$ or $\mathtt{SIGN}_1$ query with randomness ρ such that the corresponding rerandomized public key $\hat{\mathsf{pk}} := X g^\rho$ satisfies $\hat{\mathsf{pk}} = 1_\mathbb{G} := g^0$, or (ii) outputs a forgery with randomness ρ^* such that the rerandomized public key $\hat{\mathsf{pk}} := X g^{\rho^*}$ satisfies $\hat{\mathsf{pk}} = 1_\mathbb{G}$. We will prove the following lemma later in Section 6.3.

Lemma 1 ($\mathbf{G}_2 \to \mathbf{G}_3$). *There exists a PPT algorithm $\mathcal{C}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{C})$ such that*

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{C},\mathsf{GrGen}}(\lambda).$$

Game $\mathbf{G}_4$ (Rule out H_b collisions): In this game, we introduce an abort condition. The game aborts when there exists a collision in H_b, namely two distinct queries with equal returned value. The collision probability is at most Q^2/p. We have

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq Q^2/p.$$

Game $\mathbf{G}_5$ (Rule out $\mathsf{KeyAggr}$ collisions and hits): In this game, we introduce two abort conditions about key aggregation. We say a multi-set $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ is *non-trivial* if it contains some public key $\mathsf{pk}_i \neq 1_\mathbb{G}$ (i.e., it is trivial if $\mathcal{P} = \{1_\mathbb{G}, \ldots, 1_\mathbb{G}\}$). The abort conditions are checked whenever the value of $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ is determined for a non-trivial $\mathcal{P}$, namely when $\mathcal{P}$ is queried to H_a, $\mathtt{SIGN}_1$, or given as $\mathcal{A}$'s output. The two abort conditions are the following:

- *Collision*: The game aborts if $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ is equal to $\mathsf{KeyAggr}(\mathcal{P}')$ for some $\mathcal{P}' \neq \mathcal{P}$ where $\mathsf{KeyAggr}(\mathcal{P}')$ has already been determined.
- *Hitting* H_b *or* H_c *queries*: The game aborts if previously a query of the form $\mathsf{H}_b(\tilde{\mathsf{pk}}, \cdot)$ or $\mathsf{H}_c(\tilde{\mathsf{pk}}, \cdot)$ for $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ has already been made.

We claim that the probability of abort is negligible. Intuitively, this means that the value of $\mathsf{KeyAggr}(\mathcal{P})$ is unpredictable and appears random for any non-trivial $\mathcal{P}$.

To compute the probability, consider the last a_j being determined for $\mathsf{pk}_j \neq 1_{\mathbb{G}} \in \mathcal{P}$. Let S be the subset of $\mathcal{P}$ that contains every $\mathsf{pk}_i \neq \mathsf{pk}_j$ and $\mathsf{pk}_i \neq 1_{\mathbb{G}}$. Fix any values of $\{a_i\}_{\mathsf{pk}_i \in S}$. Suppose that pk_j occurs $n_j \geq 1$ times in $\mathcal{P}$. Then the aggregated key $\tilde{\mathsf{pk}} = \mathsf{pk}_j^{n_j a_j} \prod_{\mathsf{pk}_i \in S} \mathsf{pk}_i^{a_i}$ is uniformly distributed over $\mathbb{G}$ given the randomness of $a_j \in \mathbb{Z}_p$. Note that we have $\mathsf{pk}_j \neq 1_{\mathbb{G}}$ and $n_j < p$ (since $\mathcal{A}$ is polynomially bounded), so $\mathsf{pk}_j^{n_j} \neq 1_{\mathbb{G}}$, as $\mathbb{G}$ is a cyclic group of prime order p. Thus, the value of $\tilde{\mathsf{pk}}$ collides with a previous $\mathsf{KeyAggr}(\mathcal{P}')$ or hits a previous H_b or H_c query with probability at most

$$\frac{Q_{\mathsf{H}_a} + 2Q_{\mathsf{H}_b} + Q_{\mathsf{H}_c} + Q_S + 3Q_{MS} + 2}{p},$$

which is $\leq Q/p$. Here, note that a multi-signature session computes an aggregated public key and makes H_b and H_c queries containing this key, so it only contributes one distinct target for $\mathsf{KeyAggr}(\mathcal{P})$ to hit. By the union bound, we have

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq Q^2/p.$$

$\mathsf{KeyAggr}^{-1}$ Notation. By ruling out $\mathsf{KeyAggr}$ collisions, we have ensured that at any point of the game and for any $\tilde{\mathsf{pk}}$, if there exists a non-trivial $\mathcal{P}$ such that $\mathsf{KeyAggr}(\mathcal{P})$ has been determined and $\mathsf{KeyAggr}(\mathcal{P}) = \tilde{\mathsf{pk}}$, then such a $\mathcal{P}$ is unique. We use $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$ to denote this unique non-trivial $\mathcal{P}$ such that $\mathsf{KeyAggr}(\mathcal{P}) = \tilde{\mathsf{pk}}$. If there exists no non-trivial $\mathcal{P}$ such that $\mathsf{KeyAggr}(\mathcal{P}) = \tilde{\mathsf{pk}}$, then we say $\mathsf{KeyAggr}(\tilde{\mathsf{pk}})^{-1} = \perp$. This notation will be well-defined for all the following games, and we will keep using it.

Game $\mathbf{G}_6$ (The Coef_k tables): In this game, we introduce a key-value table $\mathtt{Coef}_k$ for every $k \in [Q_{MS}]$. If key-value pair $(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k$, then we write $\mathtt{Coef}_k[\bar{b}_k] = \bar{d}_k$. We use $\mathtt{Coef}_k.\mathtt{Keys}$ to denote the set of all keys in the table. We will see that the keys are guaranteed to be distinct.

- **Table initialization.** Each table $\mathtt{Coef}_k$ is initialized upon the k-th $\mathtt{SIGN}_0$ query $\mathtt{SIGN}_0(\rho_k)$. When the table is initialized, the game runs the following $\mathtt{CheckAndInsert}$ procedure on *every* H_b *query that has been made before*. Let $b = \mathsf{H}_b(\tilde{\mathsf{pk}}, R_1, R_2, m)$ be such a (previously made) query. The game executes the procedure $\mathtt{CheckAndInsert}(k, (\tilde{\mathsf{pk}}, R_1, R_2, m), b)$, defined as follows:
  1. Let $\hat{\mathsf{pk}}_k = \mathsf{RandPK}(X, \rho_k)$. By the change in game $\mathbf{G}_3$, we know that $\hat{\mathsf{pk}}_k \neq 1_{\mathbb{G}}$. This ensures that any $\mathcal{P}$ containing $\hat{\mathsf{pk}}_k$ is non-trivial.
  2. The procedure stops (and continues with the next H_b query that has not yet been visited) if $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) = \perp$ or $\hat{\mathsf{pk}}_k \notin \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$. Otherwise, let $\mathcal{P} := \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$.
  3. Insert (b, ac) into $\mathtt{Coef}_k$, where $a := \mathsf{H}_a(\langle \mathcal{P} \rangle, \hat{\mathsf{pk}}_k)$ and $c := \mathsf{H}_c(\tilde{\mathsf{pk}}, R_1 R_2^b, m)$.
- **Table maintenance.** For every H_b query $b = \mathsf{H}_b(\tilde{\mathsf{pk}}, R_1, R_2, m)$, the game performs $\mathtt{CheckAndInsert}(k, (\tilde{\mathsf{pk}}, R_1, R_2, m), b)$ for every $\mathtt{Coef}_k$ that has been initialized. That is, we maintain the invariant that for every k such that $\mathtt{Coef}_k$ has been initialized, and for every H_b query $b = \mathsf{H}_b(\tilde{\mathsf{pk}}, R_1, R_2, m)$ that has been made, the game has run $\mathtt{CheckAndInsert}(k, (\tilde{\mathsf{pk}}, R_1, R_2, m), b)$.

Before continuing, we present two properties of the table, stated as the following lemmas. First, for each fixed session k, the table $\mathtt{Coef}_k$ never contains duplicate keys - every entry is uniquely indexed. Second, from $\mathbf{G}_6$ onward, every time the second-round signing oracle $\mathtt{SIGN}_1$ outputs a value $s_{k,1}$ (with corresponding $a_{k,1}$ and c_k), it is guaranteed that the pair $(b_k, a_{k,1}c_k)$ appears in $\mathtt{Coef}_k$. Thus, informally, the role of $\mathtt{Coef}_k$ is to store all "potential coefficients" used in the second round of the signing session k.

Lemma 2. *For any fixed k, no key is inserted twice into the table $\mathtt{Coef}_k$.*

Proof. In game $\mathbf{G}_4$, we have excluded collisions in H_b. Since the keys of $\mathtt{Coef}_k$ are outputs of H_b, they must therefore be distinct.

Lemma 3. *From $\mathbf{G}_6$ on, whenever the $\mathtt{SIGN}_1$ query for the k-th multi-signature session $\mathtt{SIGN}_1(k, \mathcal{P}, m, (R_1, R_2))$ is made and returns $s_{k,1} := r_{k,1,1} + b_k r_{k,1,2} + a_{k,1}c_k(x + \rho_k)$, it holds that $(b_k, a_{k,1}c_k) \in \mathtt{Coef}_k$.*

Proof. The abort condition introduced in $\mathbf{G}_3$ guarantees that $\hat{\mathsf{pk}} := Xg^{\rho_k} \neq 1_{\mathbb{G}}$, so for a valid $\mathtt{SIGN}_1$ query, $\mathcal{P}$ is non-trivial. Then the abort condition introduced in $\mathbf{G}_5$ guarantees that the value of $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ has been determined before the query $\mathsf{H}_b(\tilde{\mathsf{pk}}, R_1, R_2, m)$ involved in $\mathtt{SIGN}_1$ was made for the first time. In this case, $\mathbf{G}_6$ would insert or have inserted $(b_k, a_{k,1}c_k)$ to $\mathtt{Coef}_k$.

Since the query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R_1 R_2^b, m)$ made in the check-and-insert procedure was already introduced in $\mathbf{G}_1$, this game does not introduce any additional distinct queries. Consequently, the two games are indistinguishable to $\mathcal{A}$ and we have

$$\Pr[\mathbf{G}_5 \Rightarrow 1] = \Pr[\mathbf{G}_6 \Rightarrow 1].$$

Game $\mathbf{G}_7$ (Separating aggregated keys and single keys): We introduce an abort condition before the game returns the final result. For this, we define the set $\mathcal{S}_{\text{randkeys}}$ to contain all rerandomized public keys that have occurred in this game, including

- the rerandomized key $Xg^{\bar{\rho}_j}$ for each $\mathtt{SSIGN}$ query,
- the rerandomized key Xg^{ρ_k} for each $\mathtt{SIGN}_0$ query,
- the rerandomized key Xg^{ρ^*} for $\mathcal{A}$'s final output.

The game aborts if there exists $\hat{\mathsf{pk}}, \hat{\mathsf{pk}}' \in \mathcal{S}_{\text{randkeys}}$ such that $\hat{\mathsf{pk}} \in \mathsf{KeyAggr}^{-1}(\hat{\mathsf{pk}}')$. This condition essentially asserts the existence of a non-trivial $\mathcal{P}$ for which $\mathsf{KeyAggr}(\mathcal{P})$ has been determined (and rerandomized keys $\hat{\mathsf{pk}}, \hat{\mathsf{pk}}' \in \mathcal{S}_{\text{randkeys}}$), such that $\hat{\mathsf{pk}} \in \mathcal{P}$ and $\hat{\mathsf{pk}}' = \mathsf{KeyAggr}(\mathcal{P})$. We will prove the following lemma later in Section 6.1. Recall the notation $\ell := 2Q_{MS} + 1$.

Lemma 4 ($\mathbf{G}_6 \to \mathbf{G}_7$). *There exists a PPT algorithm $\mathcal{D}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{D})$ such that*

$$|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{D},\mathsf{GrGen}}(\lambda) + NQ/p.$$

Bounding the probability $\Pr[\mathbf{G}_7 \Rightarrow 1]$: It remains to bound the probability that $\mathbf{G}_7$ outputs 1. We do so by considering two cases, namely $b^* = 0$ (forgery for single-signer $\mathsf{Schnorr}$) and $b^* = 1$ (forgery for $\mathsf{MuSig2}$), via the following lemmas.

Lemma 5 (Bounding $\mathbf{G}_7$ and $b^* = 0$). *There exists a PPT algorithm $\mathcal{B}_0$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_0)$ and*

$$\Pr[\mathbf{G}_7 \Rightarrow 1 \wedge b^* = 0] \leq \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{B}_0,\mathsf{GrGen}}(\lambda) + Q^2/p.$$

Lemma 6 (Bounding $\mathbf{G}_7$ and $b^* = 1$). *There exists a PPT algorithm $\mathcal{B}_1$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_1)$ and*

$$\Pr[\mathbf{G}_7 \Rightarrow 1 \wedge b^* = 1] \leq \mathsf{Adv}^{\ell\text{-AOMDL}}_{\mathcal{B}_1,\mathsf{GrGen}}(\lambda) + NQ/p + (N+1)Q^2/p.$$

We give the proofs of these lemmas later in Sections 6.2 and 6.3, respectively. The above sequence of games and the final bound on $\Pr[\mathbf{G}_7 \Rightarrow 1]$ complete the proof.

In the subsequent sections, we prove the remaining lemmas by providing the corresponding reductions. Before doing so, we introduce some notation (related to the AGM) that will be useful in the proofs that follow.

6 Proofs of Lemmas

In this section, we present the proofs of the remaining lemmas used in the proof of Theorem 1, namely Lemmas 4, 5, and 6.

6.1 Proof of Lemma 4 ($\mathbf{G_6} \to \mathbf{G_7}$)

Proof (Proof of Lemma 4). Reduction $\mathcal{D}$ follows the structure described in Section 6.3. It simulates $\mathbf{G}_7$. When $\mathbf{G}_7$ aborts, $\mathcal{D}$ learns $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, ρ, ρ' such that

$$Xg^{\rho'} = \mathsf{KeyAggr}(\mathcal{P}),$$

where $Xg^{\rho} = \mathsf{pk}_j$ without loss of generality. Suppose that pk_j occurs n_j times in $\mathcal{P}$. Then we have

$$Xg^{\rho'} = (Xg^{\rho})^{n_j a_j} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} \mathsf{pk}_i^{a_i},$$

where $a_i := \mathsf{H}_a(\langle\mathcal{P}\rangle, \mathsf{pk}_i)$ for $i \in [n]$. Thus, $\mathcal{D}$ learns the discrete logarithm $\rho' - \rho n_j a_j$ of

$$X^{n_j a_j - 1} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} \mathsf{pk}_i^{a_i},$$

namely the value of $\bar{\mathtt{rep}}(X^{n_j a_j - 1} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} \mathsf{pk}_i^{a_i}) \cdot \mathbf{x}$, where $\mathbf{x} = (r_{1,1,1}, r_{1,1,2}, \ldots, r_{Q_{MS},1,1}, r_{Q_{MS},1,2}, x)^T$ as defined in Section 6.3. If

$$\begin{aligned} \bar{\mathtt{rep}}(X^{n_j a_j - 1} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} \mathsf{pk}_i^{a_i}) &= \underbrace{\bar{\mathtt{rep}}(X^{n_j a_j - 1}) + \sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} a_i \cdot \bar{\mathtt{rep}}(\mathsf{pk}_i)}_{=:\mathbf{v}_{\mathcal{P},j}} \\ &\notin \mathsf{Span}\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]}, \end{aligned} \tag{1}$$

or in other words, $\mathbf{v}_{\mathcal{P},j}$ is not in the row span of matrix $\mathbf{M}$ as defined in Section 6.3, then $\mathcal{D}$ can solve the input AOMDL instance.

It only remains to bound the probability that Equation (1) does not hold (i.e., the vector $\mathbf{v}_{\mathcal{P},j}$ is in the row span of $\mathbf{M}$). In order to do so, we define a bad event $\mathsf{BadP1}$, relate it with the event that Equation (1) does not hold, and bound the probability of $\mathsf{BadP1}$.

Definition of $\mathsf{BadP1}$. Let $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ be a non-trivial public key multiset that has been queried to H_a or $\mathtt{SIGN}_1$, so every $a_i := \mathsf{H}_a(\langle\mathcal{P}\rangle, \mathsf{pk}_i)$ for $i \in [n]$ has been defined. We say $\mathcal{P}$ is bad *with respect to* $\mathsf{pk}_j \in \mathcal{P}$ if there exists

$$\mathbf{v}_k \in \{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{\bar{b}_k} X^{\bar{d}_k})\}_{(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k} \cup \{\mathbf{0}\} \quad \text{for} \quad k \in [Q_{MS}],$$

such that

$$\mathbf{v}_{\mathcal{P},j} := \bar{\mathtt{rep}}(X^{n_j a_j - 1}) + \sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} a_i \cdot \bar{\mathtt{rep}}(\mathsf{pk}_i) \in \mathsf{Span}\{\mathbf{v}_k\}_{k \in [Q_{MS}]}. \tag{2}$$

Intuitively, $\mathcal{P}$ is bad with respect to $\mathsf{pk}_j \in \mathcal{P}$ if there exists a way to make $\mathtt{SIGN}_1$ queries and therefore manipulate the set $\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]}$, such that when the abort condition of $\mathbf{G}_7$ is triggered by $\mathcal{P}$ with pk_j being the randomized key, Equation 1 does not hold. The set

$$\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{\bar{b}_k} X^{\bar{d}_k})\}_{(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k} \cup \{\mathbf{0}\}$$

contains the algebraic representations of all the potential $\mathtt{DLOG}$ queries that $\mathcal{D}$ may make in $\mathtt{SIGN}_1$ for the k-th multi-signature session. The zero vector helps handle the special case where $\mathtt{Coef}_k$ has not been initialized yet. Moreover, we say $\mathcal{P}$ is bad if there exists pk_j such that $\mathcal{P}$ is bad with respect to pk_j. Note that a *good* (i.e., not bad) $\mathcal{P}$ may turn bad due to more key-value pairs being inserted into $\mathtt{Coef}_k$. We say that $\mathsf{BadP1}$ occurs if, *whenever the game aborts*, there exists a bad $\mathcal{P}$.

Then this proof can be completed by the following two claims.

Claim. If $\mathbf{G}_7$ aborts and Equation (1) does not hold, then $\mathsf{BadP1}$ occurs.

Claim. It holds that $\Pr[\mathsf{BadP1}] \leq NQ/p$.

Proof (Proof of Claim 6.1). Suppose that Equation (1) does not hold, namely

$$\mathbf{v}_{\mathcal{P},j} \in \mathsf{Span}\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]}.$$

By Lemma 3, we have $(b_k, a_{k,1} c_k) \in \mathtt{Coef}_k$ for every $k \in [Q_{MS}]$. Therefore, when the abort condition is triggered by $\mathcal{P}$, ρ, ρ', it must hold that $\mathcal{P}$ is bad with respect to pk_j. Here, we simply have

$$\mathbf{v}_k = \bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k}) \quad \text{for} \quad k \in [Q_{MS}].$$

Probability of $\mathsf{BadP1}$. To prove Claim 6.1, let us take a closer look at the meaning of a public key multiset $\mathcal{P}$ being bad in the following claim.

Claim. Fix any $\mathcal{P}$ and $\mathsf{pk}_j \in \mathcal{P}$. Suppose that pk_j occurs in $\mathcal{P}$ for n_j times. Let $(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'')$ be $\sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} a_i \cdot \bar{\mathtt{rep}}(\mathsf{pk}_i)$. Then $\mathcal{P}$ is bad with respect to $\mathsf{pk}_j \in \mathcal{P}$ if and only if both of the following two conditions hold:

- For every $k \in [Q_{MS}]$, either $\alpha''_k = \beta''_k = 0$, or $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$.
- It further holds that
$$\sum_{k\in[Q_{MS}],\alpha''_k\neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k] = \gamma'' + n_j a_j - 1.$$

Proof (Proof of Claim 6.1). By definition, $\mathcal{P}$ is bad with respect to pk_j if and only if there exist $\{\eta_k\}_{k\in[Q_{MS}]}$ and $\{(\bar{e}_k, \bar{b}_k, \bar{d}_k)\}_{k\in[Q_{MS}]}$ such that

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0) \quad \text{or} \quad \bar{e}_k = 1,\ (\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k \tag{3}$$

and

$$(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'' + n_j a_j - 1) = \left(\eta_1\bar{e}_1, \eta_1\bar{b}_1, \ldots, \eta_{Q_{MS}}\bar{e}_{Q_{MS}}, \eta_{Q_{MS}}\bar{b}_{Q_{MS}}, \sum_k \eta_k \bar{d}_k\right). \tag{4}$$

Now we prove Claim 6.1 in the "if" direction, i.e., we assume both conditions in the claim hold and show that $\mathcal{P}$ is bad with respect to pk_j. For every $k \in [Q_{MS}]$, if $\alpha''_k = \beta''_k = 0$, then let

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0) \quad \text{and} \quad \eta_k = 0.$$

If $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$, then let

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (1, \beta''_k/\alpha''_k, \mathtt{Coef}_k[\beta''_k/\alpha''_k]) \quad \text{and} \quad \eta_k = \alpha''_k.$$

Then, by inspection, Equations (3–4) hold, so $\mathcal{P}$ is bad with respect to pk_j.

Next, we prove the "only if" direction. Assume that $\mathcal{P}$ is bad with respect to pk_j. Consequently, there exists a way to choose $(\bar{e}_k, \bar{b}_k, \bar{d}_k)$ and η_k such that Equations (3–4) hold.

First, let us prove the first condition in Claim 6.1. For any $k \in [Q_{MS}]$, let us consider components (α''_k, β''_k). We know that $(\alpha''_k, \beta''_k) = (\eta_k\bar{e}_k, \eta_k\bar{b}_k)$.

- If $\alpha''_k = 0$, then we have $\eta_k = 0$ or $\bar{e}_k = 0$. If $\eta_k = 0$, then $\eta_k\bar{b}_k = 0$. On the other hand, if $\bar{e}_k = 0$, then $(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0)$ by assumption. Thus, in both cases, it follows that $\eta_k\bar{b}_k = 0$. Therefore, $\alpha''_k = 0$ implies that $\beta''_k = 0$.
- If $\alpha''_k \neq 0$, then it follows that $\bar{e}_k = 1$ and $\eta_k = \alpha''_k$ (the only other alternative, $\bar{e}_k = 0$, does not work). Thus, we conclude that $\bar{b}_k = \beta''_k/\eta_k = \beta''_k/\alpha''_k$ is in $\mathtt{Coef}_k.\mathtt{Keys}$. Therefore, $\alpha''_k \neq 0$ implies that $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$.

Thus, the first condition holds.

Moreover, for those $\alpha''_k \neq 0$, the choice of η_k and $(\bar{e}_k, \bar{b}_k, \bar{d}_k)$ are already fixed to be α''_k and $(1, \beta''_k/\alpha''_k, \mathtt{Coef}_k[\beta''_k/\alpha''_k])$, respectively. For $\alpha''_k = 0$, we have $\eta_k = 0$ or $(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0)$, so $\eta_k\bar{d}_k$ is fixed to be 0.

Consequently, we have

$$\gamma'' + n_j a_j - 1 = \sum_k \eta_k \bar{d}_k = \sum_{k\in[Q_{MS}],\alpha''_k\neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k],$$

where the first equation follows from Equation (4). Thus, the second condition in Claim 6.1 also holds.

Proof (Proof of Claim 6.1). For each non-trivial $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, we will bound the probability that it is bad at the end of the game in two steps. First, we bound the probability that it is bad when the value of $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ is determined. Second, if it was good (i.e., not bad) when $\tilde{\mathsf{pk}}$ was determined, then we bound the probability that it turns bad later.

Fix any $\mathsf{pk}_j \in \mathcal{P}$. Suppose that pk_j occurs in $\mathcal{P}$ for n_j times. Let $(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'')$ be $\sum_{i:\mathsf{pk}_i\neq\mathsf{pk}_j\in\mathcal{P}} a_i \cdot \mathtt{re\bar{p}}(\mathsf{pk}_i)$. By Claim 6.1, $\mathcal{P}$ is bad with respect to pk_j only if

- For every $k \in [Q_{MS}]$, either $\alpha''_k = \beta''_k = 0$, or $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$.
- It further holds that
$$\sum_{k\in[Q_{MS}],\alpha''_k\neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k] = \gamma'' + n_j a_j - 1.$$

Conditioned on a_i for every $\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}$, $(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'')$ is fixed, while a_j remains uniformly random over $\mathbb{Z}_p$. Further, by Lemma 2, we know that for each $k \in [Q_{MS}]$, the keys stored in $\mathtt{Coef}_k$ are distinct. Therefore, $\mathtt{Coef}_k[\beta''_k/\alpha''_k]$ is uniquely determined and fixed.[9] Thus, if the first condition holds, then the second condition holds with probability $1/p$. It follows that each non-trivial $\mathcal{P}$ is bad with respect to any $\mathsf{pk}_j \in \mathcal{P}$ with probability at most $1/p$ when the value of $\mathsf{KeyAggr}(\mathcal{P})$ is determined.

Again by Claim 6.1, conversely, $\mathcal{P}$ is not bad with respect to $\mathsf{pk}_j \in \mathcal{P}$ only if at least one of the following holds:

- There exists $k \in [Q_{MS}]$ such that $\alpha''_k = 0$ and $\beta''_k \neq 0$.
- There exists $k \in [Q_{MS}]$ such that $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \notin \mathtt{Coef}_k.\mathtt{Keys}$.
- For every $k \in [Q_{MS}]$, $\alpha''_k \neq 0$ implies $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$. Meanwhile, it holds that

$$\sum_{k \in [Q_{MS}], \alpha''_k \neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k] \neq \gamma'' + n_j a_j - 1.$$

Suppose that $\mathcal{P}$ was not bad with respect to pk_j at the time it was determined. As additional key-value pairs are inserted into $\mathtt{Coef}_k$, only the second condition can change from true to false. This is because no duplicate keys can be inserted into $\mathtt{Coef}_k$; once the value of $\mathtt{Coef}_k[\beta''_k/\alpha''_k]$ is defined, it will never be updated. Consequently, the third condition cannot change from true to false. Therefore, to bound the probability that a good query becomes bad, it suffices to analyze the probability that the second condition becomes false.

Suppose that the second condition held when $\mathsf{KeyAggr}(\mathcal{P})$ was determined. Then there exists a certain k such that $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \notin \mathtt{Coef}_k.\mathtt{Keys}$. By definition, $\alpha''_k \neq 0$ implies that the k-th signing session has already started (before $\mathcal{P}$ was output by $\mathcal{A}$ and $\mathsf{KeyAggr}(\mathcal{P})$ was determined). In particular, $\mathtt{Coef}_k$ has already been initialized. For table maintenance, however, the keys inserted into $\mathtt{Coef}_k$ are the uniformly random values returned by H_b queries. Thus, a newly inserted key matches the fixed value β''_k/α''_k with probability $1/p$. Overall, $\mathcal{P}$ becomes bad with probability at most $(Q_{MS} + Q_{\mathsf{H}_b})/p$.

The above analysis has shown that at the end of the game, each $\mathcal{P}$ is bad *with respect to any* $\mathsf{pk}_j \in \mathcal{P}$ with probability at most $(Q_{MS} + Q_{\mathsf{H}_b} + 1)/p < Q/p$. By the union bound, each $\mathcal{P}$ of size n is bad at the end of the game with probability at most nQ/p. In total, there exists a bad $\mathcal{P}$ at the end of the game with probability at most NQ/p.

6.2 Proof of Lemma 5 (the case $b^* = 0$)

Proof (Proof of Lemma 5). Reduction $\mathcal{B}_0$ follows the structure described in Section 6.3. It simulates $\mathbf{G}_7$ until the game returns. If $\mathcal{A}$ wins with $b^* = 0$, then (R^*, s^*) is a valid signature on m^* under rerandomized key $\hat{\mathsf{pk}}^* := \mathsf{RandPK}(\mathsf{pk}, \rho^*) = X g^{\rho^*}$. Namely, it holds that $g^{s^*} = R^*(X g^{\rho^*})^{c^*}$, where $c^* := \mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$. The winning condition also requires that $(\hat{\mathsf{pk}}^*, m^*) \notin Q_0$, so $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ was not programmed in $\mathtt{SSIGN}$.

$\mathcal{A}$ provides the reduced algebraic representation $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma, \delta)$ of R^*, such that

$$R^* = g^\delta \prod_{k \in [Q_{MS}]} (R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k}) X^\gamma.$$

Since $g^{s^*} = R^*(X g^{\rho^*})^{c^*}$, $\mathcal{B}$ learns the discrete logarithm $s^* - c^*\rho^*$ of

$$R^* X^{c^*} = g^\delta \prod_{k \in [Q_{MS}]} (R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k}) X^{\gamma + c^*}.$$

Consequently, it learns the value of

$$\bar{\mathtt{rep}}(R^* X^{c^*}) \cdot \mathbf{x} = (\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma + c^*) \cdot \mathbf{x},$$

where $\mathbf{x} = (r_{1,1,1}, r_{1,1,2}, \ldots, r_{Q_{MS},1,1}, r_{Q_{MS},1,2}, x)^T$ as defined in Section 6.3. If

$$\bar{\mathtt{rep}}(R^* X^{c^*}) = (\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma + c^*) \notin \mathsf{Span}\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]}, \tag{5}$$

[9] If there were two possible values of $\mathtt{Coef}_k[\beta''_k/\alpha''_k]$ for a fixed key β''_k/α''_k, then the left-hand side of the equation $\sum_{k \in [Q_{MS}], \alpha''_k \neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k] = \gamma'' + n_j a_j - 1$ could take up to $2^{Q_{MS}}$ different values, potentially enabling the adversary to mount ROS-type or k-list-type attacks.

then $\mathcal{B}$ can solve the input AOMDL instance, as argued in Section 6.3.

It only remains to bound the probability that Equation (5) does not hold. To do so, we define a bad event $\mathsf{BadHc1}$, relate it with the event that Equation (5) does not hold, and bound the probability of $\mathsf{BadHc1}$.

Excluded Pair. For any $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$,[10] let the returned value be c. We now define the "excluded key-value pair" $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ for $k \in [Q_{MS}]$ with respect to this query. We let $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}$ be the key of this pair. Recall that in $\mathbf{G}_1$, we have introduced artificial H_c queries in H_b. Moreover, in $\mathbf{G}_6$, we have introduced $\mathtt{Coef}_k$ table maintenance in H_b, where for each $k \in [Q_{MS}]$, some key-value pair may be inserted into $\mathtt{Coef}_k$. In short, if $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was artificial, then we define $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ to be the pair inserted into $\mathtt{Coef}_k$ in the corresponding H_b query. If $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was not artificial, or it was, but the table maintenance did not insert any key-value pair into $\mathtt{Coef}_k$, then we define $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ to be $\perp$.

Specifically, recall that in response to any query $\mathsf{H}_b(\hat{\mathsf{pk}}, (R_1, R_2), m)$, the game makes an artificial query (introduced in $\mathbf{G}_1$) in the form of $\mathsf{H}_c(\hat{\mathsf{pk}}, R_1 R_2^b, m)$, where b is the returned value of the current H_b query. If $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was not an artificial query made in H_b, then we define $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m) = \perp$.

Now, consider the case that $c = \mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was an artificial query made in some H_b query $b = \mathsf{H}_b(\hat{\mathsf{pk}}, (R_1, R_2), m)$ (which means $R = R_1 R_2^b$). Recall that in $\mathbf{G}_6$, we have introduced table maintenance in H_b, where the game runs $\mathtt{CheckAndInsert}(k, (\hat{\mathsf{pk}}, R_1, R_2, m), b)$ for every opened multi-signature session k. If the check passes, i.e. $\mathsf{KeyAggr}^{-1}(\hat{\mathsf{pk}}) \neq \perp$ and $\hat{\mathsf{pk}}_k := X g^{\rho_k} \in \mathsf{KeyAggr}^{-1}(\hat{\mathsf{pk}})$, then the procedure inserts (b, ac) into $\mathtt{Coef}_k$, where $a := \mathsf{H}_a(\langle \mathsf{KeyAggr}^{-1}(\hat{\mathsf{pk}}) \rangle, \hat{\mathsf{pk}}_k)$. If the check does not pass, then we define $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m) = \perp$. If the check passes, then we define $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ to be this pair (b, ac) inserted into $\mathtt{Coef}_k$.

Equivalently, we can define excluded pair by modifying the game (without affecting the view of the adversary). We consider $\mathtt{Excl}_k$ as a table for every $k \in [Q_{MS}]$. In response to any query $\mathsf{H}_b(\hat{\mathsf{pk}}, (R_1, R_2), m)$, if the artificial query $\mathsf{H}_c(\hat{\mathsf{pk}}, R_1 R_2^b, m)$ is a new query, then when procedure $\mathtt{CheckAndInsert}(k, (\hat{\mathsf{pk}}, R_1, R_2, m), b)$ inserts the pair (b, ac) into $\mathtt{Coef}_k$, we let the game additionally record this pair (b, ac) as $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R_1 R_2^b, m)$.

Definition of $\mathsf{BadHc1}$. We say a query $c = \mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is *bad*[11] if there exists

$$\mathbf{v}_k \in \{\bar{\mathsf{rep}}(R_{k,1,1} R_{k,1,2}^{\bar{b}_k} X^{\bar{d}_k})\}_{(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)\}} \cup \{\mathbf{0}\}$$

for $k \in [Q_{MS}]$, such that

$$\bar{\mathsf{rep}}(R \cdot X^c) \in \mathsf{Span}\{\mathbf{v}_k\}_{k \in [Q_{MS}]}. \tag{6}$$

Here, we exclude $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ from the allowable choice of $(\bar{b}_k, \bar{d}_k)$. Looking ahead, in the proof of Claim 6.2, we will see that excluding this pair from the choice of $\mathbf{v}_k$ is crucial for bounding the probability of $\mathsf{BadHc1}$. Note that a good query may turn bad later due to more key-value pairs inserted into $\mathtt{Coef}_k$. We say that $\mathsf{BadHc1}$ occurs if, *whenever the game returns* (and $b^* = 0$), there exists a bad query.

Then this proof can be completed by the following two claims.

Claim. If $\mathbf{G}_7$ returns (and $b^* = 0$) and Equation (5) does not hold, then $\mathsf{BadHc1}$ occurs.

Claim. It holds that $\Pr[\mathsf{BadHc1}] \leq Q^2/p$.

Proof (Proof of Claim 6.2). Suppose that Equation (5) does not hold, namely

$$\bar{\mathsf{rep}}(R^* X^{c^*}) \in \mathsf{Span}\{\bar{\mathsf{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]}.$$

We argue that $c^* = \mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ is a bad query; in particular, $\mathsf{BadHc1}$ occurs.

First of all, we need to show that $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ is indeed a query instead of programmed in $\mathtt{SSIGN}$. To see this, recall that the winning condition of $\mathcal{A}$ requires that $(\hat{\mathsf{pk}}^*, m^*) \notin Q_0$, so $\mathtt{SSIGN}$ must never have signed m^* under $\hat{\mathsf{pk}}^*$. In particular, $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ was never programmed in $\mathtt{SSIGN}$.

Furthermore, by Lemma 3, we have $(b_k, a_{k,1} c_k) \in \mathtt{Coef}_k$ for every $k \in [Q_{MS}]$. It therefore remains to show that $(b_k, a_{k,1} c_k) \neq \mathtt{Excl}_k(\hat{\mathsf{pk}}^*, R^*, m^*)$. Then we choose $\mathbf{v}_k = \bar{\mathsf{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})$ for $k \in [Q_{MS}]$, and the claim follows.

[10] For the proof in the case $b^* = 0$, we write the input to H_c (as well as to the related H_b queries) in the form $\hat{\mathsf{pk}}$. Later, in the proof for the case $b^* = 1$, we write the input to H_c (and related H_b queries) in the form $\tilde{\mathsf{pk}}$.

[11] We stress that only (possibly artificial) *queries* to H_c can be classified as bad, and not hash values that are programmed by the reduction in the absence of any corresponding external query.

Let us prove that $(b_k, a_{k,1}c_k) \neq \mathtt{Excl}_k(\hat{\mathsf{pk}}^*, R^*, m^*)$. That is, we want to show that if $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ was an artificial query made in H_b (introduced in $\mathbf{G}_1$), then $(b_k, a_{k,1}c_k)$ must not be the key-value pair inserted into $\mathtt{Coef}_k$ in the `CheckAndInsert` procedure for this H_b query (introduced in $\mathbf{G}_6$). In fact, we show that if $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ was an artificial query made in H_b, then the corresponding `CheckAndInsert` procedure did not insert any key-value pair, since the check did not pass. Consequently, we have $\mathtt{Excl}_k(\hat{\mathsf{pk}}^*, R^*, m^*) = \bot$ for every $k \in [Q_{MS}]$.

Recall that for a query in the form of $\mathsf{H}_b(\tilde{\mathsf{pk}}, \cdot)$, an artificial query in the form of $\mathsf{H}_c(\tilde{\mathsf{pk}}, \cdot)$ is made. Then the `CheckAndInsert` procedure inserts a key-value pair to $\mathtt{Coef}_k$ only if $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) \neq \bot$.

On the other hand, recall that we have ruled out the possibility that $\hat{\mathsf{pk}}^*$ equals to some $\mathsf{KeyAggr}(\mathcal{P})$ in $\mathbf{G}_7$. Therefore, if $\mathsf{H}_c(\hat{\mathsf{pk}}^*, R^*, m^*)$ was an artificial query made in a query in the form of $\mathsf{H}_b(\hat{\mathsf{pk}}^*, \cdot)$, then the corresponding `CheckAndInsert` procedure must not insert any key-value pair, since $\mathsf{KeyAggr}^{-1}(\hat{\mathsf{pk}}^*) = \bot$.

Probability of BadHc1. To prove Claim 6.2, let us take a close look at the meaning of a query being bad in the following claim.

Claim. Fix any query $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$. Let $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ be $\mathtt{r\bar{e}p}(R)$, and c be the returned value. Then $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is a bad query if and only if both of the following conditions hold:

- For every $k \in [Q_{MS}]$, either $\alpha_k = \beta_k = 0$, or $\alpha_k \neq 0$ and $\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- It further holds that
$$\sum_{k \in [Q_{MS}], \alpha_k \neq 0} \alpha_k \cdot \mathtt{Coef}_k[\beta_k/\alpha_k] = \gamma + c.$$

Proof (Proof of Claim 6.2). By definition, $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is bad if and only if there exist $\{\eta_k\}_{k \in [Q_{MS}]}$ and $\{(\bar{e}_k, \bar{b}_k, \bar{d}_k)\}_{k \in [Q_{MS}]}$ such that

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0) \quad \text{or} \quad \bar{e}_k = 1,\ (\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)\} \tag{7}$$

and

$$(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma + c) = \left(\eta_1\bar{e}_1, \eta_1\bar{b}_1, \ldots, \eta_{Q_{MS}}\bar{e}_{Q_{MS}}, \eta_{Q_{MS}}\bar{b}_{Q_{MS}}, \sum_k \eta_k \bar{d}_k\right). \tag{8}$$

Now we prove Claim 6.2 in the "if" direction, i.e., we assume both conditions in the claim hold and show that $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is bad. For every $k \in [Q_{MS}]$, if $\alpha_k = \beta_k = 0$, then let

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0) \quad \text{and} \quad \eta_k = 0.$$

If $\alpha_k \neq 0$ and $\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$, then let

$$(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (1, \beta_k/\alpha_k, \mathtt{Coef}_k[\beta_k/\alpha_k]) \quad \text{and} \quad \eta_k = \alpha_k.$$

Then Equations (7–8) hold, so $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is bad.

Next, we prove the "only if" direction. Assume that $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is bad. Thus, there exists a way to choose $(\bar{e}_k, \bar{b}_k, \bar{d}_k)$ and η_k such that Equations (7–8) hold.

First, let us prove the first condition in Claim 6.2. For any $k \in [Q_{MS}]$, consider the component (α_k, β_k). We know that $(\alpha_k, \beta_k) = (\eta_k\bar{e}_k, \eta_k\bar{b}_k)$.

- If $\alpha_k = 0$, then we have $\eta_k = 0$ or $\bar{e}_k = 0$. If $\eta_k = 0$, then $\eta_k\bar{b}_k = 0$. On the other hand, if $\bar{e}_k = 0$, then $(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0)$ by assumption. Thus, in both cases, it follows that $\eta_k\bar{b}_k = 0$. Therefore, $\alpha_k = 0$ implies that $\beta_k = 0$.
- If $\alpha_k \neq 0$, then it follows that $\bar{e}_k = 1$ and $\eta_k = \alpha_k$ (the only other alternative, $\bar{e}_k = 0$, does not work). Thus, $\bar{b}_k = \beta_k/\eta_k = \beta_k/\alpha_k$ is in $\mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$. Therefore, $\alpha_k \neq 0$ implies that
$$\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}.$$

Thus, the first condition holds.

Moreover, for those $\alpha_k \neq 0$, the choice of η_k and $(\bar{e}_k, \bar{b}_k, \bar{d}_k)$ are already fixed to be α_k and $(1, \beta_k/\alpha_k, \mathtt{Coef}_k[\beta_k/\alpha_k])$, respectively.[12] For $\alpha_k = 0$, we have $\eta_k = 0$ or $(\bar{e}_k, \bar{b}_k, \bar{d}_k) = (0,0,0)$, so $\eta_k\bar{d}_k$ is fixed to be 0.

[12] Again, note that there is only a single value $\bar{d}_k$ in the table $\mathtt{Coef}_k$ with key β_k/α_k, namely $\bar{d}_k = \mathtt{Coef}_k[\beta_k/\alpha_k]$.

Consequently, we have

$$\gamma + c = \sum_k \eta_k \bar{d}_k = \sum_{k \in [Q_{MS}], \alpha_k \neq 0} \alpha_k \cdot \mathtt{Coef}_k[\beta_k/\alpha_k],$$

where the first equation follows from Equation (8). Thus, the second condition in Claim 6.2 also holds.

Proof (Proof of Claim 6.2). For each H_c query, we will bound the probability that it is bad at the end of the game in two steps. First, we bound the probability that it is bad when it is queried. Second, if it was originally good when being queried, we bound the probability that it turns bad later.

Fix any query $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$. Let $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ be $\mathtt{r\bar{e}p}(R)$, and c be the returned value. By Claim 6.2, $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is bad only if

- For every $k \in [Q_{MS}]$, either $\alpha_k = \beta_k = 0$, or $\alpha_k \neq 0$ and $\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- It further holds that
$$\sum_{k \in [Q_{MS}], \alpha_k \neq 0} \alpha_k \cdot \mathtt{Coef}_k[\beta_k/\alpha_k] = \gamma + c.$$

Consider that the query $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is fixed, and the returned value c is to be chosen uniformly. Then $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ and $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)\}$ have been fixed. For the latter, note that if this H_c query is artificial, then some key-value pair may be inserted into $\mathtt{Coef}_k$ meanwhile, and the value of this pair depends on c. However, this pair is exactly the excluded pair $\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)$ by definition, and so $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)\}$ does not depend on c.

Assume that the first condition holds. Then we have $\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$, and hence the value of $\mathtt{Coef}_k[\beta_k/\alpha_k]$ does not depend on c. With c to be uniformly chosen, the second condition holds with probability $1/p$. It follows that every H_c query is bad with probability at most $1/p$ when being queried.

Again by Claim 6.2, conversely, $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ is not bad only if at least one of the following holds:

- There exists $k \in [Q_{MS}]$ such that $\alpha_k = 0$ and $\beta_k \neq 0$.
- There exists $k \in [Q_{MS}]$ such that $\alpha_k \neq 0$ and $\beta_k/\alpha_k \notin \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- For every $k \in [Q_{MS}]$, $\alpha_k \neq 0$ implies $\beta_k/\alpha_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m).\mathtt{Key}\}$. Meanwhile, it holds that
$$\sum_{k \in [Q_{MS}], \alpha_k \neq 0} \alpha_k \cdot \mathtt{Coef}_k[\beta_k/\alpha_k] \neq \gamma + c.$$

Suppose that $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was not bad when it was queried. After that, only the second condition can change from true to false due to more key-value pairs inserted into $\mathtt{Coef}_k$, and hence into $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\hat{\mathsf{pk}}, R, m)\}$. We analyze the probability of turning the second condition from true to false (and thus possibly turns the query bad). By definition, $\alpha_k \neq 0$ implies that the k-th signing session has already started (before $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ was queried). In particular, $\mathtt{Coef}_k$ has already been initialized. For table maintenance, however, the keys inserted into $\mathtt{Coef}_k$ are the uniformly random values returned by H_b queries. Thus, a newly inserted key matches the fixed value β_k/α_k with probability $1/p$. Overall, $\mathsf{H}_c(\hat{\mathsf{pk}}, R, m)$ becomes bad with probability at most $(Q_{MS} + Q_{\mathsf{H}_b})/p$.

Combining the above two steps, we know that each H_c query is bad at the end of the game with probability at most $(Q_{MS} + Q_{\mathsf{H}_b} + 1)/p < Q/p$. By the union bound, there exists a bad query at the end of the game with probability at most Q^2/p.

6.3 Proof of Lemma 6 (the case $b^* = 1$)

Proof (Proof of Lemma 6). Reduction $\mathcal{B}_1$ follows the structure described in Section 6.3. It simulates $\mathbf{G}_7$ until the game returns. If $\mathcal{A}$ wins with $b^* = 1$, then (R^*, s^*) is a valid signature on m^* under aggregated key $\tilde{\mathsf{pk}}^* := \mathsf{KeyAggr}(\mathcal{P}^*)$, and the rerandomized key $\hat{\mathsf{pk}} := \mathsf{RandPK}(\mathsf{pk}, \rho^*)$ is contained in $\mathcal{P}^*$. Namely, it holds that $g^{s^*} = R^*(\tilde{\mathsf{pk}}^*)^{c^*}$, where $c^* := \mathsf{H}_c(\tilde{\mathsf{pk}}^*, R^*, m^*)$. Also, it holds that $\tilde{\mathsf{pk}}^* = \prod_{i \in [n]} \mathsf{pk}_i^{a_i}$, where $a_i := \mathsf{H}_a(\langle \mathcal{P}^* \rangle, \mathsf{pk}_i)$, and there exists $\mathsf{pk}_j \in \mathcal{P}$ such that $\mathsf{pk}_j = \hat{\mathsf{pk}} := X g^{\rho^*}$. Suppose that pk_j occurs n_j times in $\mathcal{P}^*$. Then we have

$$\tilde{\mathsf{pk}}^* = (X g^{\rho^*})^{n_j a_j} \prod_{i : \mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}^*} \mathsf{pk}_i^{a_i}. \tag{9}$$

By the validity of the forged multi-signature, $\mathcal{B}_1$ learns the discrete logarithm s^* of

$$R^*(\tilde{\mathsf{pk}}^*)^{c^*} = R^*((Xg^{\rho^*})^{n_j a_j} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}^*} \mathsf{pk}_i^{a_i})^{c^*}.$$

Consequently, it learns the value of

$$\bar{\mathtt{rep}}(R^*((Xg^{\rho^*})^{n_j a_j} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}^*} \mathsf{pk}_i^{a_i})^{c^*}) \cdot \mathbf{x},$$

where $\mathbf{x} = (r_{1,1,1}, r_{1,1,2}, \ldots, r_{Q_{MS},1,1}, r_{Q_{MS},1,2}, x)$ as defined in Section 6.3. If

$$\begin{aligned}\bar{\mathtt{rep}}(R^*((Xg^{\rho^*})^{n_j a_j} \prod_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}^*} \mathsf{pk}_i^{a_i})^{c^*}) = \bar{\mathtt{rep}}(R^*) + c^*(\underbrace{\bar{\mathtt{rep}}(X^{n_j a_j}) + \sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}^*} a_i \cdot \bar{\mathtt{rep}}(\mathsf{pk}_i)}_{=:\mathbf{v}_{\mathcal{P}^*,j}}) \\ \notin \mathsf{Span}\{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]},\end{aligned} \tag{10}$$

that is, if $\bar{\mathtt{rep}}(R^*) + c^* \mathbf{v}_{\mathcal{P}^*,j}$ is not in the row span of matrix $\mathbf{M}$ defined in Section 6.3, then $\mathcal{B}_1$ can solve its AOMDL instance. Note that here we abuse the notation, and the definition of vector $\mathbf{v}_{\mathcal{P}^*,j}$ is slightly different from that in Section 6.1.

We will bound the probability that Equation (10) does not hold. To do so, we define a bad event $\mathsf{BadHc2}$, relate it with the event that Equation (10) does not hold, and bound the probability of $\mathsf{BadHc2}$. In order to analyze the probability of $\mathsf{BadHc2}$, we define another event $\mathsf{BadP2}$ and bound the probabilities of $\mathsf{BadP2}$ and $\neg\mathsf{BadP2} \wedge \mathsf{BadHc2}$ separately.

Definition of $\mathsf{BadP2}$**.** Let $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ be a non-trivial public key multiset that has been queried to H_a or $\mathtt{SIGN}_1$, so $a_i := \mathsf{H}_a(\langle \mathcal{P} \rangle, \mathsf{pk}_i)$ for every $i \in [n]$ has been defined. We say $\mathcal{P}$ is bad *with respect to* $\mathsf{pk}_j \in \mathcal{P}$ if there exists

$$\mathbf{v}_k \in \{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{\bar{b}_k} X^{\bar{d}_k})\}_{(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k} \cup \{\mathbf{0}\} \quad \text{for} \quad k \in [Q_{MS}]$$

such that

$$\mathbf{v}_{\mathcal{P},j} := \bar{\mathtt{rep}}(X^{n_j a_j}) + \sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} a_i \cdot \bar{\mathtt{rep}}(\mathsf{pk}_i) \in \mathsf{Span}\{\mathbf{v}_k\}_{k \in [Q_{MS}]}, \tag{11}$$

where $a_i := \mathsf{H}_a(\mathcal{P}, \mathsf{pk}_i)$ for $i \in [n]$.

We say $\mathcal{P}$ is bad if there exists pk_j such that $\mathcal{P}$ is bad with respect to pk_j. Note that a good (i.e. not bad) $\mathcal{P}$ may turn bad due to more key-value pairs being inserted into $\mathtt{Coef}_k$. We say that $\mathsf{BadP2}$ occurs if, *whenever the game returns* (and $b^* = 1$), there exists a bad $\mathcal{P}$.

Excluded Pair. In preparation for the definition of $\mathsf{BadHc2}$, we define excluded key-value pairs in the same way as in Section 6.2. For any query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$, let the returned value be c. We define the "excluded key-value pair" $\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)$ for $k \in [Q_{MS}]$ with respect to this query as follows.

If $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ was not an artificial query (introduced in $\mathbf{G}_1$), then $\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m) = \perp$ for every $k \in [Q_{MS}]$. Consider the case that $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ was such an artificial query made in response to a H_b query. In this case, the H_b query is in the form of $b = \mathsf{H}_b(\tilde{\mathsf{pk}}, R_1, R_2, m)$ for some b, R_1, R_2. As introduced in $\mathbf{G}_6$, when the H_b query was made, the game executed procedure $\mathtt{CheckAndInsert}(k, (\tilde{\mathsf{pk}}, R_1, R_2, m), b)$ for every opened signing session k. If the check passed, then some key-value pair $(b, \bar{d}'_k)$ was inserted into $\mathtt{Coef}_k$, where b is the returned value of this H_b query. If the check did not pass, we define $\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m) = \perp$. If the check passed, define $\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m) = (b, \bar{d}'_k)$ as the pair inserted into $\mathtt{Coef}_k$.

Definition of $\mathsf{BadHc2}$**.** For any $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$, let the returned value be c. We say this query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is *bad with respect to* pk_j if both of the following conditions hold:

1. $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) \neq \perp$, and $\mathsf{pk}_j \in \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$. Let $\mathcal{P} := \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$.
2. There exists
$$\mathbf{v}_k \in \{\bar{\mathtt{rep}}(R_{k,1,1} R_{k,1,2}^{\bar{b}_k} X^{\bar{d}_k})\}_{(\bar{b}_k, \bar{d}_k) \in \mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)\}} \cup \{\mathbf{0}\}$$
for $k \in [Q_{MS}]$ such that
$$\bar{\mathtt{rep}}(R) + c\mathbf{v}_{\mathcal{P},j} \in \mathsf{Span}\{\mathbf{v}_k\}_{k \in [Q_{MS}]}. \tag{12}$$

We say query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is bad[13] if there exists pk_j such that $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is bad with respect to pk_j. Note that a good query may turn bad later due to more key-value pairs being inserted into $\mathtt{Coef}_k$. We say that BadHc2 occurs if, *whenever the game returns* (and $b^* = 1$), there exists a bad query.

The proof can be completed by Claims 6.3–6.3, since

$$\begin{aligned}\Pr[\mathsf{BadHc2}] &= \Pr[\mathsf{BadHc2} \wedge \mathsf{BadP2}] + \Pr[\mathsf{BadHc2} \wedge \neg\mathsf{BadP2}] \\ &\leq \Pr[\mathsf{BadP2}] + \Pr[\mathsf{BadHc2} \mid \neg\mathsf{BadP2}].\end{aligned}$$

Claim. If $\mathbf{G}_7$ returns (and $b^* = 1$) and Equation (10) does not hold, then BadHc2 occurs.

Claim. It holds that $\Pr[\mathsf{BadP2}] \leq NQ/p$.

Claim. It holds that $\Pr[\mathsf{BadHc2} \mid \neg\mathsf{BadP2}] \leq (N+1)Q^2/p$.

Proof (Proof of Claim 6.3). Suppose that Equation (10) does not hold, namely

$$\mathtt{r\bar{e}p}(R^*) + c^*\mathbf{v}_{\mathcal{P}^*,j} \in \mathsf{Span}\{\mathtt{r\bar{e}p}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k})\}_{k\in[Q_{MS}]}.$$

We show that $c^* = \mathsf{H}_c(\tilde{\mathsf{pk}}^*, R^*, m^*)$ is bad with respect to $\mathsf{pk}_j = Xg^{\rho^*}$; in particular, BadHc2 occurs.

By the winning condition, we have $\tilde{\mathsf{pk}}^* = \mathsf{KeyAggr}(\mathcal{P}^*)$, and $\mathsf{pk}_j \in \mathcal{P}^*$. By the separation between single (rerandomized) keys and aggregated keys (see $\mathbf{G}_7$), the oracle $\mathtt{SSIGN}$ never signs under $\tilde{\mathsf{pk}}^*$. Hence, the hash value $\mathsf{H}_c(\tilde{\mathsf{pk}}^*, R^*, m^*)$ cannot have been programmed during the simulation of signing queries and corresponds to an actual query. Also, Condition 1 of BadHc2 holds.

We next show that Condition 2 of BadHc2 holds. To see this, let $\mathbf{v}_k = \mathtt{r\bar{e}p}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k})$ for every $k \in [Q_{MS}]$. By Lemma 3, we have $(b_k, a_{k,1}c_k) \in \mathtt{Coef}_k$ for $k \in [Q_{MS}]$. It remains to show that $(b_k, a_{k,1}c_k)$ is not the excluded pair $\mathsf{Excl}_k(\tilde{\mathsf{pk}}^*, R^*, m^*)$.

Toward a contradiction, we assume that $(b_k, a_{k,1}c_k) = \mathtt{Excl}_k(\tilde{\mathsf{pk}}^*, R^*, m^*)$. Recall that this implies that $\mathsf{H}_c(\tilde{\mathsf{pk}}^*, R^*, m^*)$ is an artificial query (introduced in $\mathbf{G}_1$) made in response to a H_b query $\mathsf{H}_b(\tilde{\mathsf{pk}}^*, R_1, R_2, m^*)$ for some R_1, R_2, and b_k is the answer to this H_b query. Since we have ruled out H_b collisions in $\mathbf{G}_4$, this H_b query is the unique one that returns b_k. Furthermore, since we have ruled out KeyAggr collisions for non-trivial public key multisets in $\mathbf{G}_5$, $\mathcal{P}^*$ is the unique public key multiset such that $\mathsf{KeyAggr}(\mathcal{P}^*) = \tilde{\mathsf{pk}}^*$. Therefore, the fact that $b_k = \mathsf{H}_b(\tilde{\mathsf{pk}}^*, R_1, R_2, m^*)$ means that in the k-th multi-signature session, oracle $\mathtt{SIGN}_1$ was queried to sign m^* under public key multiset $\mathcal{P}^*$.[14] Consequently, the tuple $(\mathcal{P}^*, m^*)$ should have been inserted into Q_1, which contradicts the winning requirement that $(\mathcal{P}^*, m^*) \notin Q_1$. This concludes our proof of Claim 6.3.

Probability of BadP2. The proof of Claim 6.3 follows exactly the same reasoning as the proof of Claim 6.1. We therefore omit the proof. We still state below an analogue of Claim 6.1, which specifies the conditions under which a public key multiset is bad. This analogue will also be used in the proof of Claim 6.3.

Claim. Fix any $\mathcal{P}$ and $\mathsf{pk}_j \in \mathcal{P}$. Suppose that pk_j occurs in $\mathcal{P}$ for n_j times. Let $(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'')$ be $\sum_{i:\mathsf{pk}_i \neq \mathsf{pk}_j \in \mathcal{P}} a_i \cdot \mathtt{r\bar{e}p}(\mathsf{pk}_i)$. Then $\mathcal{P}$ is bad with respect to $\mathsf{pk}_j \in \mathcal{P}$ if and only if both of the following two conditions hold:

- For every $k \in [Q_{MS}]$, either $\alpha''_k = \beta''_k = 0$, or $\alpha''_k \neq 0$ and $\beta''_k/\alpha''_k \in \mathtt{Coef}_k.\mathtt{Keys}$.
- It further holds that $\sum_{k\in[Q_{MS}],\alpha''_k\neq 0} \alpha''_k \cdot \mathtt{Coef}_k[\beta''_k/\alpha''_k] = \gamma'' + n_j a_j$.

Proof (Proof of Claim 6.3). Analogous to the proof of Claim 6.1.

Proof (Proof of Claim 6.3). Analogous to the proof of Claim 6.1.

Probability of BadHc2. In the following claim, we take a closer look at the meaning of a query being bad.

[13] Again, we stress that only (possibly artificial) *queries* to H_c can be classified as bad, and not hash values that are programmed by the reduction in the absence of any corresponding external query.

[14] Recall that b_k is, by definition, the H_b-coefficient used in the second round of the k-th multi-signature signing session.

Claim. Fix any query $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ and pk_j. Suppose that Condition 1 of this query being bad with respect to pk_j holds, that is, $\mathcal{P} = \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) \neq \bot$, and $\mathsf{pk}_j \in \mathcal{P}$. Suppose that pk_j occurs in $\mathcal{P}$ for n_j times. Let $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ be $\mathtt{r\bar{e}p}(R)$ and $(\alpha'_1, \beta'_1, \ldots, \alpha'_{Q_{MS}}, \beta'_{Q_{MS}}, \gamma')$ be $\mathbf{v}_{\mathcal{P},j}$ (recall Equation 11 for the notation).[15] Then $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is bad with respect to pk_j (i.e., Condition 2 of this query being bad with respect to pk_j holds) if and only if both of the following two conditions hold:

- For every $k \in [Q_{MS}]$, either $\alpha_k + c\alpha'_k = \beta_k + c\beta'_k = 0$, or $\alpha_k + c\alpha'_k \neq 0$ and $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k) \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- It further holds that

$$\sum_{k \in [Q_{MS}], \alpha_k + c\alpha'_k \neq 0} (\alpha_k + c\alpha'_k) \cdot \mathtt{Coef}_k[(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)] = \gamma + c\gamma'.$$

Proof (Proof of Claim 6.3). Analogous to the proof of Claim 6.2.

Proof (Proof of Claim 6.3). For each H_c query, we will bound the probability that it is bad at the end of the game in two steps. First, we bound the probability that it is bad when it is queried. Second, if it was originally good when being queried, we bound the probability that it turns bad later. Recall that this claim bounds the probability of BadHc2 conditioned on ¬BadP2. In the first step, our analysis will be based on the assumption that ¬BadP2 happens.

Originally bad queries. Fix a query $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$. Assume that Condition 1 of $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ being bad with respect to pk_j holds. That is, we have $\mathcal{P} = \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) \neq \bot$ and $\mathsf{pk}_j \in \mathcal{P}$. We bound the probability of Condition 2. Let $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ be $\mathtt{r\bar{e}p}(R)$ and $(\alpha'_1, \beta'_1, \ldots, \alpha'_{Q_{MS}}, \beta'_{Q_{MS}}, \gamma')$ be $\mathbf{v}_{\mathcal{P},j}$. By Claim 6.3, Condition 2 holds only if both of the following conditions hold:

- (i) For every $k \in [Q_{MS}]$, either $\alpha_k + c\alpha'_k = \beta_k + c\beta'_k = 0$, or $\alpha_k + c\alpha'_k \neq 0$ and $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k) \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- (ii) It further holds that

$$\sum_{k \in [Q_{MS}], \alpha_k + c\alpha'_k \neq 0} (\alpha_k + c\alpha'_k) \cdot \mathtt{Coef}_k[(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)] = \gamma + c\gamma'.$$

We consider the following two cases:

- Case 1: There exists $k \in [Q_{MS}]$ such that (α_k, β_k) and (α'_k, β'_k) are linearly independent.
- Case 2: For every $k \in [Q_{MS}]$, (α_k, β_k) and (α'_k, β'_k) are linearly dependent.

In Case 1, we bound the probability that Condition (i) holds. In Case 2, we bound the probability that Condition (ii) holds assuming Condition (i).

When analyzing the probabilities, we consider that the query $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is fixed, and the returned value c is to be chosen uniformly. Then $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$, $(\alpha'_1, \beta'_1, \ldots, \alpha'_{Q_{MS}}, \beta'_{Q_{MS}}, \gamma')$, and $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)\}$ for every $k \in [Q_{MS}]$ have been fixed. For the latter, note that if this H_c query is artificial (introduced in $\mathbf{G}_1$), then some key-value pair may be inserted into $\mathtt{Coef}_k$ at the same time, and the value of this pair depends on c. However, this pair is exactly the excluded pair $\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)$ by definition, so $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)\}$ does not depend on c and has been fixed.

Case 1. Condition (i) requires that $(\alpha_k, \beta_k) + c(\alpha'_k, \beta'_k) = (0, 0)$ or $(1, (\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)) = (1, \bar{b}_k)$ for some $\bar{b}_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. This can be rewritten into the requirement that $(\alpha_k, \beta_k) + c(\alpha'_k, \beta'_k) \in \mathsf{Span}\{(1, \bar{b}_k)\}$ for some $\bar{b}_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. Now take $k \in [Q_{MS}]$ such that (α_k, β_k) and (α'_k, β'_k) are linearly independent, which exists by assumption. Further, fix a $\bar{b}_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. If $(\alpha_k, \beta_k) + c(\alpha'_k, \beta'_k) \in \mathsf{Span}\{(1, \bar{b}_k)\}$, then for any $c' \neq c \in \mathbb{Z}_p$, it follows that $(\alpha_k, \beta_k) + c'(\alpha'_k, \beta'_k) \notin \mathsf{Span}\{(1, \bar{b}_k)\}$. Otherwise, we could derive a linear dependence of (α_k, β_k) and (α'_k, β'_k), contradicting our choice of k. Therefore, for this specific $\bar{b}_k$, with $c \leftarrow_\$ \mathbb{Z}_p$ chosen uniformly at random (and $\bar{b}_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$ having been fixed), the probability that $(\alpha_k, \beta_k) + c(\alpha'_k, \beta'_k) \in \mathsf{Span}\{(1, \bar{b}_k)\}$ is at most $1/p$. By the union bound, Condition (i) thus holds with probability at most Q/p.

Case 2. We first rule out the following bad events:

[15] From the definition of $\mathbf{v}_{\mathcal{P},j}$ it directly follows that $(\alpha'_1, \beta'_1, \ldots, \alpha'_{Q_{MS}}, \beta'_{Q_{MS}}, \gamma') = (\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'' + n_j a_j)$, where $(\alpha''_1, \beta''_1, \ldots, \alpha''_{Q_{MS}}, \beta''_{Q_{MS}}, \gamma'')$ is defined as in Claim 6.3 above.

- There exists $k \in [Q_{MS}]$ such that, $\alpha_k \neq 0$ or $\alpha'_k \neq 0$, while $\alpha_k + c\alpha'_k = 0$.
- It holds that $\sum_{k\in[Q_{MS}],\alpha'_k\neq 0} \alpha'_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] = \gamma'$.

We bound the probability of the first bad event. Fix any $k \in [Q_{MS}]$. If $\alpha'_k \neq 0$, then $\alpha_k + c\alpha'_k = 0$ with probability at most $1/p$. If $\alpha'_k = 0$ and $\alpha_k \neq 0$, then $\alpha_k + c\alpha'_k \neq 0$. Therefore, the probability that $\alpha_k + c\alpha'_k = 0$ is at most $1/p$. By the union bound, the bad event occurs with probability at most Q_{MS}/p.

We relate the second bad event to $\mathsf{BadP2}$. We show that, here in Case 2, if the second bad event occurs, and Condition (i) holds, then $\mathcal{P}$ is bad with respect to pk_j, contradicting the assumption of $\neg\mathsf{BadP2}$.

Condition (i) requires $(\alpha_k, \beta_k) + c(\alpha'_k, \beta'_k) \in \mathsf{Span}\{(1, \bar{b}_k)\}$ for some $\bar{b}_k \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. Since (α_k, β_k) and (α'_k, β'_k) are linearly dependent, we have $(\alpha'_k, \beta'_k) \in \mathsf{Span}\{(1, \bar{b}_k)\}$. Thus, the first condition of Claim 6.3 holds (which specifies the two conditions for $\mathcal{P}$ being bad with respect to pk_j). The second condition of Claim 6.3 is exactly $\sum_{k\in[Q_{MS}],\alpha'_k\neq 0} \alpha'_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] = \gamma'$. Therefore, by Claim 6.3, $\mathsf{BadP2}$ occurs, a contradiction.

In the following, we assume that both of these bad events do not occur. Define $\mathcal{K} := \{k \in [Q_{MS}] : \alpha_k \neq 0 \vee \alpha'_k \neq 0\}$ (i.e., $\mathcal{K}$ is the complement of $\{k \in [Q_{MS}] : \alpha_k = \alpha'_k = 0\}$). Since the first bad event does not occur, we have $\alpha_k + c\alpha'_k \neq 0$ if and only if $k \in \mathcal{K}$. For $k \in \mathcal{K}$, since $(\alpha_k, \alpha'_k) \neq (0, 0)$ and (α_k, β_k) and (α'_k, β'_k) are linearly dependent (by assumption in Case 2), there exists $\eta_k \in \mathbb{Z}_p$ such that $(\alpha_k, \beta_k) = \eta_k(\alpha'_k, \beta'_k)$. This implies that

$$\frac{\beta_k + c\beta'_k}{\alpha_k + c\alpha'_k} = \frac{\eta_k\beta'_k + c\beta'_k}{\eta_k\alpha'_k + c\alpha'_k} = \frac{\beta'_k(\eta_k c + 1)}{\alpha'_k(\eta_k c + 1)} = \frac{\beta'_k}{\alpha'_k},$$

which is independent of c. Condition (ii) is thus equivalent to

$$\sum_{k\in\mathcal{K}} (\alpha_k + c\alpha'_k) \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] = \gamma + c\gamma',$$

which is equivalent to

$$\sum_{k\in\mathcal{K}} \alpha_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] = \gamma + c\left(\gamma' - \sum_{k\in\mathcal{K}} \alpha'_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k]\right). \tag{13}$$

Since the second bad event does not occur, we have

$$\sum_{k\in\mathcal{K}} \alpha'_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] = \sum_{k\in[Q_{MS}],\alpha'_k\neq 0} \alpha'_k \cdot \mathtt{Coef}_k[\beta'_k/\alpha'_k] \neq \gamma'.$$

Therefore, Equation (13) holds for exactly one $c \in \mathbb{Z}_p$, so Condition (ii) holds with probability $1/p$.

In conclusion, given $\neg\mathsf{BadP2}$, $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is bad with respect to any $\mathsf{pk}_j \in \mathcal{P} = \mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}})$ with probability at most Q/p. By the union bound, $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ is bad at the time it is queried with probability at most NQ/p.

Good queries turn bad. Fix a query $c = \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$. Suppose that it was good when it is answered. First, consider the case that $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) = \bot$. By the change in $\mathbf{G}_5$, for any non-trivial $\mathcal{P}$, $\mathsf{KeyAggr}(\mathcal{P})$ that is determined later than this H_c query cannot hit $\tilde{\mathsf{pk}}$. Therefore, this query cannot turn bad.

Therefore, it suffices to consider the case that $\mathsf{KeyAggr}^{-1}(\tilde{\mathsf{pk}}) = \mathcal{P} \neq \bot$. We fix any $\mathsf{pk}_j \in \mathcal{P}$, and analyze the probability that Condition 2 of the query being bad with respect to pk_j turns from false to true.

Let $(\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma)$ be $\mathtt{r\bar{e}p}(R)$ and $(\alpha'_1, \beta'_1, \ldots, \alpha'_{Q_{MS}}, \beta'_{Q_{MS}}, \gamma')$ be $\mathbf{v}_{\mathcal{P},j}$. By Claim 6.3, Condition 2 does not hold only if at least one of the following conditions holds:

- There exists $k \in [Q_{MS}]$ such that $\alpha_k + c\alpha'_k = 0$ and $\beta_k + c\beta'_k \neq 0$.
- There exists $k \in [Q_{MS}]$ such that $\alpha_k + c\alpha'_k \neq 0$ and $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k) \notin \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$.
- For every $k \in [Q_{MS}]$, $\alpha_k + c\alpha'_k \neq 0$ implies $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k) \in \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. Meanwhile, it holds that $\sum_{k\in[Q_{MS}],\alpha_k+c\alpha'_k\neq 0} (\alpha_k + c\alpha'_k) \cdot \mathtt{Coef}_k[(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)] \neq \gamma + c\gamma'$.

Suppose that, when the H_c query is answered, one of these conditions hold. After that, only the second condition can change from true to false due to more key-value pairs inserted into $\mathtt{Coef}_k$, and hence into $\mathtt{Coef}_k \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m)\}$. This is because we have ruled out H_b collisions in $\mathbf{G}_4$, so no repeated keys can be inserted into $\mathtt{Coef}_k$ (also see Lemma 2), and thus the third conditions cannot turn from true to false.

Therefore, to bound the probability that a good query turns bad, it suffices to analyze the probability that the second condition turns false.

Suppose that the second condition holds when the H_c query was answered. Then there exists a certain k such that $\alpha_k + c\alpha'_k \neq 0$ and $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k) \notin \mathtt{Coef}_k.\mathtt{Keys} \setminus \{\mathtt{Excl}_k(\tilde{\mathsf{pk}}, R, m).\mathtt{Key}\}$. The second condition turns false only if a new key inserted into $\mathtt{Coef}_k$ hits the fixed value of $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)$ for this specific k. Note that $\alpha_k + c\alpha'_k \neq 0$ implies $(\alpha_k, \alpha'_k) \neq (0, 0)$, which means that the k-th signing session has already started (before $\mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ was queried) and $\mathtt{Coef}_k$ has already been initialized. For table maintenance, however, the keys inserted into $\mathtt{Coef}_k$ are the uniformly random values returned by H_b queries. Thus, a newly inserted key hits the fixed value $(\beta_k + c\beta'_k)/(\alpha_k + c\alpha'_k)$ with probability $1/p$. By the union bound, a good query ever turns bad with probability at most Q/p.

Putting it all together. Combining the probabilities that a query 1) is originally bad, and 2) was originally good but turns bad later, we can conclude that each H_c query is bad at the end of the game with probability at most $(N+1)Q/p$. By the union bound, we have $\Pr[\mathsf{BadHc2} \mid \neg\mathsf{BadP2}] \leq (N+1)Q^2/p$. This concludes our proof of Claim 6.3.

Algebraic Representations In the AGM, whenever the adversary $\mathcal{A}$ outputs a group element in its oracle queries or the final forgery, it is required to also output an algebraic representation of the output group element with respect to the group elements it has received. These include g, the honest public key X, nonces $\bar{R}_1, \ldots, \bar{R}_{Q_S}$ received from $\mathtt{SSIGN}$, and nonces $R_{1,1,1}, R_{1,1,2}, \ldots, R_{Q_{MS},1,1}, R_{Q_{MS},1,2}$ received from $\mathtt{SIGN}_0$. Concretely, $\mathcal{A}$ outputs $\bar{\alpha}_1, \ldots, \bar{\alpha}_{Q_S}, \alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma, \delta$, such that the output element is equal to

$$g^{\delta} X^{\gamma} \prod_{j \in [Q_S]} \bar{R}_j^{\bar{\alpha}_j} \prod_{k \in [Q_{MS}]} (R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k}).$$

If $\mathcal{A}$ has not made the j-th $\mathtt{SIGN}$ or the k-th $\mathtt{SIGN}_0$ query when this group element is output, we view $\bar{\alpha}_j$ and α_k, β_k as 0. If $\mathcal{A}$ outputs the same group element for multiple times throughout the game, we assume that it always provides a consistent algebraic representation for the element. This is without loss of generality, since we can build a wrapper that always reuses the first representation for every element being output multiple times.

Reduced Representations. Recall that $(\bar{R}_j, \bar{s}_j)$ is a $\mathsf{Schnorr}$ signature under the rerandomized public key $\bar{X}_j = X g^{\bar{\rho}_j}$. Namely, it holds that $g^{\bar{s}_j} = \bar{R}_j (X g^{\bar{\rho}_j})^{\bar{c}_j}$. Therefore, by substituting these relations into the above algebraic representation, the game can obtain a representation $\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma', \delta'$ with respect to $R_{1,1,1}, R_{1,1,2}, \ldots, R_{Q_{MS},1,1}, R_{Q_{MS},1,2}, X, g$, such that the output group element is equal to

$$g^{\delta'} X^{\gamma'} \prod_{k \in [Q_{MS}]} R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k},$$

where $\gamma' := \gamma - \sum_{j \in [Q_S]} \bar{\alpha}_j \bar{c}_j$, $\delta' := \delta + \sum_{j \in [Q_S]} \bar{\alpha}_j (\bar{s}_j - \bar{c}_j \bar{\rho}_j)$. We call it the *reduced representation*. We consider $\mathcal{A}$ to be directly outputting this reduced representation.

Rep-Notation. We introduce some notations about reduced algebraic representations. Let

$$Y = \left(\prod_{k \in [Q_{MS}]} R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k} \right) \cdot X^{\gamma} g^{\delta} \cdot \prod_{j \in [l]} Z_j^{\eta_j} \tag{14}$$

be a linear combination of $R_{1,1,1}, R_{1,1,2}, \ldots, R_{Q_{MS},1,1}, R_{Q_{MS},1,2}, X, g$, and some group elements $Z_1, \ldots, Z_l$ that $\mathcal{A}$ has output. We let $\mathtt{rep}(Y)$ denote the reduced algebraic representation of Y with respect to $R_{1,1,1}, R_{1,1,2}, \ldots, R_{Q_{MS},1,1}, R_{Q_{MS},1,2}, X, g$, derived by substituting the reduced representations of $Z_1, \ldots, Z_l$ that $\mathcal{A}$ provides into Equation (14). All group elements involved in this proof can be written in the format of Y, so a representation $\mathtt{rep}(Y)$ can always be derived. Further, we let $\mathtt{r\bar{e}p}(Y)$ be $\mathtt{rep}(Y)$ with the δ-component dropped. That is,

$$\mathtt{r\bar{e}p}(g^{\delta} X^{\gamma} \prod_{k \in [Q_{MS}]} (R_{k,1,1}^{\alpha_k} R_{k,1,2}^{\beta_k})) = (\alpha_1, \beta_1, \ldots, \alpha_{Q_{MS}}, \beta_{Q_{MS}}, \gamma).$$

This notation is particularly convenient for expressing sparse algebraic representations such as

$$\mathtt{rep}(R_{1,1,1} R_{1,1,2}^{b_k} X^{a_{k,1} c_k}) = (1, b_k, 0, 0, \ldots, 0, 0, a_{k,1} c_k, 0).$$

We note that the reduced representations are always row vectors.

Let $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ be a multi-set of public keys such that $a_i := \mathsf{H}_a(\langle\mathcal{P}\rangle, \mathsf{pk}_i)$ for every $i \in [n]$ has already been queried. $\mathcal{A}$ provides $\mathtt{rep}(\mathsf{pk}_i) = (\alpha_{i,1}, \beta_{i,1}, \ldots, \alpha_{i,Q_{MS}}, \beta_{i,Q_{MS}}, \gamma_i, \delta_i)$ for every $\mathsf{pk}_i \in \mathcal{P}$. Then for $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P}) = \prod_{i\in[n]} \mathsf{pk}_i^{a_i}$, we have

$$\mathtt{rep}(\tilde{\mathsf{pk}}) = \sum_{i\in[n]} a_i \cdot \mathtt{rep}(\mathsf{pk}_i) = \sum_{i\in[n]} a_i \cdot (\alpha_{i,1}, \beta_{i,1}, \ldots, \alpha_{i,Q_{MS}}, \beta_{i,Q_{MS}}, \gamma_i, \delta_i).$$

Structure of All of Our Reductions All our reductions $\mathcal{C}$, $\mathcal{D}$, $\mathcal{B}_0$, and $\mathcal{B}_1$ share the same main structure. The reduction takes $(\mathbb{G}, p, g)$ and $2Q_{MS}+1$ group elements given by the AOMDL instance as inputs. It simulates the respective hybrid game for $\mathcal{A}$ with $(\mathbb{G}, p, g)$ as the public parameters. It uses the $2Q_{MS}+1$ AOMDL challenge elements as the target public key X and the nonces $R_{1,1,1}$, $R_{1,1,2}$, …, $R_{Q_{MS},1,1}$, $R_{Q_{MS},1,2}$ output by $\mathtt{SIGN}_0$. The secret key x and randomness $r_{1,1,1}$, $r_{1,1,2}$, …, $r_{Q_{MS},1,1}$, $r_{Q_{MS},1,2}$ are the unknown discrete logarithms that the reduction aims to output at the end.

Recall that the game simulates the single-signer oracle $\mathtt{SSIGN}$ using honest-verifier zero-knowledge (HVZK) simulation for $\mathsf{Schnorr}$ (see $\mathbf{G}_2$). Further, recall that the game returns $s_{k,1} := r_{k,1,1} + b_k r_{k,1,2} + a_{k,1}c_k(x + \rho_k)$ in $\mathtt{SIGN}_1(k, \cdot)$ for session k. The reduction uses its $\mathtt{DLOG}$ oracle to simulate $\mathtt{SIGN}_1$ without knowing x, $r_{k,1,1}$, and $r_{k,1,2}$. Specifically, it returns $s_{k,1} := \mathtt{DLOG}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k}) + a_{k,1}c_k\rho_k$.

Reductions $\mathcal{B}_0$ and $\mathcal{B}_1$ simulate $\mathbf{G}_7$ until the game returns. By the assumption we made in $\mathbf{G}_0$, $\mathcal{A}$ has completed all Q_{MS} multi-signature sessions at the end of the game. Reductions $\mathcal{C}$ and $\mathcal{D}$ simulate $\mathbf{G}_3$ and $\mathbf{G}_7$, respectively, until the abort condition introduced in the game is triggered. $\mathcal{A}$ may not have completed every signing session so far. We therefore let the reductions themselves make the remaining $\mathtt{SIGN}_0$ and $\mathtt{SIGN}_1$ queries, adhering to the same conditions prescribed by the assumption made in $\mathbf{G}_0$. That is, a signing session k is closed by querying $\mathtt{SIGN}_1(k, \mathcal{P}_k, m_k, \tilde{\mathsf{pm}}_1)$ for some $(\mathcal{P}_k, m_k, \tilde{\mathsf{pm}}_1)$ such that $(\mathcal{P}_k, m_k) \neq (\mathcal{P}^*, m^*)$, thereby completing all Q_{MS} signing sessions. Consequently, when the simulation terminates, the reduction has made exactly Q_{MS} $\mathtt{DLOG}$ queries, each for answering one $\mathtt{SIGN}_1$ query.

From every query $\mathtt{DLOG}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k})$, the reduction has learned a linear equation of the unknown discrete logarithms x, $r_{1,1,1}$, $r_{1,1,2}$, …, $r_{Q_{MS},1,1}$, $r_{Q_{MS},1,2}$. Concretely, let

$$\mathbf{x} := (r_{1,1,1}, r_{1,1,2}, \ldots, r_{Q_{MS},1,1}, r_{Q_{MS},1,2}, x)^T \in \mathbb{Z}_p^{2Q_{MS}+1} \tag{15}$$

be the vector consisting of the unknown discrete logarithms that the reduction needs to solve for. From the Q_{MS} queries to $\mathtt{DLOG}$ that the reduction has made, it has learned $\bar{\mathtt{rep}}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k}) \cdot \mathbf{x}$ (see Section 6.3 for this notation) for $k \in [Q_{MS}]$, namely, the value of $\mathbf{M} \cdot \mathbf{x}$, where $\mathbf{M} \in \mathbb{Z}_p^{Q_{MS}\times(2Q_{MS}+1)}$ is given by

$$\mathbf{M} = \begin{bmatrix} 1 & b_1 & 0 & 0 & \cdots & 0 & 0 & a_{1,1}c_1 \\ 0 & 0 & 1 & b_2 & \cdots & 0 & 0 & a_{2,1}c_2 \\ & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & b_{Q_{MS}} & a_{Q_{MS},1}c_{Q_{MS}} \end{bmatrix}.$$

Under certain conditions, the reduction can learn from $\mathcal{A}$ one more linear equation independent of these Q_{MS} ones. In other words, it learns the inner product value $\mathbf{v} \cdot \mathbf{x} \in \mathbb{Z}_p$ for some row vector $\mathbf{v} \in \mathbb{Z}_p^{2Q_{MS}+1}$ that is not in the row span of $\mathbf{M}$:

$$\mathbf{v} \notin \mathsf{Span}\{\bar{\mathtt{rep}}(R_{k,1,1}R_{k,1,2}^{b_k}X^{a_{k,1}c_k})\}_{k\in[Q_{MS}]}.$$

Then the reduction can use its remaining Q_{MS} $\mathtt{DLOG}$ queries to learn Q_{MS} more independent equations. In total, the reduction collects $2Q_{MS}+1$ linearly independent equations of $\mathbf{x}$ and thus can solve for the value of $\mathbf{x}$, using a total of $2Q_{MS}$ queries to $\mathtt{DLOG}$. This solves the AOMDL instance.

We now explain why an efficient completion to a full-rank system is possible. This is stated in the following lemma, along with an explicit procedure. Essentially, this is a computationally efficient version of the classical Steinitz exchange lemma.

Lemma 7 (Efficient Completion to Full Rank). *Let $m := 2Q_{MS}+1$ and let $k \geq Q_{MS}+1$ be any number. Furthermore, let $\mathbf{M} \in \mathbb{Z}_p^{k\times m}$ be a matrix with $\mathrm{rank}(\mathbf{M}) = Q_{MS}+1$. Then, on input $\mathbf{M}$, Q_{MS}, and p, there exists an efficient algorithm that finds (column) vectors $\mathbf{u}_1, \ldots, \mathbf{u}_{Q_{MS}} \in \mathbb{Z}_p^m$ such that*

$$\mathrm{rank}\left(\begin{pmatrix} \mathbf{M} \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{Q_{MS}}^T \end{pmatrix}\right) = 2Q_{MS}+1.$$

Proof. Consider the standard basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_m \in \mathbb{Z}_p^m$, where $\mathbf{e}_i$ is the vector with a 1 in the i-th position and 0 elsewhere. Starting with $\mathbf{M}_0 := \mathbf{M}$, iterate over $i = 1, \ldots, m$ and test whether

$$\operatorname{rank}\left(\begin{pmatrix}\mathbf{M}_j \\ \mathbf{e}_i^T\end{pmatrix}\right) = \operatorname{rank}(\mathbf{M}_j) + 1.$$

If so, set $\mathbf{M}_{j+1} := \begin{pmatrix}\mathbf{M}_j \\ \mathbf{e}_i^T\end{pmatrix}$, and record $\mathbf{u}_{j+1} := \mathbf{e}_i$; otherwise, skip $\mathbf{e}_i$ and continue.

Since $\operatorname{rank}(\mathbf{M}) = Q_{MS} + 1$ and $m = 2Q_{MS} + 1$, the row space of $\mathbf{M}$ has codimension Q_{MS} in $\mathbb{Z}_p^m$. Hence, among the vectors $\mathbf{e}_1, \ldots, \mathbf{e}_m$, there exist exactly Q_{MS} vectors that are linearly independent of the rows of $\mathbf{M}$. The above procedure thus succeeds in selecting Q_{MS} such vectors and yields a matrix of full rank m. Each rank test can be performed efficiently using Gaussian elimination over $\mathbb{Z}_p$. Since at most m such tests are performed and $m = 2Q_{MS} + 1$, the overall procedure runs in time polynomial in Q_{MS}.

For each of our reductions, we only need to specify under which conditions it can learn one more linear equation from $\mathcal{A}$, and then we analyze the probability of those conditions.

Proof of Lemma 1 ($\mathbf{G_2} \to \mathbf{G_3}$)

Proof (Proof of Lemma 1).

Reduction $\mathcal{C}$ follows the structure detailed in Section 6.3. It simulates $\mathbf{G}_3$ for $\mathcal{A}$. We now argue that, if the abort condition introduced in $\mathbf{G}_3$ occurs, then $\mathcal{C}$ learns a new linear equation of the unknown discrete logarithms which is independent of those it has learned from the previous $\texttt{DLOG}$ queries. Recall that $\mathbf{G}_3$ aborts if the adversary $\mathcal{A}$ (i) issues a $\texttt{SSIGN}$ or $\texttt{SIGN}_1$ query with randomness ρ such that the corresponding rerandomized public key $\hat{\mathsf{pk}} := Xg^{\rho}$ satisfies $\hat{\mathsf{pk}} = 1_{\mathbb{G}}$, or (ii) outputs a forgery with randomness $\rho = \rho^*$ such that the rerandomized public key $\hat{\mathsf{pk}} := Xg^{\rho^*}$ satisfies $\hat{\mathsf{pk}} = 1_{\mathbb{G}}$. If this happens, then $\mathcal{C}$ learns that the discrete logarithm x of X is $-\rho$. Namely, it learns the value of $\bar{\mathsf{rep}}(X) \cdot \mathbf{x} = (0, 0, \ldots, 0, 0, 1) \cdot \mathbf{x}$, where $\mathbf{x} = (r_{1,1,1}, r_{1,1,2}, \ldots, r_{Q_{MS},1,1}, r_{Q_{MS},1,2}, x)^T$. We can see that

$$\bar{\mathsf{rep}}(X) \notin \mathsf{Span}\{\bar{\mathsf{rep}}(R_{k,1,1} R_{k,1,2}^{b_k} X^{a_{k,1} c_k})\}_{k \in [Q_{MS}]},$$

that is, $\bar{\mathsf{rep}}(X) = (0, 0, \ldots, 0, 1)$ is not in the row span of matrix $\mathbf{M}$ as defined in Section 6.3. Thus, $\mathcal{C}$ can solve its AOMDL instance if the abort condition introduced in $\mathbf{G}_3$ occurs. Therefore, we have

$$\mathsf{Adv}^{\mathrm{AOMDL}}_{\mathcal{C},\mathsf{GrGen}}(\lambda) \geq \left|\Pr\left[\mathbf{G}_2 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_3 \Rightarrow 1\right]\right|. \qquad \blacksquare$$

7 Application to Deterministic Wallets

Cryptocurrency users require a secure mechanism to store and manage their keys and related information, commonly referred to as a *wallet*. We focus on the case of *deterministic wallets*, as suggested in Bitcoin Improvement Proposal 32 (BIP32) [Wui12]. At a high level, a deterministic wallet contains a pair of master keys $(\mathsf{msk}, \mathsf{mpk})$ and a secret bitstring ch, referred to as the *chaincode*. To derive a key pair associated with an identifier ID, a randomness offset is first deterministically computed as $\omega := \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$, followed by the key derivation $\mathsf{sk}_{\mathsf{ID}} := \mathsf{msk} + \omega$ and $\mathsf{pk}_{\mathsf{ID}} := \mathsf{mpk} \cdot g^{\omega}$. This derivation satisfies two important properties: (1) *compact key storage*: the wallet only needs to store the master keys $(\mathsf{msk}, \mathsf{mpk})$ rather than a large set of random key pairs; and (2) *unlinkability*: the derived key pair $(\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}})$ cannot be linked to the master key pair $(\mathsf{msk}, \mathsf{mpk})$ as long as the chaincode ch remains secret.

In previous work [DFL19], a model and security analysis for deterministic wallets has been proposed. However, this work covers the case where the keys managed by wallets are only used for standard signatures. This motivates the adaptation that we present here: we model a wallet that allows to use keys for both standard and multi-signatures, and show how such a wallet can be constructed from our joint-signature scheme with rerandomizable keys $\hat{\mathsf{JS}}$.

7.1 Joint Deterministic Wallets

Below, we give the syntax and definition for a joint deterministic wallet.

Definition 7 (Joint Deterministic Wallet). *A* joint deterministic wallet *is defined as a tuple of algorithms* $\mathsf{JWal} = (\mathsf{Setup}, \mathsf{MKeyGen}, \mathsf{SKDer}, \mathsf{PKDer}, \mathsf{WKeyAggr}, \mathsf{WSSign}, \mathsf{WMSign}, \mathsf{WVerify})$ *specified as follows:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{par}, \mathsf{ch})$*: The* setup *algorithm takes as input the security parameter* 1^λ *and outputs global system parameters* par *and a chaincode* ch*.*
- $\mathsf{MKeyGen}(\mathsf{par}) \to (\mathsf{msk}, \mathsf{mpk})$*: The* master key generation *algorithm* MKeyGen *takes as input the parameters* par *and outputs a pair of master keys* $(\mathsf{msk}, \mathsf{mpk})$*.*
- $\mathsf{SKDer}(\mathsf{msk}, \mathsf{ID}) \to \mathsf{sk}_{\mathsf{ID}}$*: The deterministic* secret-key derivation *algorithm* SKDer *takes as input the master secret key* msk *and an identifier* ID*, and outputs a secret key* $\mathsf{sk}_{\mathsf{ID}}$*.*
- $\mathsf{PKDer}(\mathsf{mpk}, \mathsf{ID}) \to \mathsf{pk}_{\mathsf{ID}}$*: The deterministic* public-key derivation *algorithm* PKDer *takes as input the master public key* mpk *and an identifier* ID*, and outputs a public key* $\mathsf{pk}_{\mathsf{ID}}$*.*
- $\mathsf{WKeyAggr}(\mathcal{P}) \to \tilde{\mathsf{pk}}$*: The deterministic* key aggregation *algorithm* WKeyAggr *takes as input a set of public keys* $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ *and outputs an aggregated public key* $\tilde{\mathsf{pk}}$*.*
- $\mathsf{WSSign}(\mathsf{sk}, m) \to \sigma$*: The* standard signing *algorithm* WSSign *takes as input a secret key* sk *and a message* m*, and outputs a signature* σ*.*
- $\mathsf{WMSign} = (\mathsf{WSign}_0, \mathsf{WSignAggr}_0, \mathsf{WSign}_1, \mathsf{WSignAggr}_1, \mathsf{WSign}_2)$*: The interactive* multi signing protocol *is formally given by five algorithms:*
  - $\mathsf{WSign}_0() \to (\mathsf{pm}_1, \mathsf{St}_1)$*: This algorithm takes no explicit input, and outputs a first-round protocol message* pm_1 *and a first-round private state* St_1*.*
  - $\mathsf{WSignAggr}_0(\mathcal{M}_1) \to \tilde{\mathsf{pm}}_1$*: This algorithm is deterministic and takes as input a tuple of first-round protocol messages* $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,n})$*, and outputs an aggregated first-round protocol message* $\tilde{\mathsf{pm}}_1$*.*
  - $\mathsf{WSign}_1(\mathcal{P}, \mathsf{sk}, m, \mathsf{St}_1, \tilde{\mathsf{pm}}_1) \to (\mathsf{pm}_2, \mathsf{St}_2)$*: This algorithm takes as input a multiset of public keys* $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$*, a secret key* sk*, a message* m*, a first-round private state* St_1*, an aggregated first-round protocol message* $\tilde{\mathsf{pm}}_1$*, and outputs a second-round protocol message* pm_2 *and a second-round private state* St_2*.*
  - $\mathsf{WSignAggr}_1(\mathcal{M}_2) \to \tilde{\mathsf{pm}}_2$*: This algorithm is deterministic and takes as input a tuple of second-round protocol messages* $\mathcal{M}_2 = (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,n})$*, and outputs an aggregated second-round protocol message* $\tilde{\mathsf{pm}}_2$*.*
  - $\mathsf{WSign}_2(\mathsf{St}_2, \tilde{\mathsf{pm}}_2) \to \sigma$*: This algorithm takes as input a second-round private state* St_2 *and an aggregated second-round protocol message* $\tilde{\mathsf{pm}}_2$*, and outputs a signature* σ*.*
- $\mathsf{WVerify}(\mathsf{pk}, m, \sigma) \to 0/1$*: the deterministic* verification *algorithm* WVerify *takes as input a public key* pk*, a message* m*, and a signature* σ*, and outputs either* 1 *(accept) or* 0 *(reject).*

We say that a joint deterministic wallet scheme JWal *is* correct *if*

1. *for all* $(\mathsf{par}, \mathsf{ch}) \in \mathsf{Setup}(1^\lambda)$*,* $\mathsf{ID} \in \{0,1\}^*$*, all* $(\mathsf{msk}, \mathsf{mpk}) \in \mathsf{MKeyGen}(\mathsf{par})$*, and* $m \in \{0,1\}^*$*, with* $\mathsf{pk}_{\mathsf{ID}} := \mathsf{PKDer}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$ *and* $\mathsf{sk}_{\mathsf{ID}} := \mathsf{SKDer}(\mathsf{msk}, \mathsf{ch}, \mathsf{ID})$*, it holds that:*

$$\Pr\left[\mathsf{WVerify}(\mathsf{pk}_{\mathsf{ID}}, m, \sigma) = 1 \mid \sigma \leftarrow \mathsf{WSSign}(\mathsf{sk}_{\mathsf{ID}}, m)\right] = 1$$

2. *for all* $n \in \mathit{poly}(\lambda)$*,* $i \in [n]$*,* $(\mathsf{par}, \mathsf{ch}_i) \in \mathsf{Setup}(1^\lambda)$*,* $\mathsf{ID}_i \in \{0,1\}^*$*,* $(\mathsf{msk}_i, \mathsf{mpk}_i) \in \mathsf{MKeyGen}(\mathsf{par})$*, with* $\mathsf{pk}_{\mathsf{ID}_i} := \mathsf{PKDer}(\mathsf{mpk}_i, \mathsf{ch}_i, \mathsf{ID}_i)$*,* $\mathsf{sk}_{\mathsf{ID}_i} := \mathsf{SKDer}(\mathsf{msk}_i, \mathsf{ch}_i, \mathsf{ID}_i)$*, it holds that:*

$$\Pr\left[\mathsf{Verify}(\tilde{\mathsf{pk}}, m, \sigma) = 1 \;\middle|\; \begin{array}{r} \tilde{\mathsf{pk}} \leftarrow \mathsf{KeyAggr}(\mathcal{P}), \\ \sigma \leftarrow \mathsf{JWal.Exec}(\mathcal{P}, \mathcal{S}, m) \end{array}\right] = 1,$$

where $\mathcal{P} := \{\mathsf{pk}_{\mathsf{ID}_1}, \ldots, \mathsf{pk}_{\mathsf{ID}_n}\}$*,* $\mathcal{S} := \{\mathsf{sk}_{\mathsf{ID}_1}, \ldots, \mathsf{sk}_{\mathsf{ID}_n}\}$*, and algorithm* $\mathsf{JWal.Exec}(\mathcal{P}, \mathcal{S}, m)$ *models an honest execution of the multi-signing protocol* WMSign *for* JWal *as defined in Figure 6.*

Security. We consider the model of *hot* and *cold* wallets, which has been widely studied in recent works [DFL19, AADE+20, DEF+21]. In this model, the hot wallet refers to a wallet that is almost always connected to the Internet; for instance, it may be a mobile or browser-based software application running on the user's phone or laptop. In contrast, the cold wallet denotes its offline counterpart, such as a USB device. The general idea is to store the public key in the hot wallet, while keeping the secret key in the cold wallet. This setup allows the secret key to come online only when necessary and otherwise remain offline, thereby providing protection against online attacks. Note that the chaincode is stored in both wallets, and due to the deterministic nature of key derivation, a valid key pair can be generated from the corresponding master key, identifier, and the chaincode without any interaction between the two wallets.

Algorithm $\mathsf{JWal.Exec}(\mathcal{P}, \mathcal{S}, m)$

$\mathbf{let}\ \mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\},\ \mathcal{S} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$
$\mathbf{for}\ i \in [n] : (\mathsf{pm}_{1,i}, \mathsf{St}_{1,i}) \leftarrow \mathsf{WSign}_0()$
$\mathbf{let}\ \mathcal{M}_1 := (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,n})$
$\tilde{\mathsf{pm}}_1 := \mathsf{WSignAggr}_0(\mathcal{M}_1)$
$\mathbf{for}\ i \in [n] : (\mathsf{pm}_{2,i}, \mathsf{St}_{2,i}) \leftarrow \mathsf{WSign}_1(\mathcal{P}, \mathsf{sk}_i, m, \mathsf{St}_{1,i}, \tilde{\mathsf{pm}}_1)$
$\mathbf{let}\ \mathcal{M}_2 := (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,n})$
$\tilde{\mathsf{pm}}_2 := \mathsf{WSignAggr}_1(\mathcal{M}_2)$
$\mathbf{for}\ i \in [n] : \sigma_i \leftarrow \mathsf{WSign}_2(\mathsf{St}_{2,i}, \tilde{\mathsf{pm}}_2)$
$\mathbf{if}\ \exists i \neq j \in [n]\ \text{s.t}\ \sigma_i \neq \sigma_j : \mathbf{return}\ \bot$
$\mathbf{return}\ \sigma := \sigma_1$

Fig. 6. Algorithm JWal.Exec for a joint wallet scheme JWal. This models an honest execution of the multi-signing protocol WMSign for public key multiset $\mathcal{P}$, secret keys $\mathcal{S}$, and message m.

Game $\mathbf{WUNF}^{\mathcal{A}}_{\mathsf{JWal}}(\lambda)$

$(\mathsf{par}, \mathsf{ch}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{MKeyGen}(\mathsf{par})$
$\mathsf{sid} := 0,\ S_0 := \emptyset,\ S_1 := \emptyset,\ Q := \emptyset$
$\mathtt{WMSIGN} := (\mathtt{WSIGN}_0, \mathtt{WSIGN}_1, \mathtt{WSIGN}_2)$
$\mathsf{oracles} := (\mathtt{PK}, \mathtt{WSSIGN}, \mathtt{WMSIGN})$
$(\mathcal{P}^*, m^*, \sigma^*, \mathsf{ID}^*, b^*) \leftarrow \mathcal{A}^{\mathsf{oracles}}(\mathsf{par}, \mathsf{ch}, \mathsf{mpk})$
$\mathsf{pk}_{\mathsf{ID}^*} \leftarrow \mathsf{PKDer}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID}^*)$
$\mathbf{if}\ (\mathsf{pk}_{\mathsf{ID}^*} \notin \mathcal{P}^*) \vee ((\mathcal{P}^*, m^*) \in Q) \vee (b^* \notin \{0,1\}) : \mathbf{return}\ 0$
$\mathbf{if}\ b^* = 0 :$
  $\mathbf{if}\ \mathcal{P}^* \neq \mathsf{pk}_{\mathsf{ID}^*} : \mathbf{return}\ 0$
  $\mathbf{return}\ \mathsf{WVerify}(\mathsf{pk}_{\mathsf{ID}^*}, m^*, \sigma^*)$
$\mathbf{return}\ \mathsf{WVerify}(\mathsf{KeyAggr}(\mathcal{P}^*), m^*, \sigma^*)$

Game $\mathbf{WUNL}^{\mathcal{A}}_{\mathsf{JWal}}(\lambda)$

$(\mathsf{par}, \mathsf{ch}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{MKeyGen}(\mathsf{par})$
$\mathsf{sid} := 0,\ S_0 := \emptyset,\ S_1 := \emptyset,\ Q := \emptyset$
$\mathtt{WMSIGN} := (\mathtt{WSIGN}_0, \mathtt{WSIGN}_1, \mathtt{WSIGN}_2)$
$\mathsf{oracles} := (\mathtt{PK}, \mathtt{WSSIGN}, \mathtt{WMSIGN}, \mathtt{CHALL})$
$b' \leftarrow \mathcal{A}^{\mathsf{oracles}}$
$\mathbf{if}\ b' = b : \mathbf{return}\ 1$
$\mathbf{return}\ 0$

Oracle $\mathtt{CHALL}(\mathsf{ID})$

If $K[\mathsf{ID}] \neq \bot$: $\mathbf{return}\ \bot$
$\mathsf{sk}^0 \leftarrow \mathsf{SKDer}(\mathsf{msk}, \mathsf{ch}, \mathsf{ID})$
$\mathsf{pk}^0 \leftarrow \mathsf{PKDer}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$
$(\mathsf{sk}^1, \mathsf{pk}^1) \leftarrow \mathsf{MKeyGen}(\mathsf{par})$
$b \leftarrow_\$ \{0,1\}$
$K[\mathsf{ID}] := (\mathsf{sk}^b, \mathsf{pk}^b)$
$\mathbf{return}\ (\mathsf{sk}^b, \mathsf{pk}^b)$

Oracle $\mathtt{PK}(\mathsf{ID})$

If $K[\mathsf{ID}] \neq \bot$: $\mathbf{return}\ \bot$
$\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{SKDer}(\mathsf{msk}, \mathsf{ch}, \mathsf{ID})$
$\mathsf{pk}_{\mathsf{ID}} \leftarrow \mathsf{PKDer}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$
$K[\mathsf{ID}] := (\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}})$
$\mathbf{return}\ \mathsf{pk}_{\mathsf{ID}}$

Oracle $\mathtt{WSSIGN}(m, \mathsf{ID})$

If $K[\mathsf{ID}] = \bot$: $\mathbf{return}\ \bot$
$(\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}}) := K[\mathsf{ID}]$
$Q := Q \cup \{(\mathsf{pk}_{\mathsf{ID}}, m)\}$
$\sigma \leftarrow \mathsf{WSSign}(\mathsf{sk}_{\mathsf{ID}}, m)$
$\mathbf{return}\ \sigma$

Oracle $\mathtt{WSIGN}_0(\mathsf{ID})$

$\mathsf{sid} := \mathsf{sid} + 1$
$k := \mathsf{sid},\ S_0 := S_0 \cup \{k\}$
$(\mathsf{pm}_1, \mathsf{St}_{1,k}) \leftarrow \mathsf{WSign}_0()$
$\mathbf{return}\ \mathsf{pm}_1$

Oracle $\mathtt{WSIGN}_1(k, \mathcal{P}, m, \tilde{\mathsf{pm}}_1)$

$\mathbf{let}\ \mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$
If $K[\mathsf{ID}] = \bot$: $\mathbf{return}\ \bot$
$(\mathsf{sk}_{\mathsf{ID}}, \mathsf{pk}_{\mathsf{ID}}) := K[\mathsf{ID}]$
$\mathbf{if}\ (\mathsf{pk}_{\mathsf{ID}} \notin \mathcal{P}) \vee (k \notin S_0) :$
  $\mathbf{return}\ \bot$
$S_0 := S_0 \setminus \{k\}$
$S_1 := S_1 \cup \{k\}$
$Q := Q \cup \{(\mathcal{P}, m)\}$
$(\mathsf{pm}_2, \mathsf{St}_{2,k}) \leftarrow \mathsf{WSign}_1(\mathcal{P}, \mathsf{sk}_{\mathsf{ID}}, m, \mathsf{St}_{1,k}, \tilde{\mathsf{pm}}_1)$
$\mathbf{return}\ \mathsf{pm}_2$

Oracle $\mathtt{WSIGN}_2(k, \tilde{\mathsf{pm}}_2)$

$\mathbf{if}\ k \notin S_1 : \mathbf{return}\ \bot$
$S_1 := S_1 \setminus \{k\}$
$\sigma \leftarrow \mathsf{WSign}_2(\mathsf{St}_{2,k}, \tilde{\mathsf{pm}}_2)$
$\mathbf{return}\ \sigma$

Fig. 7. The games **WUNL** and **WUNF** for a joint deterministic wallet JWal and adversary $\mathcal{A}$. $\mathcal{A}$ has access to the oracles (PK, WSSIGN, $\mathtt{WSIGN}_0$, $\mathtt{WSIGN}_1$, $\mathtt{WSIGN}_2$) in both games, and additionally to CHALL in the **WUNL** game.

In this model, assuming that the cold wallet remains uncorrupted, we aim to ensure the properties of *wallet unforgeability* and *wallet unlinkability*, which we outline below.

At a high level, the property of *wallet unforgeability* ensures that, even when an adversary is given access to the hot wallet (specifically, the master public key mpk and the chaincode ch) and is further allowed to obtain signatures under keys derived for adversarially chosen identifiers ID, it remains computationally infeasible for the adversary to forge a valid wallet signature, except with negligible probability.

Definition 8 (WUNF Security). *Let* $\mathsf{JWal} = (\mathsf{Setup}, \mathsf{MKeyGen}, \mathsf{SKDer}, \mathsf{PKDer}, \mathsf{WKeyAggr}, \mathsf{WSSign}, \mathsf{WMSign}, \mathsf{WVerify})$ *be a joint deterministic wallet scheme. Consider the game* **WUNF** *defined in Figure 7. We say that* JWal *is WUNF secure if for all PPT adversaries* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathrm{WUNF}}_{\mathcal{A},\mathsf{JWal}}(\lambda) := \Pr\left[\mathbf{WUNF}^{\mathcal{A}}_{\mathsf{JWal}}(\lambda) \Rightarrow 1\right].$$

On the other hand, the property of *wallet unlinkability* ensures that, for an adversary who has access to the master public key mpk but not to the chaincode ch stored in the hot wallet, and who can observe signatures under keys derived for adversarially chosen identifiers ID, it is computationally infeasible to

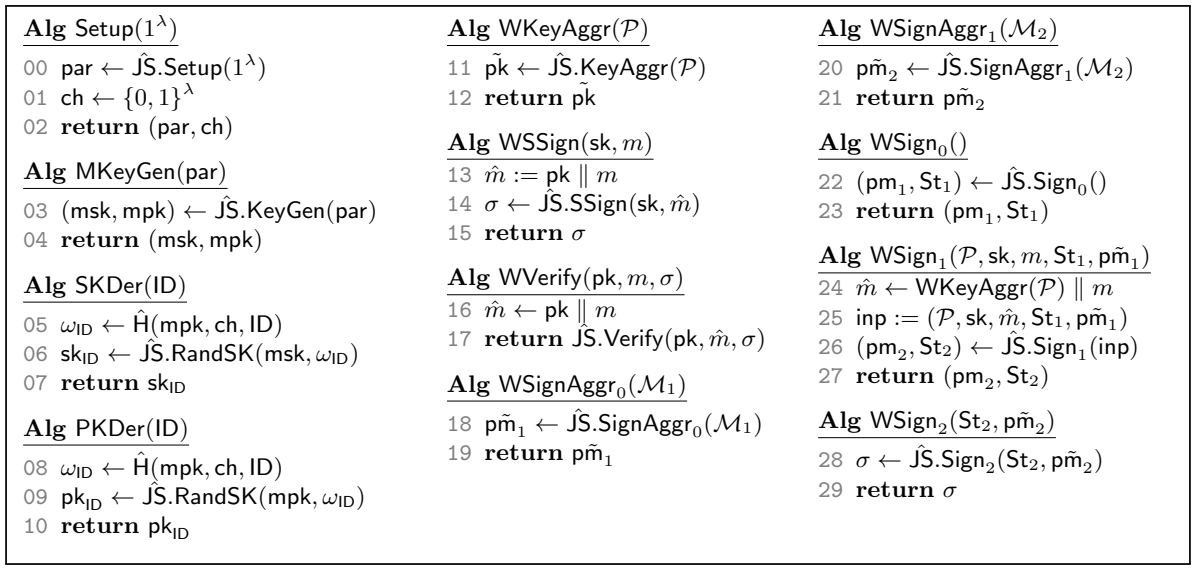


Fig. 8. The algorithms of the joint deterministic wallet scheme JWal from $\hat{\mathsf{JS}}$, our construction of joint signature scheme with rerandomizable keys and hash function $\hat{\mathsf{H}}$. Here, the joint signature scheme JS is derived from Schnorr and MuSig2 (Figure 4).

distinguish, except with negligible probability, a wallet-generated key pair from a randomly generated key pair.

Definition 9 (WUNL Security). *Let* JWal = (Setup, MKeyGen, SKDer, PKDer, WKeyAggr, WSSign, WMSign, WVerify) *be a joint deterministic wallet scheme. Consider the game* **WUNL** *defined in Figure 7. We say that* JWal *is WUNL secure if for all PPT adversaries* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathrm{WUNL}}_{\mathcal{A},\mathsf{JWal}}(\lambda) := \left|\Pr\left[\mathbf{WUNL}^{\mathcal{A}}_{\mathsf{JWal}}(\lambda) \Rightarrow 1\right] - 1/2\right|.$$

7.2 Construction and Security Analysis

In Figure 8, we construct a joint deterministic wallet scheme given a joint-signature scheme with rerandomizable keys. We now state the main theorems for this.

Theorem 2. *Let* JWal *be a joint deterministic wallet scheme as defined in Figure 8. Assume that* $\hat{\mathsf{JS}}$ *is an EUF-CMA-RK secure joint-signature scheme that satisfies the property of uniquely rerandomizable keys (Definition 10). Then,* JWal *satisfies the property of wallet unforgeability.*

Concretely, for any PPT adversary $\mathcal{A}$ *that makes at most* Q_{MS} *queries to signing oracles* $\mathtt{WSIGN}_0$, $\mathtt{WSIGN}_1$, $\mathtt{WSIGN}_2$*, at most* Q_S *queries to sign oracle* WSSIGN*, at most* $Q_{\hat{\mathsf{H}}}$ *queries to random oracle* $\hat{\mathsf{H}}$*, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathrm{WUNF}}_{\mathcal{A},\mathsf{JWal}}(\lambda) \leq \mathsf{Adv}^{\text{EUF-CMA-RK}}_{\mathcal{B},\hat{\mathsf{JS}}}(\lambda) + Q^2_{\hat{\mathsf{H}}}/p.$$

The wallet unforgeability property is proven for an underlying EUF-CMA-RK secure joint signature scheme which has the following property of *uniquely rerandomizable keys.*

Definition 10 (Joint-signature scheme with uniquely rerandomizable keys). *A rerandomizable joint-signature scheme* $\hat{\mathsf{JS}}$*, is said to have uniquely rerandomizable public keys if for all* $(\rho, \rho') \in \mathcal{R}$*, we have that* $\mathsf{RandPK}(\mathsf{pk}, \rho) = \mathsf{RandPK}(\mathsf{pk}, \rho')$ *implies* $\rho = \rho'$*.*

Proof. Let $\mathcal{A}$ be an adversary against the WUNF security of JWal. We show the statement via a sequence of games and a reduction to the EUF-CMA-RK security of $\hat{\mathsf{JS}}$. Concretely, we will build a reduction $\mathcal{B}$ that has black-box access to the oracles underlying the EUF-CMA-RK game of $\hat{\mathsf{JS}}$ and uses these to simulate to $\mathcal{A}$ the oracles underlying the WUNF game of JWal. Finally, when $\mathcal{A}$ returns a valid forgery in the WUNF game of JWal, the reduction $\mathcal{B}$ translates this into a valid forgery in the EUF-CMA-RK game of $\hat{\mathsf{JS}}$ it is playing. We start by giving a sequence of games.

Game $\mathbf{G}_0$: This game behaves exactly like the unforgeability game **WUNF**, that is $\mathbf{G}_0 = \mathbf{WUNF}^{\mathcal{A}}_{\mathsf{JWal}}$. As a small difference, internally $\mathbf{G}_0$ maintains a flag, initialized as $\mathsf{flag} := 0$, that is set as $\mathsf{flag} := 1$ when the following event happens: a query to the $\mathtt{PK}$ oracle on input identifier ID results in a public key $\mathsf{pk}_{\mathsf{ID}}$ that collides with a previously generated key $\mathsf{pk}_{\mathsf{ID}'} \in K[\mathsf{ID}']$, for $\mathsf{ID} \neq \mathsf{ID}'$.

Game $\mathbf{G}_1$: This game behaves same as game $\mathbf{G}_0$, except that it aborts when $\mathsf{flag} = 1$. Here we analyze the probability of abort in this game. Due to the property of uniquely rerandomizable keys (Definition 10), a collision of the form $\mathsf{pk}_{\mathsf{ID}} = \mathsf{pk}_{\mathsf{ID}'}$ implies $\hat{\mathsf{H}}(\mathsf{mpk}, ch, \mathsf{ID}) = \hat{\mathsf{H}}(\mathsf{mpk}, ch, \mathsf{ID}')$. Since there are at most $Q_{\hat{\mathsf{H}}}$ queries to $\hat{\mathsf{H}}$, the overall probability is bounded by ${Q_{\hat{\mathsf{H}}}}^2/p$ via a standard birthday bound.

Final Claim. We now conclude the proof by claiming that

$$\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] \leq \mathsf{Adv}^{\text{EUF-CMA-RK}}_{\mathcal{B},\hat{\mathsf{JS}}}(\lambda).$$

To prove the claim, we describe an efficient reduction $\mathcal{B}$ that plays in the EUF-CMA-RK game for $\hat{\mathsf{JS}}$ and simulates game $\mathbf{G}_1$ to adversary $\mathcal{A}$. To this end, $\mathcal{B}$ gets as input, parameters par and a public key $\bar{\mathsf{pk}}$ from $\mathbf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{B}}_{\bar{\mathsf{JS}}}$. Additionally, it has access to random oracles $\bar{\mathsf{H}}, \bar{\mathsf{H}}_a, \bar{\mathsf{H}}_b, \bar{\mathsf{H}}_c$, and also access to signing oracles $\mathtt{SS\bar{I}GN}, \mathtt{S\bar{I}GN}_0, \mathtt{S\bar{I}GN}_1, \mathtt{S\bar{I}GN}_2$ that simulate algorithms $\mathsf{SSign}, \mathsf{Sign}_0, \mathsf{Sign}_1, \mathsf{Sign}_2$ on secret key $\bar{\mathsf{sk}}$ (for public key $\bar{\mathsf{pk}}$) and randomness $\bar{\rho}$, respectively. Then, it simulates game $\mathbf{G}_1$ for $\mathcal{A}$ on input $\mathsf{mpk} := \bar{\mathsf{pk}}$ as follows:

- **Random oracle $\hat{\mathsf{H}}$.** On input u, $\mathcal{B}$ checks whether $\hat{\mathsf{H}}$ is already set at this point, then it returns $\hat{\mathsf{H}}[u]$. Otherwise, it samples $\rho \leftarrow_{\$} \mathbb{Z}_p$. $\mathcal{B}$ then sets $\hat{\mathsf{H}}[u] := \rho$ and returns ρ.
- **Public key oracle $\mathtt{PK}$.** On input identifier ID, $\mathcal{B}$ derives $\mathsf{pk}_{\mathsf{ID}}$ through the function $\hat{\mathsf{JS}}.\mathsf{RandPK}(\mathsf{mpk}, \omega_{\mathsf{ID}})$, where $\omega_{\mathsf{ID}} = \hat{\mathsf{H}}(\mathsf{mpk}, s, \mathsf{ID})$. If $\mathcal{B}$ encounters a collision of $\mathsf{pk}_{\mathsf{ID}}$ with a previously stored key in K it aborts the simulation. Otherwise it stores $K := K \cup (\mathsf{pk}_{\mathsf{ID}}, \omega_{\mathsf{ID}})$.
- **Signing oracle WSSign.** On input (m, ID), $\mathcal{B}$ fetches $(\mathsf{pk}_{\mathsf{ID}}, \omega_{\mathsf{ID}}) = K[\mathsf{ID}]$ and if $K[\mathsf{ID}] = \bot$ it aborts. Otherwise, it derives the public key prefixed message $\hat{m} = \mathsf{pk}_{\mathsf{ID}} \,\|\, m$ and makes an internal query to $\mathtt{SS\bar{I}GN}$ and receives a signature as $\sigma \leftarrow \mathtt{SS\bar{I}GN}(\hat{m}, \omega_{\mathsf{ID}})$.
- **Signing oracles $\mathtt{WSIGN}_i$.** These oracles for $i \in \{0, 2\}$ are simulated by forwarding the query to oracles $\mathtt{S\bar{I}GN}_i$.
- **Signing oracles $\mathtt{WSIGN}_1$.** On inputs session id k, public key set $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, message m, aggregated first round message $\tilde{\mathsf{pm}}_1$ and identifier ID, $\mathcal{B}$ fetches $(\mathsf{pk}_{\mathsf{ID}}, \omega_{\mathsf{ID}}) = K[\mathsf{ID}]$ and if $K[\mathsf{ID}] = \bot$ it aborts. Otherwise, it derives the public key prefixed message $\hat{m} = \mathsf{WKeyAggr}(\mathcal{P}) \,\|\, m$ and makes an internal query to $\mathtt{S\bar{I}GN}_1$ and receives output $(\mathsf{pm}_2, \mathsf{St}_{2,k}) \leftarrow \mathtt{S\bar{I}GN}_1(k, \mathcal{P}, \hat{m}, \tilde{\mathsf{pm}}_1, \omega_{\mathsf{ID}})$.

Finally, when $\mathcal{A}$ outputs its forgery $(\mathcal{P}^*, m^*, \sigma^*, \mathsf{ID}^*)$ where $\sigma^* = (R^*, s^*)$ and game $\mathbf{G}_1$ outputs 1 (which $\mathcal{B}$ can check efficiently), the reduction $\mathcal{B}$ proceeds as follows. It first recovers $(\mathsf{pk}_{\mathsf{ID}^*}, \omega_{\mathsf{ID}^*})$ from list K.

- If $\mathsf{WVerify}(\mathsf{pk}_{\mathsf{ID}^*}, \mathsf{pk}_{\mathsf{ID}^*} \,\|\, m^*, \sigma^*) = 1$, it submits a forgery $(\mathsf{pk}_{\mathsf{ID}^*}, \mathsf{pk}_{\mathsf{ID}^*} \,\|\, m^*, \sigma^*, \omega_{\mathsf{ID}^*})$ to the EUF-CMA-RK game.
- Let $\tilde{\mathcal{P}}^* = \mathsf{WKeyAggr}(\mathcal{P}^*)$. If $\mathsf{WVerify}(\tilde{\mathcal{P}}^*, \tilde{\mathcal{P}}^* \,\|\, m^*, \sigma^*) = 1$ then it submits $(\tilde{\mathcal{P}}^*, \tilde{\mathcal{P}}^* \,\|\, m^*, \sigma^*, \omega_{\mathsf{ID}^*})$ to the EUF-CMA-RK game.

In either case $\mathcal{B}$ submits a valid forgery in the EUF-CMA-RK game because:

- In the case of a multi-signature, it is clear $\mathsf{pk}_{\mathsf{ID}^*} \in \mathcal{P}^*$.
- If $(\mathcal{P}^*, m^*) \notin Q$, then $\mathcal{B}$ never simulated a signature for this message to $\mathcal{A}$, this implies $({\mathsf{pk}_{\mathsf{ID}}}^*, {\mathsf{pk}_{\mathsf{ID}}}^* \,\|\, m^*) \notin \bar{Q}$ and $(\mathcal{P}^*, \mathsf{WKeyAggr}(\mathcal{P}^*) \,\|\, m^*) \notin \bar{Q}$. Hence the forgery is on a fresh message.

Theorem 3. *Let JWal be a joint deterministic wallet scheme as defined in Figure 8. Assume that $\hat{\mathsf{JS}}$ satisfies the properties of perfect rerandomizability of keys. Then, JWal satisfies the property of wallet unlinkability.*

Concretely, for any PPT adversary $\mathcal{A}$ that makes at most $Q_{\mathtt{PK}}$ queries to public key oracle, at most Q_{MS} queries to signing oracles $\mathtt{WSIGN}_0$, $\mathtt{WSIGN}_1$, $\mathtt{WSIGN}_2$, at most Q_S queries to sign oracle $\mathtt{WSSIGN}$, at most $Q_{\hat{\mathsf{H}}}$ queries to random oracle $\hat{\mathsf{H}}$, we have

$$\mathsf{Adv}^{\text{WUNL}}_{\mathcal{A},\mathsf{JWal}}(\lambda) \leq Q_{\hat{\mathsf{H}}}/2^{\lambda}.$$

Proof. It is easy to see that, due to the perfect rerandomizability property of $\hat{\mathsf{JS}}$, the keys pk^0 and pk^1 are uniformly distributed as fresh keys, from $\mathcal{A}$'s perspective, as long as the chaincode ch is hidden. Now we analyze the probability with which $\mathcal{A}$ makes a query involving ch, of the form $\hat{\mathsf{H}}(\cdot, \mathsf{ch}, \cdot)$. Since the number of queries to $\hat{\mathsf{H}}$ is at most $Q_{\hat{\mathsf{H}}}$, the probability that $\mathcal{A}$ makes such a query is bounded by $Q_{\hat{\mathsf{H}}}/2^{\lambda}$.

Acknowledgments

Julian Loss was supported by the European Union, ERC-2023-StG-101116713, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972. Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

AADE+20. Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1017–1031, New York, NY, USA, 2020. Association for Computing Machinery.

AB21. Handan Kilinç Alper and Jeffrey Burdges. Two-round trip Schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, August 2021. Springer, Cham.

Bla23. Brandon Black. Field report: Implementing musig2, 2023.

BLL+22. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. *Journal of Cryptology*, 35(4):25, October 2022.

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

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

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Cho24. Ava Chow. wallet: Be able to receive and spend inputs involving musig2 aggregate keys. Pull request to Bitcoin Core repository, 2024.

DEF+21. Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. The exact security of BIP32 wallets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1020–1042. ACM Press, November 2021.

DEF+25. Poulami Das, Andreas Erwig, Sebastian Faust, Philipp-Florens Lehwalder, Julian Loss, Ziyan Qu, and Siavash Riahi. Bip32-compatible threshold wallets. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '25, page 856–872, New York, NY, USA, 2025. Association for Computing Machinery.

DFL19. Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 651–668. ACM Press, November 2019.

ER22. Andreas Erwig and Siavash Riahi. Deterministic wallets for adaptor signatures. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part II*, volume 13555 of *LNCS*, pages 487–506. Springer, Cham, September 2022.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

FKM+16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Berlin, Heidelberg, March 2016.

GK24. Conrado P. L. Gouvea and Chelsea Komlo. Re-randomized FROST. Cryptology ePrint Archive, Paper 2024/436, 2024.

GKMN21. François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 2021. Springer, Cham.

GS15. Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 497–504. Springer, Berlin, Heidelberg, January 2015.

GS22. Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 365–396. Springer, Cham, May / June 2022.

IS24. Salvatore Ingala and Yannick Seurin. Musig2 – a valuable addition to the bitcoin builder's toolkit, 2024.
Kat10. Jonathan Katz. *Digital Signatures: Background and Definitions*, pages 3–33. Springer US, Boston, MA, 2010.
LFA20. Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese. Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 323–343. Springer, Cham, December 2020.
MPSW19. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptography*, 87(9):2139–2164, September 2019.
Nak08. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
Nic22a. Jonas Nick. A Note on the Unforgeability of MuSig2 with Key Tweaking. `https://github.com/jonasnick/musig2-tweaking`, 2022. GitHub repository, accessed 2025-12-30.
Nic22b. Jonas Nick. Add mandatory pk argument to noncegen and update test vectors. Pull request to BIPs repository, 2022.
Nic24. Jonas Nick. Add module "musig" that implements musig2 multi-signatures (bip 327). Pull request to libsecp256k1 repository, 2024.
NKDM03. Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2003)*, San Diego, CA, USA, February 2003. Internet Society.
NRJ22. Jonas Nick, Tim Ruffing, and Elliott Jin. MuSig2 for BIP340-compatible multi-signatures. Bitcoin Improvement Proposal (BIP) 327, Bitcoin Improvement Proposals, 2022.
NRS21. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Cham.
NRS26. Jonas Nick, Tim Ruffing, and Yannick Seurin. DahLIAS: Discrete logarithm-based interactive aggregate signatures. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, volume 16541 of *Lecture Notes in Computer Science*, pages 215–245, Cham, 2026. Springer.
NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020.
Ruf25. Tim Ruffing. The post-quantum security of Bitcoin's Taproot as a commitment scheme. Cryptology ePrint Archive, Report 2025/1307, 2025.
TR22. Jonas Nick Tim Ruffing, Elliott Jin. Re: [bitcoin-dev] musig2 bip. Post to bitcoin-dev mailing list, 2022.
Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Berlin, Heidelberg, August 2002.
WNR20. Pieter Wuille, Jonas Nick, and Tim Ruffing. Schnorr signatures for secp256k1. Bitcoin Improvement Proposal (BIP) 340, Bitcoin Improvement Proposals, 2020.
WNT20a. Pieter Wuille, Jonas Nick, and Anthony Towns. Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal (BIP) 341, Bitcoin Improvement Proposals, 2020.
WNT20b. Pieter Wuille, Jonas Nick, and Anthony Towns. Validation of taproot scripts. Bitcoin Improvement Proposal (BIP) 342, Bitcoin Improvement Proposals, 2020.
Wui12. Pieter Wuille. Hierarchical deterministic wallets. Bitcoin Improvement Proposal (BIP) 32, Bitcoin Improvement Proposals, 2012.

A Postponed Definitions

A.1 MuSig2 Multi-Signature

We give a textual description of the Schnorr multi-signature scheme $\mathsf{MuSig2}[\mathsf{GrGen}, \nu]$.

- **Parameters Setup (Setup).** On input 1^λ, the setup algorithm Setup runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$, selects three hash functions $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c \colon \{0,1\}^* \to \mathbb{Z}_p$, and outputs the public parameters $\mathsf{par} := (\mathbb{G}, p, g, \mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c)$.
- **Key Generation (KeyGen).** A signer samples $x \leftarrow_\$ \mathbb{Z}_p$, computes $X := g^x \in \mathbb{G}$, and outputs its public-secret key pair as $(\mathsf{sk}, \mathsf{pk}) := (x, X)$.
- **Key Aggregation ($\mathsf{KeyAggr}$).** Let $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ be a multiset of public keys. For all $i \in [n]$, the i-th key aggregation coefficient is defined as $a_i := \mathsf{H}_a(\langle \mathcal{P} \rangle, \mathsf{pk}_i)$. Then, the aggregated public key $\tilde{\mathsf{pk}}$ (corresponding to $\mathcal{P}$) is computed as $\tilde{\mathsf{pk}} := \prod_{i \in [n]} \mathsf{pk}_i^{a_i}$.
- **Signing Protocol.** For n signers, let $\mathcal{P} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ be the multiset of their public keys. We describe the signing protocol $\mathsf{Sign} = (\mathsf{Sign}_0, \mathsf{SignAggr}_0, \mathsf{Sign}_1, \mathsf{SignAggr}_1, \mathsf{Sign}_2)$ from the perspective of the first signer with secret key sk_1 and public key pk_1.

- **First Signing Round (Sign_0).** For all $j \in [\nu]$, sample $r_{1,j} \leftarrow_\$ \mathbb{Z}_p$ and compute $R_{1,j} := g^{r_{1,j}}$. Store the private state $\mathsf{St}_1 := (r_{1,1}, \ldots, r_{1,\nu})$ and output the protocol message $\mathsf{pm}_1 := (R_{1,1}, \ldots, R_{1,\nu})$.
- **First-Round Aggregation ($\mathsf{SignAggr}_0$).** Let $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,n})$ be the tuple of first-round protocol messages received from other signers. Here, message $\mathsf{pm}_{1,i}$ is sent by signer i and has the form $\mathsf{pm}_{1,i} = (R_{i,1}, \ldots, R_{i,\nu})$. Then, for each $j \in [\nu]$, compute $R_j := \prod_{i \in [n]} R_{i,j}$, and output $\tilde{\mathsf{pm}}_1 = (R_1, \ldots, R_\nu)$.
- **Second Signing Round (Sign_1).** Let $\tilde{\mathsf{pm}}_1$ be the aggregated first-round protocol message as computed above, and let $m \in \{0,1\}^*$ be some message to be signed. Compute $\tilde{\mathsf{pk}} := \mathsf{KeyAggr}(\mathcal{P})$ and $b := \mathsf{H}_b(\tilde{\mathsf{pk}}, \tilde{\mathsf{pm}}_1, m)$. Further, compute $c := \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$ with

$$R := \prod_{j \in [\nu]} R_j^{b^{j-1}}, \quad s_1 := c a_1 \mathsf{sk}_1 + \sum_{j \in [\nu]} r_{1,j} b^{j-1},$$

  where a_1 is computed as above. Store the private state $\mathsf{St}_2 := R$ and output the protocol message $\mathsf{pm}_2 := s_1$.
- **Second-Round Aggregation ($\mathsf{SignAggr}_1$).** Let $\mathcal{M}_2 = (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,n})$ be the tuple of second-round protocol messages received from other signers. Here, message $\mathsf{pm}_{2,i}$ is sent by signer i and has the form $\mathsf{pm}_{1,i} = s_i$. Compute the sum $s := \sum_{i \in [n]} s_i \in \mathbb{Z}_p$ and output $\tilde{\mathsf{pm}}_2 := s$.
- **Signature Output (Sign_2).** Let $\tilde{\mathsf{pm}}_2$ be the aggregated second-round protocol message as computed above. Output the signature $\sigma := (R, s)$ where $s := \tilde{\mathsf{pm}}_2$.

- **Verification.** To verify a signature σ for public key $\tilde{\mathsf{pk}}$ and message m, parse $\sigma = (R, s)$ and compute $c := \mathsf{H}_c(\tilde{\mathsf{pk}}, R, m)$. Then, output 1 iff $g^s = R \cdot \tilde{\mathsf{pk}}^c$ holds.

A.2 EUF-CMA Security for Joint-Signatures

We give a formal definition of EUF-CMA security for joint-signature schemes.

Definition 11 (EUF-CMA Security of JS). *Let $\mathsf{JS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAggr}, \mathsf{SSign}, \mathsf{MSign}, \mathsf{Verify})$ be a joint-signature scheme. Consider the game* **EUF-CMA** *defined in Figure 9. We say that JS is EUF-CMA secure if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{A},\mathsf{JS}}(\lambda) := \Pr\left[\mathbf{EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{JS}}(\lambda) \Rightarrow 1\right].$$

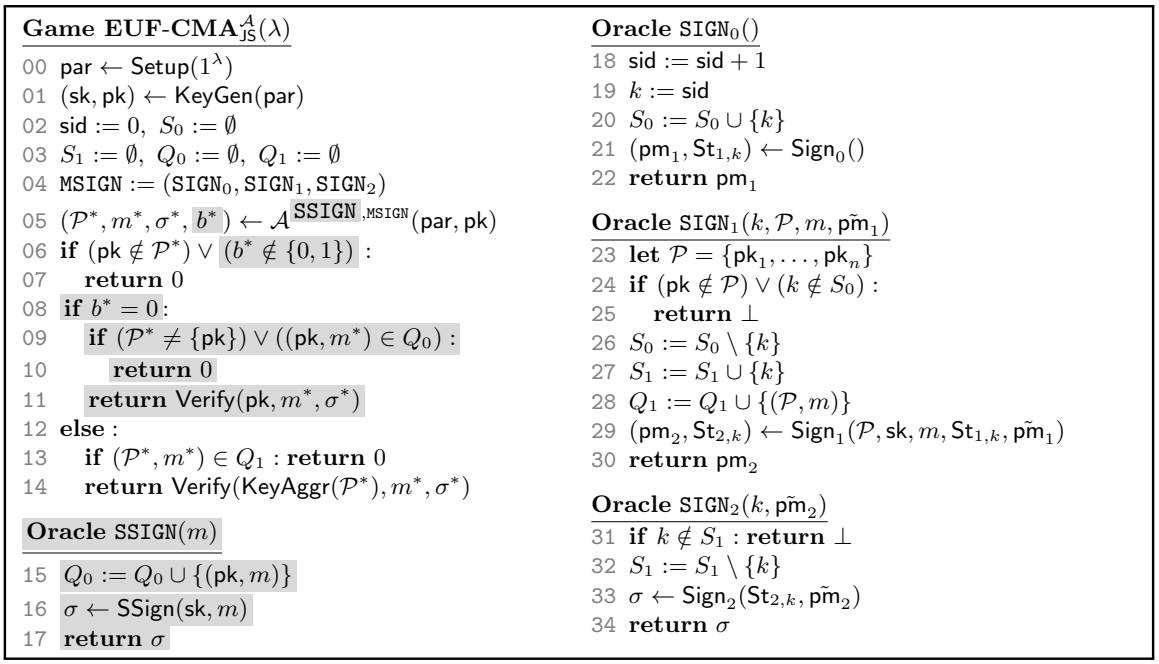


Fig. 9. The game **EUF-CMA** for a joint-signature scheme JS and adversary $\mathcal{A}$. We highlight differences to the **EUF-CMA** game [NRS21] for a multi-signature scheme in gray boxes.