Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites[1], Chelsea Komlo[2], and Mary Maller[3]

[1] Web3 Foundation
[2] University of Waterloo & Zcash Foundation
[3] Ethereum Foundation & PQShield, UK
elizabeth@web3.foundation, ckomlo@uwaterloo.ca, mary.maller@ethereum.org

Abstract. We prove adaptive security of a simple three-round threshold Schnorr signature scheme, which we call Sparkle+. The standard notion of security for threshold signatures considers a *static* adversary – one who must declare which parties are corrupt at the beginning of the protocol. The stronger *adaptive* adversary can at any time corrupt parties and learn their state. This notion is natural and practical, yet not proven to be met by most schemes in the literature.

In this paper, we demonstrate that Sparkle+ achieves several levels of security based on different corruption models and assumptions. To begin with, Sparkle+ is statically secure under minimal assumptions: the discrete logarithm assumption (DL) and the random oracle model (ROM). If an adaptive adversary corrupts fewer than $t/2$ out of a threshold of $t+1$ signers, then Sparkle+ is adaptively secure under a weaker variant of the one-more discrete logarithm assumption (AOMDL) in the ROM. We then prove that Sparkle+ achieves *full* adaptive security, with a corruption threshold of t, under AOMDL in the algebraic group model (AGM) with random oracles. Ours is the *first* proof for a threshold Schnorr signature scheme achieving the following desiderata: (i) adaptive security for a full t out of $t+1$ signers, for any $t < n$, (ii) no requirement of secure erasure of secret state, and (iii) less than $\binom{n}{t}$ security loss. More than that, our reduction for full adaptive security is tight. We achieve this using minimal network assumptions and only two online signing rounds with pre-processing.

Update: Recent work [33] shows that the adaptive security of Sparkle+, among other threshold Schnorr schemes, cannot hold for t corruptions without assuming the hardness of a search problem P defined in [33]. (The $t/2$ adaptive security of Sparkle+ is unaffected.) Later work [31] shows the same for a closely related problem, LDVR, defined in [31]. The FROST threshold Schnorr signature scheme is proven adaptively secure for t corruptions under the LDVR assumption [31]. We leave it to future work to explore whether Sparkle+ can be proven adaptively secure for t corruptions using similar techniques.

Table of Contents

1 Introduction 3
2 Related Work 6
3 Preliminaries 10
3.1 General Notation and Definitions 10
3.2 Digital Signatures 12
3.3 Assumptions 13
4 Threshold Signature Schemes 15
4.1 Definition of Static Security 18
4.2 Definition of Adaptive Security 19
5 Threshold Schnorr Signature Scheme Sparkle+ 19
5.1 Communication Model 20
5.2 Protocol Description 20
6 Our Results 23
6.1 Static Security Under Standard Assumptions 23
6.2 Adaptive Security up to $t/2$ Corruptions 24
6.3 Adaptive Security up to t Corruptions 26
7 Conclusion 27
A Proof of Static Security 32
B Proof of Adaptive Security up to $t/2$ Corruptions 41
C Proof of Adaptive Security up to t Corruptions 50

1 Introduction

A threshold signature scheme allows a set of n possible signers to jointly produce a signature on a message such that it verifies under a single public key, so long as at least a threshold $t + 1$ of signers participate. Importantly, a threshold signature scheme should remain unforgeable even if t signers are under adversarial control. A recent line of work has explored multi-party signatures whose output is a standard (single-party) Schnorr signature [58, 4, 52, 35, 54, 10, 56, 62, 29]. Schnorr signatures admit an efficient and compact representation even in the multi-party setting, which makes them of particular interest for practical use [30, 21].

Static vs. Adaptive Security. Most threshold signature schemes in the literature are proven secure under static corruptions. In the static setting, an adversary must declare which parties it wishes to corrupt in advance of any messages being sent. This model places an artificial restriction on the adversary's capabilities: in reality, malicious actors may observe a system before targeting specific parties. Thus, adaptive security is a strictly stronger notion, and indeed there are schemes that are statically but not adaptively secure [24]. While there are generic methods for transforming a statically secure scheme into an adaptively secure one [26], such as guessing the corrupted parties and aborting if incorrect, these methods incur undesirable performance overhead and a tightness loss of $\binom{n}{t}$. This is acceptable only for small numbers of parties, and adaptive security without $\binom{n}{t}$ tightness loss is challenging to achieve. A number of other techniques for proving adaptive security have been proposed but require undesirable tradeoffs, such as secure erasure of secret state [26, 41, 42] or a substantial amount of communication [49, 55, 2].

In this work, we investigate the adaptive security of a simple three-round threshold Schnorr signature scheme, which we call Sparkle+, under different corruption models and security assumptions. Achieving adaptive security is not just of theoretical interest: NIST recently published a call for multi-party threshold schemes and included adaptive security as a main goal [21, 22], ideally supporting up to $n = 1024$ or more parties. Our techniques are likely of independent interest, as this paper introduces the first proof for *any* threshold Schnorr signature scheme that allows a full t adaptive corruptions out of $t+1$ signers, for any $t < n$, without secure erasures or $\binom{n}{t}$ tightness loss. Moreover, the reduction for full adaptive security is tight.

Concurrent Security. A *concurrent* adversary may open an arbitrary number of signing sessions simultaneously. Unforgeability against a concurrent adversary is also a difficult property to achieve, and indeed a host of threshold, blind, and multi-signature schemes were demonstrated to be broken by concurrent (ROS) attacks first observed by Drijvers et al. [37] and exhibited in polynomial time by Benhamouda et al. [17]. Our security reductions for Sparkle+ hold against a concurrent and adaptive adversary. Moreover, Sparkle+ allows one round of signing to be pre-processed, before the message and signing set are determined. This is optimal, as further pre-processing would make the scheme susceptible to

Scheme	Offline Signing Rounds	Online Signing Rounds	Static Security Assumptions	Proof of Adaptive Security
Lindell22 [54]	-	3	Schnorr with aborts	✗
Classic S. [56]	-	3	Schnorr with aborts	$\binom{n}{t}$ security loss
Zero S. [56]	-	3	Schnorr with aborts	Requires erasures
FROST/2/3 [52, 35, 10, 62]	1	1	AOMDL	✗
Sparkle+	1	2	DL	✓

Fig. 1. Two- and three-round threshold Schnorr signature schemes that are concurrently secure in the random oracle model (ROM). Schnorr with aborts denotes simulation-based security with respect to Schnorr as the idealized functionality, with an adversary that can cause the protocol to abort.

ROS attacks. Combining concurrency and adaptivity in a multi-party, multi-round signing protocol is the main technical achievement of this work.

Static Security of Sparkle+. Sparkle+ is a three-round threshold Schnorr signature scheme that follows a commit-reveal paradigm. The protocol consists of an offline phase, which may be executed before the message or signing set is known, and an online phase consisting of two signing rounds. To begin with, we prove the static security of Sparkle+ from minimal assumptions: that the discrete logarithm assumption (DL) holds in the random oracle model (ROM). This is the same assumption and model for which Schnorr signatures themselves are proven secure [60]. We compare Sparkle+ with existing threshold Schnorr signature schemes in Tables 1 and 3.

Adaptive Security of Sparkle+. We next consider an adaptive adversary who may corrupt up to $t/2$ out of a threshold of $t+1$ parties over the course of the protocol. We prove that Sparkle+ is adaptively secure in the random oracle model under AOMDL, a weaker variant of the one-more discrete logarithm assumption formalized in [58]. In the $t+1$-aomdl game, the adversary is given as input an AOMDL challenge that is a vector of group elements of length $t+1$. It is also given access to a discrete logarithm oracle, which returns the discrete logarithm of a group element chosen by the adversary. To win the $t+1$-aomdl game, the adversary must output all $t+1$ discrete logarithms of its challenge, having queried its DL oracle a maximum of t times. The AOMDL assumption is stronger than the discrete logarithm assumption because of the adversary's ability to request for up to t discrete logarithm solutions before returning the $t+1$ discrete logarithms of its challenge. On the other hand, the AOMDL assumption is strictly weaker than standard OMDL [11] because the adversary only queries for the discrete logarithm of linear combinations of its challenge elements, which means the DL

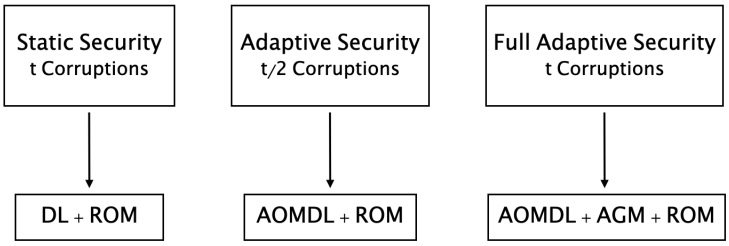

Fig. 2. Comparison of security models and assumptions for our threshold Schnorr signature scheme Sparkle+. The signing threshold is $t + 1$. DL is the discrete logarithm assumption, and AOMDL is the algebraic one-more discrete logarithm assumption. ROM is the random oracle model, and AGM is the algebraic group model.

---

oracle runs in polynomial time. This, in conjunction with the verification running in polynomial time, makes AOMDL a falsifiable assumption [9, 47, 51].

In the case where the adversary can corrupt up to a full t out of $t+1$ parties, it is not clear how to prove the adaptive security of Sparkle+ under AOMDL+ROM alone. The reason is that in order to extract an AOMDL solution, the adversary is rewound once, and there is no guarantee that the adversary will corrupt the same set of parties after the fork as it did during the first iteration of the protocol. When the adversary can corrupt only $t/2$ parties, this causes no issues, as the total number of corruptions over both iterations does not exceed t. If the adversary could corrupt more parties, the reduction would query its DL oracle more than t times and would lose the $t + 1$-aomdl game.

We thus look towards the algebraic group model (AGM) [43] for proving our strongest adaptivity result. The AGM assumes that whenever an adversary outputs a group element, it also outputs an algebraic representation specifying how the group element depends on previously seen values. Schnorr signatures were proven secure in the AGM in [44] to avoid the forking used in the original proof [60]. We use both techniques, to prove the adaptive security of Sparkle+ against $t/2$ corruptions in the ROM only, by rewinding, and to prove full adaptive security in the AGM and ROM. By full adaptive security, we mean that the adversary can adaptively corrupt up to the maximum of t out of $t + 1$ parties and still cannot forge. Our full adaptive security reduction is *straight-line*, i.e., does not rewind the adversary, and so avoids counting the number of corruptions over different forks of the adversary's execution.

Our Contributions. The contributions of this work are as follows:

- We introduce Sparkle+, an efficient and practical threshold Schnorr signature scheme with one round of pre-processing and two online signing rounds.
- We begin by proving that Sparkle+ is statically secure under the DL assumption in the ROM.

– We then demonstrate the adaptive security of Sparkle+ against up to $t/2$ corruptions under the AOMDL assumption in the ROM.
– Finally, we prove our main result: that Sparkle+ achieves full adaptive security, i.e., up to a full t corruptions, under the AOMDL assumption in the AGM and ROM.

2 Related Work

Threshold Schnorr Signatures. Closest to the design of Sparkle+ is the MSDL scheme presented by Boneh et al. [19], the three-round MuSig scheme by Maxwell et al. [57], and the 2Schnorr scheme by Nicolosi et al. [59]. However, MSDL and MuSig consider only the multi-signature setting (n-out-of-n), and 2Schnorr considers only the 2-out-of-2 setting. For multi-signatures, security holds against t corrupt parties out of $t + 1$ parties.

We now review threshold Schnorr schemes that are secure in the concurrent setting.

Stinson and Strobl [65] propose a threshold Schnorr signature scheme secure in the random oracle model under the discrete logarithm assumption. However, their scheme requires performing a three-round distributed key generation protocol (DKG) [46] to generate the nonce for each signature, which adds considerable network overhead: at a minimum, it requires participants to perform four rounds in total. Furthermore, the proof of security assumes only a static adversary.

Abe and Fehr [2] achieve a fully adaptively secure threshold Schnorr signature scheme also by using a three-round DKG for nonce generation. Security is modeled in the universally composable framework (UC) [23] and therefore captures a concurrent adversary. However, their scheme requires strong network assumptions (synchronous broadcast) and only achieves adaptive security in the honest majority setting ($t < n/2$). In contrast, Sparkle+ is secure in the asynchronous setting, does not assume broadcast or even authenticated channels, and is fully adaptively secure for any corruption threshold $t < n$.

Komlo and Goldberg [52] present a two-round threshold Schnorr signature FROST. Unlike prior threshold Schnorr schemes in the literature [45], FROST is secure against a concurrent adversary and is not susceptible to ROS attacks [17]. FROST2, proposed by Crites et al. [35] and refined by Bellare et al. [16, 10], is an optimized version of FROST that reduces the number of exponentiations required for signing from $t + 1$ to one. A further optimization of FROST2, called FROST3, is introduced by Ruffing et al. [62] and proven secure with a DKG by Chu et al. [29]. (See Table 3 for a comparison of efficiency.) However, all three of FROST, FROST2, FROST3 are only proven statically secure, under the algebraic one-more discrete logarithm assumption (AOMDL) [10]. While Sparkle+ adds an additional round of communication, it requires only a single exponentiation per signer and is proven statically secure under standard assumptions, which are also criteria of interest in [21, 22].

Lindell presents a three-round threshold Schnorr signature [54] with the goal of defining a scheme that is UC-secure in the random oracle model under the discrete

Scheme	Sign					Combine	
	Performance			Bandwidth		Performance	
	rounds	exp	H	$\mathbb{G}$	$\mathbb{F}$	exp	H
FROST [52]	2	$t+2$	$t+1$	2	1	t	t
FROST2 [10]	2	3	2	2	1	1	1
FROST3 [62]	2	3	2	1	1	1	1
Classic S. [56]	3	1	1	1	2	0	0
Lindell22 [54]	3	$11t+1$	$18t+10^7$	11	25	0	0
Sparkle+	3	1	$t+2$	1	2	0	0

Fig. 3. Efficiency of Two- and Three-Round Threshold Schnorr Signature Schemes. All output a standard Schnorr signature. We only compare schemes that are secure against ROS attacks [37, 17]. The number of network rounds between participants is given in the rounds column. exp stands for the number of group exponentiations. The total number of group and field elements sent by each signer is denoted by $\mathbb{G}$ and $\mathbb{F}$, respectively. H denotes the total number of hashing operations performed. The cost of signature verification is identical for each scheme, and is simply the cost of verifying a single Schnorr signature. Estimates for Lindell22 are made with respect to a 128-bit security level for Fischlin [40], where $r = 8$ is the number of commitments for a Fischlin proof and the length of the zero vector is $b = 16$, such that $b \cdot r = 128$.

logarithm assumption only. However, the security proof assumes the adversary is static. Sparkle+ similarly relies on only the ROM and DL assumption for static security, but does not require the use of online-extractable zero-knowledge proofs [40], and hence is both significantly more efficient and a simpler design.

Concurrent to this work, Makriyannis [56] defines a commit-reveal threshold Schnorr signature scheme similar to Sparkle+ (though without pre-processing), called Classic S., and proves security with respect to an idealized threshold signature functionality by Canetti et al. [25]. They consider adaptive security but employ the guessing argument that incurs $\binom{n}{t}$ tightness loss.

Adaptive Security of Threshold Signatures. While adaptive security for threshold schemes is a well-known topic in the literature, no proof of full adaptive security for any $t \leq n$ without $\binom{n}{t}$ tightness loss or secure erasures exists for a threshold Schnorr signature scheme in the concurrent setting. We now review several alternative signature schemes and techniques for achieving adaptive security.

Canetti et al. [26] and Frankel et al. [41, 42] independently introduce the "Single Inconsistent Player" (SIP) technique for transforming statically secure threshold schemes into adaptively secure schemes. Subsequent works [49, 55, 2] use these techniques to prove adaptive security under concurrent composition and without erasures. First, a secret key x is n-out-of-n shared between all parties $x = x_1 + \ldots + x_n$. Then, each x_i is further t-out-of-n shared. If a party does not participate in signing, then other parties can reconstruct their contribution.

In the simulation, a single party's secret x_i is not defined, and the simulation aborts if the ith party is corrupted. The SIP technique elegantly ensures that the simulation only aborts with constant probability such that a polynomial-time reduction can be repeated until successful. In contrast, our simulator can respond to corruption queries for any party.

Almansa et al. [3] present a threshold RSA scheme with proactive and adaptive security, but these results do not translate to the discrete logarithm setting. Libert et al. [53] propose a variant of the threshold BLS [18] scheme that is adaptively secure. However, it is incompatible with single-party BLS verification, an often-critical goal for threshold schemes in practice. Bacho and Loss [6] demonstrate the adaptive security of threshold BLS in Type I bilinear groups directly in the AGM from the $t+1$-omdl assumption. Their reduction is tight. Interestingly, they demonstrate that the $t+1$-omdl assumption is the minimum assumption under which threshold BLS can be proven adaptively secure.

We similarly rely on the $t+1$-(a)omdl assumption for the adaptive security of Sparkle+. In more detail, the $t+1$-(a)omdl assumption gives the reduction t-time access to an oracle that returns the discrete logarithm of a group element of its choosing. This oracle is used in both [6] and our reduction to obtain the *secret keys* of parties as they are corrupted, by querying on the public keys of the chosen parties. Since threshold BLS [6] is non-interactive, secret keys are the extent of the private information gained by the adversary through adaptive corruption. However, for interactive protocols, the reduction must not only simulate keys, but any *state* used during signing, and it must correctly simulate the corruption oracle for each round in the signing protocol (without erasing state, to avoid the assumption of secure erasures). Our technique for simulating state across three signing rounds and potentially many concurrent signing sessions, without relying on erasure of state, is the main technical achievement of this work.

Subsequent Work to Our Original Work [34]. Since our original work, there has been a surge of research on adaptive security of threshold schemes.

Das and Ren [36] show adaptive security of threshold BLS in Type III bilinear groups under the co-computational Diffie-Hellman assumption (co-CDH) and decisional Diffie-Hellman assumption (DDH) in the ROM.

Makriyannis [56] introduces a threshold Schnorr signature scheme, Zero S., which relies on Pedersen commitments, Fischlin proofs of knowledge [40], and an interactive protocol to identify misbehaving parties. Zero S. is proven adaptively secure in a straight-line manner with a reduction to DL but requires secure erasure of secret state.

Twinkle, by Bacho et al. [8], is a family of three-round threshold signature schemes. The first, efficient construction is a distributed Chaum-Pedersen proof of equality of discrete logarithms $X = g^x$ and $\mathsf{H}(m)^x$ [27] (vs. a proof of knowledge of the discrete logarithm of $X = g^x$, as in Schnorr signatures). Thus, signatures are Chaum-Pedersen outputs $(\mathsf{H}(m)^x, c, z)$, where (c, z) is a Schnorr signature. This scheme is proven fully adaptively secure under a one-more variant of the computational Diffie-Hellman assumption (OMCDH) in the ROM. The second, matrix construction is proven fully adaptively secure under the DDH assumption

in the ROM. The adaptive simulation does not allow pre-processing: the message, which is hashed in the first round, is given along with the signing set in the first round. Furthermore, neither scheme is compatible with Schnorr signatures.

Katsumata et al. [50] prove the full adaptive security of two threshold Schnorr schemes, Crackle and Snap, using rewinding and assuming the EUF-CMA security of an additional signature scheme. Adaptive security critically relies on partial public keys not being revealed, however, and neither scheme allows for identifiable abort. That is, there is no way to identify even a single malicious party when the signing protocol results in an invalid signature. Moreover, the schemes require five and four online signing rounds, respectively.

Chen [28] proves the full adaptive security of a three-round threshold Schnorr scheme based on FROST, called FROST-Mask, and a two-round scheme that requires (consistent) session identifiers, called HBTS-Mask. As techniques from [50] are used, both schemes inherit the lack of identifiable abort.

Glacius, by Bacho et al. [5], is a five-round threshold Schnorr signature scheme that is proven fully adaptively secure under DDH in the ROM using similar techniques to [36]. Its security relies on authenticated channels between all participants.

HARTS, by Bacho et al. [7], is a robust, constant-round threshold Schnorr protocol that achieves adaptive security in the $t/2$ setting under AOMDL in the ROM, assuming secure erasure of secret state and authenticated channels.

Crites et al. [32] prove a series of impossibility results for threshold signature schemes with keys of a common form, which they call key-unique schemes. In particular, they show that key-unique threshold Schnorr signature schemes cannot be proven adaptively secure above $t/2$ corruptions under (A)OMDL in the programmable ROM for a fully black-box reduction that rewinds the adversary and programs with fresh randomness on at least one random oracle query. This is exactly the type of reduction we employ in our $t/2$ adaptive security proof, confirming their result of a maximum corruption threshold of $t/2$.

Definitions. In this work, we employ a game-based approach to defining the static and adaptive security of a threshold signature, formalizing prior notions presented in the literature [53]. Alternative definitions of adaptive security in the UC setting have been proposed by Canetti et al. [25]. They prove their threshold ECDSA scheme adaptively secure assuming that ECDSA is secure, in addition to other non-interactive and falsifiable assumptions. However, their construction focuses on n-out-of-n multi-party signing, and their techniques critically do not translate to the $(t+1)$-out-of-n setting unless $\binom{n}{t}$ is small. Our fully adaptive $(t+1)$-out-of-n construction requires the algebraic group model, and incorporating algebraic adversaries into the UC setting is known to be a hard problem [1].

On the other hand, Lindell [54] shows that their protocol UC-realizes the Schnorr functionality with aborts, in the presence of an adversary that non-adaptively corrupts t parties. Secure evaluation of the Schnorr functionality is stronger than unforgeability. As noted in [25], it is arguably overly strong in the sense that it necessitates certain design decisions, such as incorporating online-extractable zero-knowledge proofs. Indeed, [54] elected for Fischlin proofs

(see Table 3). The only method to bias the nonces in both Sparkle+ and [54] is to abort; this is not the case in FROST [52]. However, all three works allow aborts, and therefore the distribution of the secret randomness cannot be considered uniform [38].

Differences to the Original Work [34]. The authors of [8] identified one class of adversarial behavior not captured by our original security analysis, namely that honest parties may not have the same view of commitments generated in the first signing round. This is fixed by having parties sign the view of the commitments they receive from other parties, using any EUF-CMA-secure signature scheme.[4] In particular, a Schnorr signature may be used, which we assume in Fig. 2 and Tables 1 and 3 since it is itself secure under the DL assumption in the ROM [60]. Signature-based authentication is used in other recent works, such as a version of FROST achieving a higher level of security (TS-UF-4) in the formalism of Bellare et al. [10], in the blind threshold signature scheme Snowblind [30], for identifiable abort in the five-round scheme Glacius [7], and for achieving adaptive security for the five- and four-round schemes Crackle and Snap [50]. In this work, we are able to achieve adaptive security for a Schnorr threshold signature scheme with only two online signing rounds (where the message and signing set deferred to the second round). We prove adaptive security of Sparkle+, with pre-processing, using our original simulation techniques.

3 Preliminaries

3.1 General Notation and Definitions

Let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ its unary representation. A function $\nu : \mathbb{N} \to \mathbb{R}$ is called *negligible* if for all $c \in \mathbb{R}, c > 0$, there exists $k_0 \in \mathbb{N}$ such that $|\nu(k)| < \frac{1}{k^c}$ for all $k \in \mathbb{N}, k \geq k_0$. For a non-empty set S, let $x \leftarrow\!\!\$\, S$ denote sampling an element of S uniformly at random and assigning it to x. We use $[n]$ to represent the set $\{1, \ldots, n\}$ and $[0..n]$ to represent the set $\{0, \ldots, n\}$. We represent vectors as $\vec{a} = (a_1, \ldots, a_n)$.

Let PPT denote probabilistic polynomial time. Algorithms are randomized unless explicitly noted otherwise. Let $y \leftarrow A(x; \rho)$ denote running algorithm A on input x and randomness ρ and assigning its output to y. Let $y \leftarrow\!\!\$\, A(x)$ denote $y \leftarrow A(x; \rho)$ for a uniformly random ρ. The set of values that have non-zero probability of being output by A on input x is denoted by $[A(x)]$.

Group Generation. Let GrGen be a polynomial-time algorithm that takes as input a security parameter 1^κ and outputs a group description $(\mathbb{G}, p, g)$ consisting of a group $\mathbb{G}$ of order p, where p is a κ-bit prime, and a generator g of $\mathbb{G}$.

[4] We place minimal requirements on the communication network for signing to give practitioners the flexibility to implement Sparkle+ in myriad settings, without the risk of compromising security. See Section 5.1 for more details on our communication model.

Random Oracle Model [14]. The random oracle model is an idealized model that treats a hash function H as an oracle in the following way. When queried on an input in the domain of H, the oracle first checks if it has an entry stored in its table for this input. If so, it returns this value. If not, it samples an output in the codomain of H uniformly at random, stores the input-output pair in its table, and returns the output.

Algebraic Group Model [43]. The algebraic group model is an idealized model that places the following restriction on the computation of the adversary. An adversary is *algebraic* if for every group element $Z \in \mathbb{G} = \langle g \rangle$ that it outputs, it is required to output a representation $\vec{a} = (a_0, a_1, a_2, \dots)$ such that $Z = g^{a_0} \prod {Y_i}^{a_i}$, where $Y_1, Y_2, \dots \in \mathbb{G}$ are group elements that the adversary has seen thus far. Intuitively, this captures the notion that an algorithm should know how it computes its outputs from the values it has received as input so far. The AGM is reminiscent of the generic group model (GGM), but lies somewhere between it and the standard model.

Polynomial Interpolation. A polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_t x^t$ of degree t over a field $\mathbb{F}$ can be interpolated by $t+1$ points. Let $\eta \subseteq [n]$ be the list of $t+1$ distinct indices corresponding to the x-coordinates $x_i \in \mathbb{F}, i \in \eta$, of these points. Then the Lagrange polynomial $L_i(x)$ has the form:

$$L_i(x) = \prod_{j \in \eta; j \neq i} \frac{x - x_j}{x_i - x_j} \tag{1}$$

Given a set of $t+1$ points $(x_i, f(x_i))_{i \in [t+1]}$, any point $f(x_\ell)$ on the polynomial f can be determined by Lagrange interpolation as follows:

$$f(x_\ell) = \sum_{k \in \eta} f(x_k) \cdot L_k(x_\ell)$$

Definition 1 (Shamir Secret Sharing [64]). *Shamir secret sharing is an $(n, t+1)$-threshold secret sharing scheme consisting of algorithms* ($\mathsf{IssueShares}$, $\mathsf{Recover}$)*, defined as follows:*

- $\mathsf{IssueShares}(x, n, t+1) \to \{(1, x_1), \dots, (n, x_n)\}$*: On input a secret x, number of participants n, and threshold $t+1$, perform the following. First, define a polynomial $f(Z) = x + a_1 Z + a_2 Z^2 + \cdots + a_t Z^t$ by sampling t coefficients at random: $a_1, \dots, a_t \leftarrow_\$ \mathbb{Z}_p$. Then, set each participant's share $x_i, i \in [n]$, to be the evaluation of $f(i)$:*
$$x_i \leftarrow x + \sum_{j \in [t]} a_j i^j$$
Output $\{(i, x_i)\}_{i \in [n]}$.
- $\mathsf{Recover}(t+1, \{(i, x_i)\}_{i \in \mathcal{S}}) \to \perp / x$*: On input threshold $t+1$ and a set of shares $\{(i, x_i)\}_{i \in \mathcal{S}}$, output $\perp$ if $\mathcal{S} \not\subseteq [n]$ or if $|\mathcal{S}| < t+1$. Otherwise, recover x as follows:*
$$x \leftarrow \sum_{i \in \mathcal{S}} \lambda_i x_i$$

where the Lagrange coefficient for the set $\mathcal{S}$ is defined by

$$\lambda_i = \prod_{j \in \mathcal{S}, j \neq i} \frac{j}{j - i}$$

3.2 Digital Signatures

Definition 2 (Digitial Signatures). *A digitial signature scheme* DS *consists of polynomial-time algorithms* $(\mathsf{DS.Setup}, \mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$*, defined as follows:*

- $\mathsf{DS.Setup}(1^\kappa) \to \mathsf{par}_{\mathsf{DS}}$*: On input a security parameter* 1^κ*, this algorithm outputs public parameters* $\mathsf{par}_{\mathsf{DS}}$ *(which are given implicitly as input to all other algorithms).*
- $\mathsf{DS.KeyGen}() \to (\hat{\mathsf{pk}}, \hat{\mathsf{sk}})$*: This probabilistic algorithm outputs a public key* $\hat{\mathsf{pk}}$ *and secret key* $\hat{\mathsf{sk}}$*.*
- $\mathsf{DS.Sign}(\hat{\mathsf{sk}}, \mathsf{msg}) \to \hat{\sigma}$*: On input a secret key* $\hat{\mathsf{sk}}$ *and a message* msg*, this algorithm outputs a signature* $\hat{\sigma}$*.*
- $\mathsf{DS.Verify}(\hat{\mathsf{pk}}, \mathsf{msg}, \hat{\sigma}) \to 0/1$*: On input a public key* $\hat{\mathsf{pk}}$*, a message* msg*, and a purported signature* $\hat{\sigma}$*, this deterministic algorithm outputs* 1 *(accept) if* $\hat{\sigma}$ *is a valid signature on* msg *under* $\hat{\mathsf{pk}}$*; else, it outputs* 0 *(reject).*

A digital signature scheme must be *correct*. It is said to be *secure* if it is existentially unforgeable under chosen-message attacks (EUF-CMA).

Definition 3 (Correctness). *A digital signature scheme* $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *is* correct *if for all security parameters* $\kappa \in \mathbb{N}$*, all key pairs* $(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \in [\mathsf{DS.KeyGen}()]$*, and all messages* msg*, we have:*

$$\Pr[\mathsf{DS.Verify}(\hat{\mathsf{pk}}, \mathsf{msg}, \mathsf{DS.Sign}(\hat{\mathsf{sk}}, \mathsf{msg})) = 1] = 1$$

Definition 4 (EUF-CMA [48]). *Let the advantage of an adversary* $\mathcal{A}$ *playing the EUF-CMA game,* $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\kappa)$*, as defined in Figure 4, be as follows:*

$$\mathit{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\kappa) = \Pr[\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\kappa) = 1]$$

A digital signature scheme $\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *is* existentially unforgeable under chosen-message attacks *if for all PPT adversaries* $\mathcal{A}$*,* $\mathit{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\kappa)$ *is negligible.*

Definition 5 (Schnorr Signatures [63]). *The Schnorr signature scheme consists of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, defined as follows:*

MAIN $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{DS}}(\kappa)$	$\mathcal{O}^{\mathsf{DS.Sign}}(\mathsf{msg})$
$\mathsf{par}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$	$\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}\}$
$\mathsf{Q}_{\mathsf{msg}} \leftarrow \emptyset$	$\hat{\sigma} \leftarrow \mathsf{DS.Sign}(\hat{\mathsf{sk}}, \mathsf{msg})$
$(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}) \leftarrow\!\!\$\ \mathsf{DS.KeyGen}()$	**return** $\hat{\sigma}$
$(\mathsf{msg}^*, \hat{\sigma}^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{DS.Sign}}}(\hat{\mathsf{pk}})$	
return 0 **if** $\mathsf{msg}^* \in \mathsf{Q}_{\mathsf{msg}}$	
$\quad \vee\ \mathsf{DS.Verify}(\hat{\mathsf{pk}}, \mathsf{msg}^*, \hat{\sigma}^*) \neq 1$	
return 1	

Fig. 4. The EUF-CMA security game for a digital signature scheme DS. The public parameters $\mathsf{par}_{\mathsf{DS}}$ are implicitly given as input to all algorithms.

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: On input a security parameter* 1^κ*, run* $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$ *and select a hash function* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$*. Output public parameters* $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H})$ *(which are given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$*: Sample a secret key* $x \leftarrow\!\!\$\ \mathbb{Z}_p$ *and compute the public key as* $X \leftarrow g^x$*. Output key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow (X, x)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*: On input a secret key* $\mathsf{sk} = x$ *and a message* m*, sample a nonce* $r \leftarrow\!\!\$\ \mathbb{Z}_p$*. Then, compute a nonce commitment* $R \leftarrow g^r$*, the challenge* $c \leftarrow \mathsf{H}(X, m, R)$*, and the response* $z \leftarrow r + cx$*. Output a signature* $\sigma \leftarrow (R, z)$*.*
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 0/1$*: On input a public key* $\mathsf{pk} = X$*, a message* m*, and a purported signature* $\sigma = (R, z)$*, compute* $c \leftarrow \mathsf{H}(X, m, R)$ *and output* 1 *(accept) if* $R \cdot X^c = g^z$*; else, output* 0 *(reject).*

A Schnorr signature is a Sigma protocol zero-knowledge proof of knowledge (as in Boneh–Shoup, §19.4 [20]) of the discrete logarithm of the public key X, made non-interactive and bound to the message m by the Fiat-Shamir transform [39]. Schnorr signatures are secure under the discrete logarithm assumption in the random oracle model [60].

3.3 Assumptions

Assumption 1 (Discrete Logarithm Assumption (DL)) *Let* GrGen *be a group generator that outputs* $(\mathbb{G}, p, g)$*. Let the advantage of an adversary* $\mathcal{A}$ *playing the discrete logarithm game,* $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{A}}(\kappa)$*, as defined in Figure 5, be as follows:*

$$\mathit{Adv}^{\mathsf{dl}}_{\mathcal{A}}(\kappa) = \Pr[\mathsf{Game}^{\mathsf{dl}}_{\mathcal{A}}(\kappa) = 1]$$

The discrete logarithm assumption holds relative to GrGen *if for all PPT adversaries* $\mathcal{A}$*,* $\mathit{Adv}^{\mathsf{dl}}_{\mathcal{A}}(\kappa)$ *is negligible.*

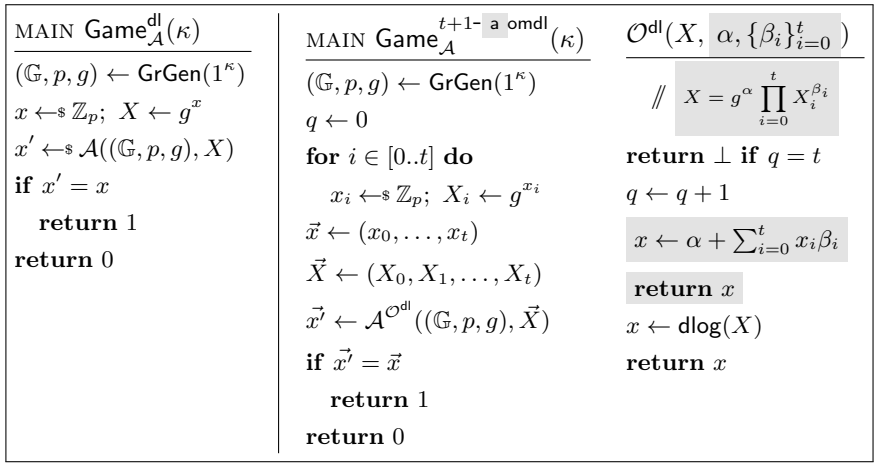

Fig. 5. The Discrete Logarithm (DL), One-More Discrete Logarithm (OMDL), and Algebraic One-More Discrete Logarithm (AOMDL) games. The differences between the OMDL and AOMDL games are highlighted in gray. The key distinction is that in the AOMDL game, the adversary can only query its discrete logarithm oracle on linear combinations of its challenge group elements; in the OMDL game, the environment must return the discrete logarithm of an arbitrary group element. dlog is an algorithm that finds the discrete logarithm of an arbitrary group element X base g.

Assumption 2 (One-More Discrete Logarithm Assumption (OMDL)) [11] *Let* GrGen *be a group generator that outputs* $(\mathbb{G}, p, g)$*. Let the advantage of an adversary* $\mathcal{A}$ *playing the* $t+1$*-one-more discrete logarithm game,* $\mathsf{Game}^{t+1\text{-}\mathsf{omdl}}_{\mathcal{A}}(\kappa)$*, as defined in Figure 5, be as follows:*

$$\mathit{Adv}^{t+\,1\text{-}\mathsf{omdl}}_{\mathcal{A}}(\kappa) = \Pr[\mathsf{Game}^{t+1\text{-}\mathsf{omdl}}_{\mathcal{A}}(\kappa) = 1]$$

The one-more discrete logarithm assumption holds relative to GrGen *if for all PPT adversaries* $\mathcal{A}$*,* $\mathit{Adv}^{t+\,1\text{-}\mathsf{omdl}}_{\mathcal{A}}(\kappa)$ *is negligible.*

Algebraic One-More Discrete Logarithm Assumption (AOMDL). The algebraic one-more discrete logarithm assumption (AOMDL) was introduced formally by Jonas, Ruffing, and Seurin for proving the security of the two-round Schnorr multi-signature scheme MuSig2 [58]. We use it as an assumption to prove the adaptive security of $\mathsf{Sparkle}$+, but it actually (implicitly) appears elsewhere in the literature where OMDL is used [13, 59, 15, 44].

As in the standard OMDL game, the AOMDL adversary is given as input a group description $(\mathbb{G}, p, g)$ and a set of challenge group elements $(X_0, X_1, \ldots, X_t)$ and has access to a discrete logarithm solution oracle $\mathcal{O}^{\mathsf{dl}}$ that it may query up to t times. The adversary wins the game if it computes the discrete logarithms $(x_0, x_1, \ldots, x_t)$ without exceeding t queries to $\mathcal{O}^{\mathsf{dl}}$ (i.e., the adversary is able to

compute "one more" discrete logarithm). The difference in the AOMDL game is that, when the adversary queries the $\mathcal{O}^{\mathsf{dl}}$ oracle on a group element X, it must provide an algebraic representation $(\alpha, \{\beta_i\}_{i=0}^t)$ of X relative to the generator g and challenge group elements $(X_0, X_1, \ldots, X_t)$, so that $X = g^\alpha \prod_{i=0}^t X_i^{\beta_i}$. The $\mathcal{O}^{\mathsf{dl}}$ oracle may then use this representation to compute $\alpha + \sum_{i=0}^t \beta_i x_i$. This is in contrast to standard OMDL, where the $\mathcal{O}^{\mathsf{dl}}$ oracle may be queried for discrete logarithms of arbitrary group elements. Thus, AOMDL is a weaker assumption than standard OMDL, because it is falsifiable. Indeed, not only does the algorithm verifying the correctness of the AOMDL solutions run in polynomial time, but the $\mathcal{O}^{\mathsf{dl}}$ algorithm does as well.

Assumption 3 (Algebraic One-More Discrete Logarithm Assumption) [58] *Let* GrGen *be a group generator that outputs* $(\mathbb{G}, p, g)$*. Let the advantage of an adversary* $\mathcal{A}$ *playing the* $t+1$*-algebraic one-more discrete logarithm game,* $\mathsf{Game}_{\mathcal{A}}^{t+1\text{-}\mathsf{aomdl}}(\kappa)$*, as defined in Figure 5, be as follows:*

$$\mathsf{Adv}_{\mathcal{A}}^{t+1\text{-}\mathsf{aomdl}}(\kappa) = \Pr[\mathsf{Game}_{\mathcal{A}}^{t+1\text{-}\mathsf{aomdl}}(\kappa) = 1]$$

The algebraic one-more discrete logarithm assumption holds relative to GrGen *if for all PPT adversaries* $\mathcal{A}$*,* $\mathsf{Adv}_{\mathcal{A}}^{t+1\text{-}\mathsf{aomdl}}(\kappa)$ *is negligible.*

4 Threshold Signature Schemes

We begin with the definition of a threshold signature scheme. Our syntax assumes a centralized key generation algorithm to generate the threshold public key pk and key pairs $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}$ for all n parties. However, our construction and proofs can be adapted to use a distributed key generation protocol (DKG). We refer to [6] for a discussion on how to achieve adaptively secure DKGs. Outputting public key shares is also optional.

Our definition models a three-round signing protocol, where the message and signing set are given in the first round. We then provide game-based definitions of static and adaptive security for threshold signature schemes with pre-processing (i.e., where the message and signing set are deferred to the second round), as these are the notions of security achieved by $\mathsf{Sparkle}+$.

Definition 6 (Threshold Signatures). *A threshold signature scheme* TS *with three signing rounds consists of polynomial-time algorithms* $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, (\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3), \mathsf{Combine}, \mathsf{Verify})$*, defined as follows:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: This algorithm takes as input a security parameter* 1^κ *and outputs public parameters* par *(which are given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}(n, t+1) \to (\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]})$*: A probabilistic algorithm that takes as input the number of signers* n *and the threshold* $t+1$ *and outputs the public key* pk *representing the set of* n *signers, the set* $\{\mathsf{pk}_i\}_{i\in[n]}$ *of public keys for each signer, and the set* $\{\mathsf{sk}_i\}_{i\in[n]}$ *of secret keys for each signer. It is assumed that participant* i *is sent its secret key* sk_i *privately (see Section 5.1).*

$\mathsf{TSignHon}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[\mathcal{S}]}, \mathcal{S}, m)$

return $\perp$ **if** $\mathcal{S} \notin [n] \wedge |S| < t+1$
for $k \in \mathcal{S}$ **do** $(\mathsf{pm}_{k,1}, \mathsf{st}_{k,1}) \leftarrow \mathsf{Sign}_1(k, \mathcal{S}, m, \mathsf{sk}_k)$
for $k \in \mathcal{S}$ **do** $(\mathsf{pm}_{k,2}, \mathsf{st}_{k,2}) \leftarrow \mathsf{Sign}_2(k, \mathsf{st}_{k,1}, \{\mathsf{pm}_{i,1}\}_{i\in\mathcal{S}})$
for $k \in \mathcal{S}$ **do** $\mathsf{pm}_{k,3} \leftarrow \mathsf{Sign}_3(k, \mathsf{st}_{k,2}, \{\mathsf{pm}_{i,2}\}_{i\in\mathcal{S}})$
$\sigma \leftarrow \mathsf{Combine}(\mathcal{S}, m, \{\mathsf{pm}_{i,1}, \mathsf{pm}_{i,2}, \mathsf{pm}_{i,3}\}_{i\in\mathcal{S}})$
return σ

Fig. 6. An algorithm $\mathsf{TSignHon}$ for honestly generating a threshold signature in a three-round signing protocol.

– $(\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3)$: *A set of algorithms where each subsequent algorithm represents a single stage in an interactive three-round signing protocol, performed by each signing party in a signing set $\mathcal{S} \subseteq [n], |\mathcal{S}| \geq t+1$, with respect to a message m, defined as follows:*

$$\begin{aligned}(\mathsf{pm}_{k,1}, \mathsf{st}_{k,1}) &\leftarrow \mathsf{Sign}_1(k, \mathcal{S}, m, \mathsf{sk}_k)\\ (\mathsf{pm}_{k,2}, \mathsf{st}_{k,2}) &\leftarrow \mathsf{Sign}_2(k, \mathsf{st}_{k,1}, \{\mathsf{pm}_{i,1}\}_{i\in\mathcal{S}})\\ \mathsf{pm}_{k,3} &\leftarrow \mathsf{Sign}_3(k, \mathsf{st}_{k,2}, \{\mathsf{pm}_{i,2}\}_{i\in\mathcal{S}})\end{aligned} \tag{2}$$

where $\mathsf{pm}_{k,1}, \mathsf{pm}_{k,2}, \mathsf{pm}_{k,3}$ are protocol messages sent by party k, and $\mathsf{st}_{k,1}, \mathsf{st}_{k,2}$ is the state of party k at the end of each round.

– $\mathsf{Combine}(\mathcal{S}, m, \{\mathsf{pm}_{i,1}, \mathsf{pm}_{i,2}, \mathsf{pm}_{i,3}\}_{i\in\mathcal{S}}) \to \sigma$*: A deterministic algorithm that takes as input a signing set $\mathcal{S}$, a message m, and a set of protocol messages $\{\mathsf{pm}_{i,1}, \mathsf{pm}_{i,2}, \mathsf{pm}_{i,3}\}_{i\in\mathcal{S}}$ and outputs a signature σ.*

– $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 0/1$*: A deterministic algorithm that takes as input the public key pk, a message m, and purported signature σ and outputs 1 (accept) if σ is a valid signature on m under pk; else, it outputs 0 (reject).*

Remark 1 (Combine). The $\mathsf{Combine}$ algorithm may be performed by one of the signers or some external party.

Correctness. A threshold signature scheme is *correct* if for all security parameters $\kappa \in \mathbb{N}$, all $\mathsf{par} \in [\mathsf{Setup}(1^\kappa)]$, all $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) \in [\mathsf{KeyGen}(n, t+1)]$, all $\mathcal{S} \subseteq [n]$ such that $|\mathcal{S}| \geq t+1$, and all messages m, we have:

$$\Pr[\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1 \mid \sigma \leftarrow \mathsf{TSignHon}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[\mathcal{S}]}, \mathcal{S}, m)] = 1$$

where the algorithm $\mathsf{TSignHon}$ is as defined in Fig. 6, which models an honest execution of the signing protocol.

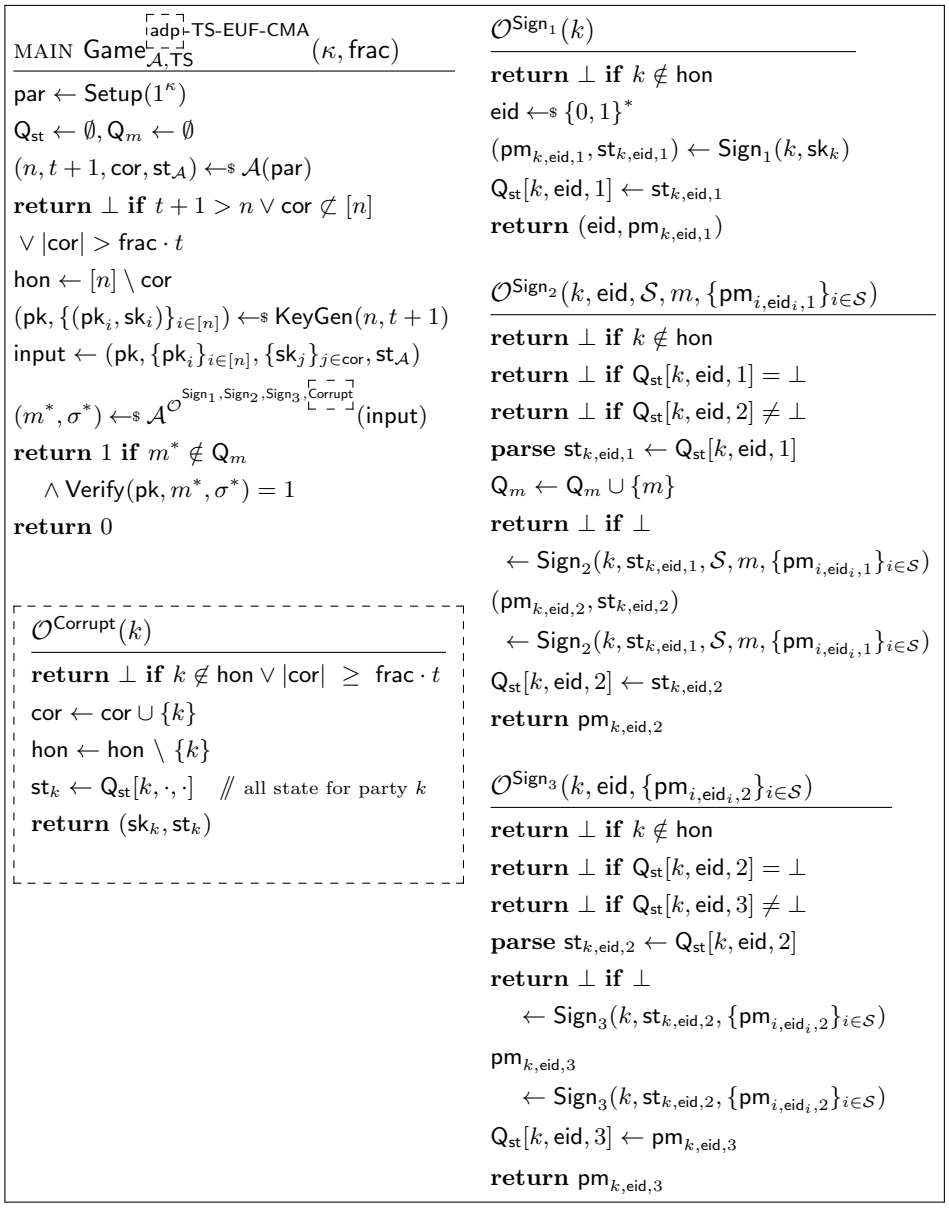

Fig. 7. Static and adaptive unforgeability games for a threshold signature scheme with one pre-processing round followed by two online signing rounds. The public parameters par are implicitly given as input to all algorithms, and $\mathsf{pm}_1, \mathsf{pm}_2, \mathsf{pm}_3$ represent protocol messages defined within the construction. The static game contains all but the dashed boxes, and the adaptive game adds the dashed boxes. frac specifies the fraction of t signers that the adversary may corrupt. For example, $\mathsf{frac} = 1$ means the adversary can corrupt a full t signers, and $\mathsf{frac} = 1/2$ means it can corrupt $t/2$ signers.

4.1 Definition of Static Security

Most threshold signature schemes in the literature are proven in the *static* security model, which captures a limited adversary that can only corrupt parties at the onset of the protocol. We provide a formal game-based definition in Fig. 7. The static security game contains all but the dashed boxes.

In the static unforgeability game, the environment accepts a security parameter κ and a corruption fraction frac. We consider a static adversary that can corrupt a full t (i.e., $\mathsf{frac} = 1$) signers out of $t + 1$ signers in a session. The environment generates public parameters par and initializes the adversary with par. The adversary chooses a set of potential signers n, a threshold $t + 1$, and a set of corrupted parties cor. The environment checks that the number of corrupted parties does not exceed the maximum allowable amount $\mathsf{frac} \cdot t$ and sets the honest parties $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$.

The environment then runs KeyGen to derive the threshold public key pk representing all n signers, the individual public key shares $\{\mathsf{pk}_i\}_{i\in[n]}$, and the secret signing shares $\{\mathsf{sk}_i\}_{i\in[n]}$. It returns pk, $\{\mathsf{pk}_i\}_{i\in[n]}$, and the set of corrupt signing shares $\{\mathsf{sk}_j\}_{j\in\mathsf{cor}}$ to the adversary.

After key generation has concluded, the adversary can then query the various signing oracles $\mathcal{O}^{\mathsf{Sign}_1,\mathsf{Sign}_2,\mathsf{Sign}_3}$ for honest signers to engage in subsequent signing rounds, across different sessions. The environment performs checks during the signing rounds to ensure each participant being called is in fact uncorrupted and in the correct signing round. Each check is carried out with respect to an execution identifier, denoted by eid, which is a random string sampled by the environment and bound to each new signing session for an honest party. Doing so enables the environment to distinguish concurrent signing sessions and update the correct session state for each subsequent signing round. Execution identifiers are akin to session identifiers (typically denoted as sid), however, we do *not* assume global consistency among signers for execution identifiers; each honest party samples and locally maintains their own eid for each signing session.

The adversary wins if it can produce a valid forgery $\sigma^* = (R^*, z^*)$ with respect to the threshold public key pk on a message m^* that has not been previously queried.

Note that our definition adds new state for each signing round. Specifically, the private state of signer k in session eid is stored in $\mathsf{st}[k, \mathsf{eid}, \mathsf{rnd}]$, where rnd is the round. The adaptive security of $\mathsf{Sparkle}+$ does not rely on secure erasures: the state in each round is carried to all subsequent rounds.

We place minimal requirements on the communication network and do not assume pairwise authenticated channels between all parties (see Section 5.1). Thus, an adversary may send inconsistent protocol messages to honest parties, even on behalf of other honest parties.

The adversary may be *rushing*, i.e., it may wait to produce its outputs after having seen honest protocol messages in each round. Our model additionally captures a *concurrent* adversary, who may open simultaneous signing sessions at once or choose not to complete a signing session. We model concurrency explicitly

in our definition to ensure that our notion of security protects against practical attacks which can occur under a concurrent adversary [37, 17].

Definition 7 (Static Security). *Let the advantage of an adversary $\mathcal{A}$ playing the static security game,* $\mathsf{Game}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac})$*, as defined in Figure 7, be as follows:*

$$\mathit{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac}) = \Pr[\mathsf{Game}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac}) = 1]$$

A threshold signature scheme TS *is* statically secure *if for all PPT adversaries $\mathcal{A}$,* $\mathit{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac})$ *is negligible.*

4.2 Definition of Adaptive Security

We build upon the notion of adaptive security for threshold signature schemes by Libert, Joye, and Yung [53]. We provide a formal game-based definition, which is specified in Figure 7. The adaptive unforgeability game contains the same inputs and algorithms as the static game, but additionally includes a corruption oracle $\mathcal{O}^{\mathsf{Corrupt}}$.

In the adaptive setting, the adversary is not restricted to choosing its set of corrupt participants cor only at the beginning of the game. Instead, the adversary can at any time choose to corrupt an honest party by querying $\mathcal{O}^{\mathsf{Corrupt}}$, receiving in return the honest party's secret key and state across all signing sessions.

In addition to producing a valid forgery, the adversary must meet the winning condition that the set of corrupted participants at the end of the experiment is within the expected bound, i.e., less than $\mathsf{frac} \cdot t$, with respect to the corruption ratio frac. An adaptive adversary may, for example, corrupt $t/2$ (i.e., $\mathsf{frac} = 1/2$) or a full t (i.e., $\mathsf{frac} = 1$) signers out of at least $t + 1$ signers in a session.

Definition 8 (Adaptive Security). *Let the advantage of an adversary $\mathcal{A}$ playing the adaptive security game* $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac})$*, as defined in Figure 7, be as follows:*

$$\mathit{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac}) = \Pr[\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac}) = 1]$$

A threshold signature scheme TS *is* adaptively secure *if for all PPT adversaries $\mathcal{A}$,* $\mathit{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac})$ *is negligible.*

Definition 9 (Full Adaptive Security). *We say that a threshold signature scheme* TS *achieves* full *adaptive security if the value of the multiplier* frac *in* $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(\kappa, \mathsf{frac})$ *is* $\mathsf{frac} = 1$*, i.e., the adversary is allowed to corrupt up to a full t participants.*

5 Threshold Schnorr Signature Scheme Sparkle+

Sparkle+ is a simple three-round threshold Schnorr signature scheme (Fig. 8). It consists of an offline phase, which may be executed before the message or signing

set are known, and an online phase consisting of two signing rounds. We assume an external mechanism to choose the set of signers $\mathcal{S} \subseteq [n]$, where $t+1 = |\mathcal{S}| \leq n$ and $\mathcal{S}$ is ordered to ensure consistency. We employ a centralized key generation algorithm; however, our scheme and proofs can be adapted to use a distributed key generation protocol (DKG). We refer to [6] for a discussion on how to achieve adaptively secure DKGs.

Intuitively, Sparkle+ follows a commit-reveal paradigm for its three-round signing protocol. In the first round, each participant in the signing set $\mathcal{S}$ commits to their nonce $R_i = g^{r_i}$ by publishing $\mathsf{cm}_i \leftarrow \mathsf{H}_{\mathsf{cm}}(i, R_i)$. In the second round, each participant reveals R_i in the clear, along with a DS signature on their protocol transcript. In the third round, each participant computes the aggregate nonce $R = \prod_{i \in \mathcal{S}} R_i$, the Schnorr challenge $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$ using R, and their signature share z_i. The signature shares are additively combined via the Combine algorithm to produce the Schnorr signature $\sigma = (R, z = \sum_{i \in \mathcal{S}} z_i)$.

This commit-reveal strategy ensures two security properties. First, requiring each participant to publish a commitment in the first round, before revealing their nonce R_i, prevents a rushing adversary from adaptively choosing their R_j as a function of other participants' R_i values, which is known to lead to ROS attacks [37, 17]. Second, as we will see later in the proofs of adaptive security, it allows the reduction to effectively handle corruptions at any point in the signing process, without requiring the erasure of secret state.

5.1 Communication Model

We assume that trusted key generation operations are performed over a reliable, authenticated, and private network communication channel. During key generation, parties may have authenticated channels with a coordinator, but are not assumed to have them with each other.

For communication between participants at the time of signing, we assume only that the network is reliable and do not assume consistency or authenticity of messages exchanged between participants. Thus, Sparkle+ additionally makes use of a standard EUF-CMA-secure digital signature scheme DS to authenticate messages sent in the signing rounds. We place minimal requirements on the communication network for signing to give practitioners the flexibility to implement Sparkle+ in myriad settings, without the risk of compromising security. This could result in some redundancy if, for example, there are pairwise authenticated channels between all parties.

5.2 Protocol Description

Parameter Generation. On input the security parameter 1^κ, the setup algorithm runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$ and selects two hash functions $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$. It also runs the setup algorithm for a signature scheme $\mathsf{par}_{\mathsf{sig}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$ used for authentication in Signing Rounds 1 and 2. It outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}, \mathsf{par}_{\mathsf{sig}})$, which are provided as input to all other algorithms and protocols.

$\mathsf{Setup}(1^\kappa)$

$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

// select two hash functions

$\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$

$\mathsf{par}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$

$\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}, \mathsf{par}_{\mathsf{DS}})$

return par

$\mathsf{KeyGen}(n, t+1)$

$x \leftarrow\!\!\$\ \mathbb{Z}_p;\ \mathsf{pk} \leftarrow g^x$

$\{(i, x_i)\}_{i\in[n]} \leftarrow\!\!\$\ \mathsf{IssueShares}(x, n, t+1)$

// Shamir secret sharing of x

for $i \in [n]$ **do**

$X_i \leftarrow g^{x_i};\ (\hat{X}_i, \hat{x}_i) \leftarrow\!\!\$\ \mathsf{DS.KeyGen}()$

$\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$

$\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i, \mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$

return $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]})$

$\mathsf{Sign}_1(k, \mathsf{sk}_k)$

$r_k \leftarrow\!\!\$\ \mathbb{Z}_p;\ R_k \leftarrow g^{r_k}$

$\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(k, R_k)$

$\mathsf{st}_{k,1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$

return $(\mathsf{cm}_k, \mathsf{st}_{k,1})$

$\mathsf{Sign}_2(k, \mathsf{st}_{k,1}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$

return $\perp$ **if** $\mathcal{S} \not\subseteq [n] \vee |\mathcal{S}| \leq t+1$
$\vee\ k \notin \mathcal{S}$

parse $(\mathsf{cm}'_k, R_k, r_k, \mathsf{sk}_k) \leftarrow \mathsf{st}_{k,1}$

// checks inputs against records in state

return $\perp$ **if** $\mathsf{cm}'_k \neq \mathsf{cm}_k$

$\mathsf{msg}_k \leftarrow (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$

$\hat{\sigma}_k \leftarrow \mathsf{DS.Sign}(\hat{x}_k, \mathsf{msg}_k)$

$\mathsf{st}_{k,2}$
$\leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \mathsf{sk}_k)$

return $((R_k, \hat{\sigma}_k), \mathsf{st}_{k,2})$

$\mathsf{Sign}_3(k, \mathsf{st}_{k,2}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$

parse
$(R'_k, r_k, \mathsf{cm}_k, \hat{\sigma}'_k, \mathcal{S}', m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}'}, \mathsf{sk}_k)$
$\leftarrow \mathsf{st}_{k,2}$

return $\perp$ **if**
$(R'_k, \hat{\sigma}'_k, \mathcal{S}') \neq (R_k, \hat{\sigma}_k, \mathcal{S})$

// checks inputs against records in state

for $i \in \mathcal{S}$ **do**

return $\perp$ **if** $\mathsf{cm}_i \neq \mathsf{H}_{\mathsf{cm}}(i, R_i)$

$\mathsf{msg}_i \leftarrow (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j\in\mathcal{S}})$

if $\mathsf{DS.Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) \neq 1$

return $\perp$

$R \leftarrow \prod_{i\in\mathcal{S}} R_i$

$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$

$z_k \leftarrow r_k + c\lambda_k x_k$

// λ_k is the Lagrange coefficient for k

return z_k

$\mathsf{Combine}(\mathcal{S}, m, \{\mathsf{cm}_i, (R_i, \hat{\sigma}_i), z_i\}_{i\in\mathcal{S}})$

$R \leftarrow \prod_{i\in\mathcal{S}} R_i;\ z \leftarrow \sum_{i\in\mathcal{S}} z_i$

$\sigma \leftarrow (R, z)$

return σ

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$

parse $(R, z) \leftarrow \sigma$

$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$

return 1 **if** $R \cdot \mathsf{pk}^c = g^z$

return 0

Fig. 8. Sparkle+ threshold signature scheme with one round of pre-processing. Consistency checks on the state, e.g., ensuring that $\mathsf{st}_{k,1}$ is only used once, are enforced in the definition of unforgeability (Fig. 7) and implementers must additionally include these checks. The public parameters par are implicitly given as input to all algorithms. We assume an external mechanism to choose the set of signers $\mathcal{S} \subseteq [n]$, where $|\mathcal{S}| = t+1$ and $\mathcal{S}$ is ordered to ensure consistency. DS is a EUF-CMA-secure digital signature scheme (e.g., a Schnorr signature). Verification is identical to the Schnorr signature scheme as in Definition 5.

Key Generation. On input the number of signers n and the threshold $t+1$, the key generation algorithm first generates the secret key $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and threshold public key $\mathsf{pk} \leftarrow g^x$. It then performs a Shamir secret sharing of x: $\{(i, x_i)\}_{i\in[n]} \leftarrow\!\!\$\ \mathsf{IssueShares}(x, n, t+1)$ (Def. 1). It computes the corresponding public key for each participant as $X_i \leftarrow g^{x_i}$. It then runs the key generation algorithm $(\hat{X}_i, \hat{x}_i) \leftarrow \mathsf{DS.KeyGen}()$. It sets $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ and $\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i, \mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$. Finally, it outputs $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]})$.

Signing Round 1 (Sign_1)**.** In the first round, a potential signer samples $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $R_i \leftarrow g^{r_i}$ and $\mathsf{cm}_i \leftarrow \mathsf{H}_{\mathsf{cm}}(i, R_i)$, and outputs commitment cm_i.

Signing Round 2 (Sign_2)**.** In the second round, each signer reveals their committed nonce R_i. On input a message m, a signing set $\mathcal{S}$, and a set of commitments $\{\mathsf{cm}_j\}_{j\in\mathcal{S}}$, each participant $i \in \mathcal{S}$ checks that the commitment given as input on behalf of them, i.e., $\mathsf{cm}_i \in \{\mathsf{cm}_j\}_{j\in\mathcal{S}}$, is in fact what is recorded in their state from Signing Round 1. They then form the message $\mathsf{msg}_i \leftarrow (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j\in\mathcal{S}})$ and run the signing algorithm $\hat{\sigma}_i \leftarrow \mathsf{DS.Sign}(\hat{x}_i, \mathsf{msg}_i)$, used to authenticate the messages sent in Signing Rounds 1 and 2. Each signer then outputs their committed nonce R_i and signature $\hat{\sigma}_i$.

Signing Round 3 (Sign_3)**.** In the third and final round, each signer outputs their partial signature z_i. On input a set of nonce-signature pairs $\{(R_j, \hat{\sigma}_j)\}_{j\in\mathcal{S}}$, each participant $i \in \mathcal{S}$ first checks that the signing set $\mathcal{S}$ and nonce-signature pair given as input on behalf of them, i.e., $(R_i, \hat{\sigma}_i) \in \{(R_j, \hat{\sigma}_j)\}_{j\in\mathcal{S}}$, is in fact what is recorded in their state from Signing Round 2. Each participant then checks that the commitments received in the first round are valid, i.e., if for some $j \in \mathcal{S}$, $\mathsf{cm}_j \neq \mathsf{H}_{\mathsf{cm}}(j, R_j)$, return $\perp$. They also form messages $\mathsf{msg}_j \leftarrow (j, \mathsf{cm}_j, R_j, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ for all $j \in \mathcal{S}$ and check that all signatures $\hat{\sigma}_j$ verify: $\mathsf{DS.Verify}(\hat{X}_j, \mathsf{msg}_j, \hat{\sigma}_j) = 1$ and return $\perp$ if not. The signatures ensures that the honest signing parties all agree on the shared view of the message, signing set, and protocol messages sent in Signing Rounds 1 and 2. If all checks pass, each participant computes the aggregate nonce $R = \prod_{j\in\mathcal{S}} R_j$, $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$, and partial signature $z_i \leftarrow r_i + c\lambda_i x_i$, where λ_i is the Lagrange coefficient for participant i with respect to signing set $\mathcal{S}$. Finally, each signer outputs z_i.

Combining Signatures. On input a signing set $\mathcal{S}$, message m, commitments $\{\mathsf{cm}_j\}_{j\in\mathcal{S}}$, nonces $\{R_j\}_{j\in\mathcal{S}}$, and partial signatures $\{z_j\}_{j\in\mathcal{S}}$, the combiner computes $R \leftarrow \prod_{j\in\mathcal{S}} R_j$ and $z \leftarrow \sum_{j\in\mathcal{S}} z_j$, and outputs $\sigma \leftarrow (R, z)$.

Verification. On input the threshold public key pk, a message m, and a purported signature $\sigma = (R, z)$, the verifier computes $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$ and accepts if $R \cdot \mathsf{pk}^c = g^z$.

Correctness of Sparkle+ is straightforward to verify. Note that verification of the signature σ is identical to verification of a standard (single-party) Schnorr signature (Def. 5) with respect to the threshold public key pk and aggregate nonce R.

6 Our Results

6.1 Static Security Under Standard Assumptions

Our first result is that Sparkle+ is statically secure under the EUF-CMA security of DS and under the discrete logarithm assumption (DL) in the random oracle model (ROM). DS itself may, for example, be a Schnorr signature, which is secure under DL in the ROM [60]. We allow a static adversary to control up to t parties, but they must be declared at the beginning of the protocol (Fig. 7).

Theorem 1. *Let* GrGen *be a group generator and* DS *be a digital signature scheme. For any PPT adversary $\mathcal{A}$ for the game $\mathsf{Game}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1)$ (Fig. 7), there exists a PPT adversary $\mathcal{B}_1$ for the game $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1,\mathsf{DS}}(\kappa)$ (Fig. 4) making at most q_S queries to its signing oracle, and a PPT adversary $\mathcal{B}_2$ for the game $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa)$ (Fig. 5) such that*

$$\mathsf{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa) \leq \sqrt{q \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa) + \frac{q^2}{2p} + \frac{2q}{p} + 2n \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1,\mathsf{DS}}(\kappa)} \tag{3}$$

where n is the number of signers, t is the maximum number of corrupt signers, p is the group size, and $q = q_H + q_S + 1$, where q_H is the number of queries $\mathcal{A}$ can make to the random oracles, and q_S is the number of queries $\mathcal{A}$ can make to the signing oracles.

We formally prove Theorem 1 in Appendix A.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the static unforgeability of Sparkle+ (Fig. 7). We construct a PPT reduction $\mathcal{B}_1$ against the EUF-CMA security of the signature scheme DS and a PPT reduction $\mathcal{B}_2$ against the DL assumption (Fig. 5). We show that if $\mathcal{A}$ produces a forgery with non-negligible probability, then either $\mathcal{B}_1$ breaks the EUF-CMA security of DS, or $\mathcal{B}_2$ breaks the DL assumption with non-negligible probability.

The reduction $\mathcal{B}_1$ takes as input a challenge public key $\hat{X}^\dagger$ for DS and simply passes it to $\mathcal{A}$ as one of the n honest parties' public keys during key generation for DS. $\mathcal{B}_1$ runs trusted key generation for Sparkle+ honestly. If $\mathcal{A}$ forges any signature, then the probability that $\mathcal{A}$ forges a DS signature for this select party in Round 2 is greater than $1/n$.

The reduction $\mathcal{B}_2$ takes as input a DL challenge $\dot{X}$ and aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$. $\mathcal{B}_2$ sets the threshold public key $\mathsf{pk} \leftarrow \dot{X}$ and performs a standard simulation of Shamir secret sharing, resulting in public key shares X_i for all $i \in [n]$.

$\mathcal{B}_2$ runs key generation for DS honestly, resulting in public key shares $\hat{X}_i = g^{\hat{x}_i}$ for all $i \in [n]$. $\mathcal{B}_2$ sets $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ and $\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i)$ for all $i \in [n]$ and returns pk, $\{\mathsf{pk}_i\}_{i\in[n]}$, and $\{\mathsf{sk}_j\}_{j\in\mathsf{cor}}$ to $\mathcal{A}$.

$\mathcal{B}_2$ simulates $\mathsf{Sparkle}$+ signing without knowing the secret keys $\{x_k\}_{k\in\mathsf{hon}}$ of the honest parties as follows. (Our simulation technique is actually quite technical, and we outline a simplified approach here.) $\mathcal{B}_2$ can simulate R_k for all $k \in \mathsf{hon}$ as $R_k \leftarrow g^{z_k} X_k^{-c\lambda_k}$ for random $z_k \leftarrow\!\!\$\ \mathbb{Z}_p$ so that the partial signature z_k output in Round 3 verifies. However, c must equal $\mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$ for aggregate R. Luckily, $\mathcal{B}_2$ can compute $R = \prod_{i\in\mathcal{S}} R_i$, where $\mathcal{S}$ is the signing set, by extracting R_j for all $j \in \mathsf{cor}$ from $\mathcal{A}$'s $\mathsf{H}_{\mathsf{cm}}(j, R_j)$ queries in Round 1. So, $\mathcal{B}_2$ samples a random value for $c \leftarrow\!\!\$\ \mathbb{Z}_p$, computes R_k for all $k \in \mathsf{hon}$ as above, and programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$ – all before $\mathcal{A}$ sees the honest R_k's output at the end of Round 2. This leaves just one problem: $\mathcal{B}_2$ needs to output $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(k, R_k)$ in Round 1 *before* being able to carry out the above steps. But $\mathcal{B}_2$ can simply output a random value $\mathsf{cm}_k \leftarrow\!\!\$\ \mathbb{Z}_p$ in Round 1 and program $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(k, R_k)$ in Round 2.

$\mathcal{B}_2$ then *rewinds* $\mathcal{A}$, using the general forking strategy of Bellare and Neven [12]. With non-negligible probability, $\mathcal{A}$ will output two forgeries across its two iterations $(m^*, (R^*, z^*)), (m', (R', z'))$ that satisfy $(R^*, m^*) = (R', m')$, but where $c^* \neq c'$. Thus, $\mathcal{B}_2$ can compute $\dot{x} = \frac{z^* - z'}{c^* - c'}$ and win the DL game.

6.2 Adaptive Security up to $t/2$ Corruptions

Our second result is that $\mathsf{Sparkle}$+ is adaptively secure against up to $t/2$ corruptions under the EUF-CMA security of DS and under the algebraic one-more discrete logarithm assumption (AOMDL) (Fig. 5) in the random oracle model (ROM). The reason the allowed corruption is $t/2$ is that, in order to extract an AOMDL solution, the reduction needs to rewind the adversary once, and there is no guarantee that the adversary will corrupt the same set of parties after the fork as it did during its first execution. When the adversary can corrupt only $t/2$ parties, this causes no issues, as the total number of corruptions over both iterations does not exceed t. If the adversary could corrupt more parties, then the reduction would query its discrete logarithm oracle more than t times and would lose the $t+1$-aomdl game.

Theorem 2. *Let* GrGen *be a group generator and* DS *be a digital signature scheme. For any PPT adversary* $\mathcal{A}$ *for the adaptive unforgeability game* $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1/2)$ *(Fig. 7), there exists a PPT adversary* $\mathcal{B}'_1$ *for the game* $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}'_1,\mathsf{DS}}(\kappa)$ *(Fig. 4) making at most* q_S *queries to its signing oracle, and a PPT adversary* $\mathcal{B}'_2$ *for the game* $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}'_2}(\kappa)$ *(Fig. 5) such that*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, 1/2) \leq \sqrt{q\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}'_2}(\kappa) + \frac{q^2}{2p} + \frac{5q}{p} + \frac{n^2}{(n-t/2)}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}'_1,\mathsf{DS}}(\kappa)} \tag{4}$$

where n *is the number of signers,* $t/2$ *is the maximum number of corrupted signers,* p *is the group size, and* $q = q_H + q_S + 1$*, where* q_H *is the number of*

queries $\mathcal{A}$ can make to the random oracles, and q_S is the number of queries $\mathcal{A}$ can make to the signing oracles.

We formally prove Theorem 2 in Appendix B.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the $t/2$ adaptive unforgeability of Sparkle+ (Fig. 7). We construct a PPT reduction $\mathcal{B}'_1$ against the EUF-CMA security of the signature scheme DS and a PPT reduction $\mathcal{B}'_2$ against the $t+1$-aomdl assumption (Fig. 5). We show that if $\mathcal{A}$ produces a forgery with non-negligible probability, then either $\mathcal{B}'_1$ breaks the EUF-CMA security of DS, or $\mathcal{B}'_2$ breaks the $t+1$-aomdl assumption with non-negligible probability. The reduction $\mathcal{B}'_1$ is the same as in the static security proof.

The reduction $\mathcal{B}'_2$ takes as input a $t+1$-aomdl challenge $(Y_0, Y_1, \ldots, Y_t)$ and aims to output y_i such that $Y_i = g^{y_i}$ for all $i \in [0..t]$ without querying its discrete logarithm solution oracle more than t times. For all $i \in [n]$, $\mathcal{B}'_2$ sets each Sparkle+ public key share as $X_i \leftarrow Y_0 Y_1^i \cdots Y_t^{i^t}$. The threshold public key is $\mathsf{pk} \leftarrow Y_0$.

$\mathcal{B}'_2$ runs key generation for DS honestly, resulting in public key shares $\hat{X}_i = g^{\hat{x}_i}$ for all $i \in [n]$. $\mathcal{B}'_2$ sets $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ and $\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i)$ for all $i \in [n]$. $\mathcal{B}'_2$ queries its DL oracle on X_j (with representation $(1, j, \ldots, j^t)$) to get x_j for the initial corrupted set cor. $\mathcal{B}'_2$ then returns $(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in\mathsf{cor}})$ to $\mathcal{A}$.

$\mathcal{B}'_2$'s simulation of signing is similar to the proof of static security. (This is by design, we use the same simulation techniques, even if they are not strictly necessary in the static setting, in order to modularize the proofs. Even so, the adaptive proofs are considerably more technical.) In particular, $\mathcal{B}'_2$ again simulates R_k for honest $k \in \mathsf{hon}$ as $R_k \leftarrow g^{z_k} X_k^{-c\lambda_k}$ in Round 2. If $\mathcal{A}$ corrupts an honest party k after Round 2 has begun, then $\mathcal{B}'_2$ queries its DL oracle on X_k (with rep. $(1, k, \ldots, k^t)$) to get x_k, computes $r_k \leftarrow z_k - c\lambda_k x_k$, and returns $x_k, \hat{x}_k$, and the honest party's state, including nonces, commitments, etc. across all signing sessions to $\mathcal{A}$. However, $\mathcal{A}$ may choose to corrupt some honest k *before* Round 2, or even before it outputs its own cm_j's in Round 1. In this case, $\mathcal{B}'_2$ samples a random $r_k \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $R_k \leftarrow g^{r_k}$, and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(k, R_k)$ for the random cm_k it output in Round 1. It then queries its DL oracle on X_k (with rep. $(1, k, \ldots, k^t)$) to get x_k and returns it, $\hat{x}_k$, and state across all signing sessions to $\mathcal{A}$. As in the proof of static security, $\mathcal{B}'_2$ rewinds $\mathcal{A}$ in order to extract $x = y_0$ from $\mathcal{A}$'s two forgeries. Assume w.l.o.g. that $\mathcal{A}$ corrupts t parties over the two iterations. ($\mathcal{A}$ can corrupt up to $t/2$ parties in each iteration, and $\mathcal{B}'_2$ can corrupt the remaining itself.) For simplicity, say the corrupt indices are $\mathsf{cor} = \{1, \ldots, t\}$. Then $\mathcal{B}'_2$ has made t DL queries on $g^{x_k} = Y_0 Y_1^k \cdots Y_t^{k^t}$. This forms a system of linear equations:

$$\begin{pmatrix} x \\ x_1 \\ \vdots \\ x_t \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t & \cdots & t^t \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{pmatrix}$$

This is a Vandermonde matrix and is therefore invertible. $\mathcal{B}'_2$ knows $(x, x_1, \ldots, x_t)$, and so can solve for $(y_0, y_1, \ldots, y_t)$ to win the $t+1$-aomdl game.

6.3 Adaptive Security up to t Corruptions

We now present our main result: Sparkle+ is secure against a full t adaptive corruptions. In particular, if exactly $t+1$ parties engage in signing, all but one of them could be malicious and the unforgeability of Sparkle+ would still hold. We prove this result under the EUF-CMA security of DS and under the AOMDL assumption (Fig. 5) in the AGM with random oracles.

Theorem 3. *Let* GrGen *be a group generator and* DS *be a digital signature scheme. For any* algebraic *PPT adversary* $\mathcal{A}$ *for the adaptive unforgeability game* $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac}=1)$ *(Fig. 7), there exists a PPT adversary* $\mathcal{B}'_1$ *for the game* $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}'_1,\mathsf{DS}}(\kappa)$ *(Fig. 4) making at most* q_S *queries to its signing oracle, and a PPT adversary* $\mathcal{B}''_2$ *for the game* $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}''_2}(\kappa)$ *(Fig. 5) such that*

$$\mathit{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa,1) \leq \mathit{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}''_2}(\kappa) + \frac{q^2}{2p} + \frac{5q}{p} + \frac{n^2}{(n-t)}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}''_1,\mathsf{DS}}(\kappa) \quad (5)$$

where n is the number of signers, t is the maximum number of corrupted signers, p is the group size, and $q = q_H + q_S + 1$, where q_H is the number of queries $\mathcal{A}$ can make to the random oracles, and q_S is the number of queries $\mathcal{A}$ can make to the signing oracles.

We formally prove Theorem 3 in Appendix C.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the full adaptive unforgeability of Sparkle+ (Fig. 7). We construct a PPT reduction $\mathcal{B}''_1$ against the EUF-CMA security of the signature scheme DS and a PPT reduction $\mathcal{B}''_2$ against the $t+1$-aomdl assumption (Fig. 5). We show that if $\mathcal{A}$ produces a forgery with non-negligible probability, then either $\mathcal{B}''_1$ breaks the EUF-CMA security of DS, or $\mathcal{B}''_2$ breaks the $t+1$-aomdl assumption with non-negligible probability. The reduction $\mathcal{B}''_1$ is the same as in the static security proof.

The reduction $\mathcal{B}''_2$ takes as input a $t+1$-aomdl challenge $(Y_0, Y_1, \ldots, Y_t)$ and aims to output y_i such that $Y_i = g^{y_i}$ for all $i \in [0..t]$ without querying its DL oracle more than t times. $\mathcal{B}''_2$ simulates key generation, signing, and corruption as in the $t/2$-adaptive proof, but does not rewind $\mathcal{A}$, so $\mathcal{A}$ may corrupt a full t parties.

$\mathcal{A}$'s forgery $(m^*, (R^*, z^*))$ verifies as $R^* = g^{z^*}\mathsf{pk}^{-c^*}$, where $\mathcal{A}$ provided a representation of R^* when querying $c^* = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m^*, R^*)$:

$$R^* = g^{\gamma^*}\mathsf{pk}^{\xi^*} X_1^{\xi_1^*} \cdots X_n^{\xi_n^*} \prod_{i=1}^{q_S} R_{i,1}^{\rho_{i,1}^*} \cdots R_{i,n}^{\rho_{i,n}^*}$$

where $\{R_{i,k}\}_{i\in[q_S]}$ are the honest nonces returned by the $\mathcal{O}^{\mathsf{Sign}_2}$ oracle over q_S signing queries and $\rho_{i,k}^* = 0$ for k not in the signing set for query i. Each $R_{i,k}$ verifies as $R_{i,k} = g^{z_{i,k}} X_k^{-c_i\lambda_{i,k}}$, where $c_i = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m_i, R_i)$. Equating the two expressions for R^* and rearranging, we have:

$$g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1}\rho^*_{i,1}} \cdots g^{-z_{i,n}\rho^*_{i,n}}$$

$$= \mathsf{pk}^{c^*} \mathsf{pk}^{\xi^*} X_1^{\xi^*_1} \cdots X_n^{\xi^*_n} \prod_{i=1}^{q_S} (X_1^{-c_i\lambda_{i,1}})^{\rho^*_{i,1}} \cdots (X_n^{-c_i\lambda_{i,n}})^{\rho^*_{i,n}}$$

$\mathcal{B}''_2$ queries its DL oracle on X_j (with rep. $(1, j, \ldots, j^t)$) to obtain $\{x_j\}_{j\in\mathsf{cor}}$ for the t corrupt parties. For all $k \in [n]$, $X_k = \mathsf{pk} Y_1^k Y_2^{k^2} \cdots Y_t^{k^t}$, and for all $i \in [t]$, $Y_i = \mathsf{pk}^{L'_{0,i}} \prod_{j\in\mathsf{cor}} g^{x_j L'_{j,i}}$, where $L'_{j,i}$ is the ith coefficient of the Lagrange polynomial $L'_j(Z)$ for the set $0 \cup \mathsf{cor}$. Plugging these in and rearranging, we have:

$$g^{\eta^*} = \mathsf{pk}^{c^*+\xi^*} \prod_{k=1}^{n} X^{\mu^*_k} g^{\nu^*_k}$$

where $\eta^* = z^* - \gamma^* - \sum_{i=1}^{q_S}(z_{i,1}\rho^*_{i,1} + \cdots + z_{i,n}\rho^*_{i,n})$, $\mu^*_k = 1 + \sum_{i=1}^{t} L'_{0,i} k^i$, and $\nu^*_k = \sum_{i=1}^{t} \left(\sum_{j\in\mathsf{cor}} x_j L'_{j,i}\right) k^i$. Then:

$$x = \frac{\eta^* - \sum_{k=1}^{n} \nu^*_k}{c^* + \xi^* + \sum_{k=1}^{n} \mu^*_k}$$

$\mathcal{A}$ fixed R^* and thus $\eta^*, \{\nu^*_i\}_{i\in[n]}, \xi^*, \{\mu^*_i\}_{i\in[n]}$ as it queried $\mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m^*, R^*)$ to receive random c^*. Thus, the denominator is nonzero with overwhelming probability and $\mathcal{B}''_2$ can solve for x. $\mathcal{B}''_2$ can then compute $(y_0, y_1, \ldots, y_t)$ as in the $t/2$-adaptive proof to win the $t+1$-aomdl game.

7 Conclusion

In this work, we present Sparkle+, an efficient and practical threshold Schnorr signature scheme with one round of pre-processing and two online signing rounds. We show that Sparkle+ achieves several levels of security based on different corruption models and assumptions. For static security, we show that Sparkle+ can be proven secure under the DL assumption in the ROM. We then show that Sparkle+ can be proven adaptively secure under $t/2$ corruptions, assuming AOMDL in the ROM. Finally, we show that Sparkle+ can be proven fully adaptively secure (with up to t corruptions), assuming AOMDL in the ROM and AGM.

Our work leaves open the intriguing question of under what conditions threshold signatures can be proven adaptively secure, and the extent to which threshold Schnorr signatures can achieve adaptive security under weaker security assumptions.

References

[1] M. Abdalla, M. Barbosa, J. Katz, J. Loss, and J. Xu. "Algebraic Adversaries in the Universal Composability Framework". In: *Advances in Cryptology - ASIACRYPT*

2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III. Ed. by M. Tibouchi and H. Wang. Vol. 13092. Lecture Notes in Computer Science. Springer, 2021, pp. 311–341.

[2] M. Abe and S. Fehr. "Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography". In: *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*. Ed. by M. K. Franklin. Vol. 3152. Lecture Notes in Computer Science. Springer, 2004, pp. 317–334. DOI: 10.1007/978-3-540-28628-8_20. URL: https://doi.org/10.1007/978-3-540-28628-8_20.

[3] J. F. Almansa, I. Damgård, and J. B. Nielsen. "Simplified Threshold RSA with Adaptive and Proactive Security". In: *EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006*. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, 2006, pp. 593–611.

[4] H. K. Alper and J. Burdges. "Two-Round Trip Schnorr Multi-signatures via Delinearized Witnesses". In: *CRYPTO 2021, Virtual Event, August 16-20, 2021*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Springer, 2021, pp. 157–188.

[5] R. Bacho, S. Das, J. Loss, and L. Ren. "Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security". In: *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part II*. Ed. by S. Fehr and P. Fouque. Vol. 15602. Lecture Notes in Computer Science. Springer, 2025, pp. 304–334. DOI: 10.1007/978-3-031-91124-8_11. URL: https://doi.org/10.1007/978-3-031-91124-8_11.

[6] R. Bacho and J. Loss. "On the Adaptive Security of the Threshold BLS Signature Scheme". In: *IACR Cryptol. ePrint Arch.* (2022), p. 534. URL: https://eprint.iacr.org/2022/534.

[7] R. Bacho, J. Loss, G. Stern, and B. Wagner. "HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures". In: *IACR Cryptol. ePrint Arch.* (2024), p. 280. URL: https://eprint.iacr.org/2024/280.

[8] R. Bacho, J. Loss, S. Tessaro, B. Wagner, and C. Zhu. "Twinkle: Threshold Signatures from DDH with Full Adaptive Security". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1482. URL: https://eprint.iacr.org/2023/1482.

[9] B. Bauer, G. Fuchsbauer, and A. Plouviez. "The One-More Discrete Logarithm Assumption in the Generic Group Model". In: *ASIACRYPT 2021, Singapore, December 6-10, 2021*. Ed. by M. Tibouchi and H. Wang. Vol. 13093. LNCS. Springer, 2021, pp. 587–617.

[10] M. Bellare, E. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. *Better than advertised security for non-interactive threshold signatures*. CRYPTO 2022. To appear. 2022.

[11] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme". In: *J. Cryptol.* 16.3 (2003), pp. 185–215.

[12] M. Bellare and G. Neven. "Multi-signatures in the plain public-Key model and a general forking lemma". In: *CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. Ed. by A. Juels, R. N. Wright, and S. D. C. di Vimercati. ACM, 2006, pp. 390–399.

[13] M. Bellare and A. Palacio. "GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks". In: *CRYPTO 2002, USA, August 18-22, 2002, Proceedings*. Ed. by M. Yung. Vol. 2442. LNCS. Springer, 2002, pp. 162–177. DOI: 10.1007/3-540-45708-9_11. URL: https://doi.org/10.1007/3-540-45708-9_11.

[14] M. Bellare and P. Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596. URL: https://doi.org/10.1145/168588.168596.

[15] M. Bellare and S. Shoup. "Two-Tier Signatures, Strongly Unforgeable Signatures, and Fiat-Shamir Without Random Oracles". In: *PKC 2007, Beijing, China, April 16-20, 2007, Proceedings*. Ed. by T. Okamoto and X. Wang. Vol. 4450. LNCS. Springer, 2007, pp. 201–216. DOI: 10.1007/978-3-540-71677-8_14. URL: https://doi.org/10.1007/978-3-540-71677-8_14.

[16] M. Bellare, S. Tessaro, and C. Zhu. "Stronger Security for Non-Interactive Threshold Signatures". In: *IACR Cryptol. ePrint Arch.* (2022), p. 833. URL: https://eprint.iacr.org/2022/833.

[17] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. "On the (in)Security of ROS". In: *J. Cryptol.* 35.4 (2022), p. 25. DOI: 10.1007/S00145-022-09436-0. URL: https://doi.org/10.1007/s00145-022-09436-0.

[18] A. Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC 2003, Miami, FL, USA, January 6-8, 2003*. Ed. by Y. Desmedt. Vol. 2567. LNCS. Springer, 2003, pp. 31–46.

[19] D. Boneh, M. Drijvers, and G. Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018*. Ed. by T. Peyrin and S. D. Galbraith. Vol. 11273. LNCS. Springer, 2018, pp. 435–464.

[20] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2020. URL: http://toc.cryptobook.us/book.pdf.

[21] L. Brandão and M. Davidson. *Notes on Threshold EdDSA/Schnorr Signatures*. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8214B.ipd.pdf. 2022.

[22] L. Brandão and R. Peralta. *NIST First Call for Multi-Party Threshold Schemes*. https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8214C.ipd.pdf. 2023.

[23] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145.

[24] R. Canetti, U. Feige, O. Goldreich, and M. Naor. "Adaptively Secure Multi-Party Computation". In: *STOC '96, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. Ed. by G. L. Miller. ACM, 1996, pp. 639–648.

[25] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *IACR Cryptol. ePrint Arch.* (2021), p. 60. URL: https://eprint.iacr.org/2021/060.

[26] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Adaptive Security for Threshold Cryptosystems". In: *CRYPTO '99, Santa Barbara, California, USA,*

August 15-19, 1999. Ed. by M. J. Wiener. Vol. 1666. LNCS. Springer, 1999, pp. 98–115.

[27] D. Chaum and T. P. Pedersen. "Wallet Databases with Observers". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by E. F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 89–105. DOI: 10.1007/3-540-48071-4_7. URL: https://doi.org/10.1007/3-540-48071-4_7.

[28] Y. Chen. "Round-Efficient Adaptively Secure Threshold Signatures with Rewinding". In: *IACR Cryptol. ePrint Arch.* (2025), p. 638. URL: https://eprint.iacr.org/2025/638.

[29] H. Chu, P. Gerhart, T. Ruffing, and D. Schröder. "Practical Schnorr Threshold Signatures Without the Algebraic Group Model". In: *CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14081. LNCS. Springer, 2023, pp. 743–773. DOI: 10.1007/978-3-031-38557-5_24. URL: https://doi.org/10.1007/978-3-031-38557-5_24.

[30] D. Connolly, C. Komlo, I. Goldberg, and C. Wood. *Two-Round Threshold Schnorr Signatures with FROST*. 2022. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/.

[31] E. Crites, J. Katz, C. Komlo, S. Tessaro, and C. Zhu. *On the Adaptive Security of FROST. To appear in CRYPTO 2025*.

[32] E. Crites, C. Komlo, and M. Maller. *On the Adaptive Security of Key-Unique Threshold Signatures*. Cryptology ePrint Archive, Paper 2025/943. 2025. URL: https://eprint.iacr.org/2025/943.

[33] E. Crites and A. Stewart. *A Plausible Attack on the Adaptive Security of Threshold Schnorr Signatures. To appear in CRYPTO 2025*.

[34] E. C. Crites, C. Komlo, and M. Maller. "Fully Adaptive Schnorr Threshold Signatures". In: *CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14081. LNCS. Springer, 2023, pp. 678–709. DOI: 10.1007/978-3-031-38557-5_22. URL: https://doi.org/10.1007/978-3-031-38557-5_22.

[35] E. C. Crites, C. Komlo, and M. Maller. "How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures". In: *IACR Cryptol. ePrint Arch.* (2021), p. 1375. URL: https://eprint.iacr.org/2021/1375.

[36] S. Das and L. Ren. "Adaptively Secure BLS Threshold Signatures from DDH and co-CDH". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1553. URL: https://eprint.iacr.org/2023/1553.

[37] M. Drijvers et al. "On the Security of Two-Round Multi-Signatures". In: *SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1084–1101.

[38] B. Edgington. *Upgrading Ethereum*. 2023. URL: https://eth2book.info/bellatrix/part2/building_blocks/randomness/.

[39] A. Fiat and A. Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO 1986, Santa Barbara, California, USA, 1986*. Ed. by A. M. Odlyzko. Vol. 263. LNCS. Springer, 1986, pp. 186–194.

[40] M. Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *CRYPTO 2005, Santa Barbara, California, USA, August 14-18, 2005*. Ed. by V. Shoup. Vol. 3621. LNCS. Springer, 2005, pp. 152–168.

[41] Y. Frankel, P. D. MacKenzie, and M. Yung. "Adaptively-Secure Distributed Public-Key Systems". In: *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*. Ed. by J. Nesetril. Vol. 1643. Lecture Notes in Computer Science. Springer, 1999, pp. 4–27. DOI: 10.1007/3-540-48481-7_2. URL: https://doi.org/10.1007/3-540-48481-7_2.

[42] Y. Frankel, P. D. MacKenzie, and M. Yung. "Adaptively-Secure Optimal-Resilience Proactive RSA". In: *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings*. Ed. by K. Lam, E. Okamoto, and C. Xing. Vol. 1716. Lecture Notes in Computer Science. Springer, 1999, pp. 180–194. DOI: 10.1007/978-3-540-48000-6_15. URL: https://doi.org/10.1007/978-3-540-48000-6_15.

[43] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: *CRYPTO 2018, Santa Barbara, CA, USA, August 19-23, 2018*. Ed. by H. Shacham and A. Boldyreva. Vol. 10992. LNCS. Springer, 2018, pp. 33–62.

[44] G. Fuchsbauer, A. Plouviez, and Y. Seurin. "Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model". In: *EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020*. Ed. by A. Canteaut and Y. Ishai. Vol. 12106. LNCS. Springer, 2020, pp. 63–95.

[45] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Applications of Pedersen's Distributed Key Generation Protocol". In: *CT-RSA 2003, San Francisco, CA, USA, April 13-17, 2003*. Ed. by M. Joye. Vol. 2612. LNCS. Springer, 2003, pp. 373–390.

[46] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *J. Cryptol.* 20.1 (2007), pp. 51–83.

[47] C. Gentry and D. Wichs. "Separating succinct non-interactive arguments from all falsifiable assumptions". In: *STOC 2011, San Jose, CA, USA, 6-8 June 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 99–108.

[48] S. Goldwasser, S. Micali, and R. L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM J. Comput.* 17.2 (1988), pp. 281–308.

[49] S. Jarecki and A. Lysyanskaya. "Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures". In: *EUROCRYPT 2000, Bruges, Belgium, May 14-18, 2000*. Ed. by B. Preneel. Vol. 1807. LNCS. Springer, 2000, pp. 221–242.

[50] S. Katsumata, M. Reichle, and K. Takemure. "Adaptively Secure 5 Round Threshold Signatures from sfMLWE /sfMSIS and sfDL with Rewinding". In: *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII*. Ed. by L. Reyzin and D. Stebila. Vol. 14926. Lecture Notes in Computer Science. Springer, 2024, pp. 459–491. DOI: 10.1007/978-3-031-68394-7_15. URL: https://doi.org/10.1007/978-3-031-68394-7_15.

[51] N. Koblitz and A. Menezes. "Another look at non-standard discrete log and Diffie-Hellman problems". In: *J. Math. Cryptol.* 2.4 (2008), pp. 311–326.

[52] C. Komlo and I. Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *SAC 2020, Halifax, NS, Canada (Virtual Event), October 21-23, 2020*. Ed. by O. Dunkelman, M. J. J. Jr., and C. O'Flynn. Vol. 12804. LNCS. Springer, 2020, pp. 34–65. DOI: 10.1007/978-3-030-81652-0_2.

[53] B. Libert, M. Joye, and M. Yung. "Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares". In: *Theoretical Computer Science* 645 (2016), pp. 1–24.

[54] Y. Lindell. "Simple Three-Round Multiparty Schnorr Signing with Full Simulatability". In: *IACR Cryptol. ePrint Arch.* (2022), p. 374. URL: https://eprint.iacr.org/2022/374.

[55] A. Lysyanskaya and C. Peikert. "Adaptive Security in the Threshold Setting: From Cryptosystems to Signature Schemes". In: *ASIACRYPT 2001, Gold Coast, Australia, December 9-13, 2001*. Ed. by C. Boyd. Vol. 2248. LNCS. Springer, 2001, pp. 331–350.

[56] N. Makriyannis. *On the Classic Protocol for MPC Schnorr Signatures*. Cryptology ePrint Archive, Paper 2022/1332. https://eprint.iacr.org/2022/1332. 2022. URL: https://eprint.iacr.org/2022/1332.

[57] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. "Simple Schnorr multi-signatures with applications to Bitcoin". In: *Des. Codes Cryptogr.* 87.9 (2019), pp. 2139–2164.

[58] J. Nick, T. Ruffing, and Y. Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *CRYPTO 2021, Virtual Event, August 16-20, 2021*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Springer, 2021, pp. 189–221.

[59] A. Nicolosi, M. N. Krohn, Y. Dodis, and D. Mazières. "Proactive Two-Party Signatures for User Authentication". In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*. The Internet Society, 2003. URL: https://www.ndss-symposium.org/ndss2003/proactive-two-party-signatures-user-authentication/.

[60] D. Pointcheval and J. Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *J. Cryptol.* 13.3 (2000), pp. 361–396.

[61] D. Pointcheval and J. Stern. "Security Proofs for Signature Schemes". In: *EUROCRYPT 1996, Saragossa, Spain, May 12-16, 1996*. Ed. by U. M. Maurer. Vol. 1070. LNCS. Springer, 1996, pp. 387–398.

[62] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. "ROAST: Robust Asynchronous Schnorr Threshold Signatures". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by H. Yin, A. Stavrou, C. Cremers, and E. Shi. ACM, 2022, pp. 2551–2564. DOI: 10.1145/3548606.3560583. URL: https://doi.org/10.1145/3548606.3560583.

[63] C. Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptol.* 4.3 (1991), pp. 161–174.

[64] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[65] D. R. Stinson and R. Strobl. "Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates". In: *ACISP 2001, Sydney, Australia, July 11-13, 2001*. Ed. by V. Varadharajan and Y. Mu. Vol. 2119. LNCS. Springer, 2001, pp. 417–434.

A Proof of Static Security

We now prove Theorem 1. Our static security proof serves as a template for the adaptive security proofs later on. In particular, it uses the same simulation techniques, even if they are not strictly necessary in the static setting. In this

Algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$

Pick random coins ρ for $\mathcal{C}$.
$h_1, \ldots, h_q \leftarrow\!\!\$\ H$
$(I, \mathsf{out}) \leftarrow \mathcal{C}\big(X, (h_1, \ldots, h_q); \rho\big)$
return $\bot$ **if** $I = 0$
$h'_I, \ldots, h'_q \leftarrow\!\!\$\ H$
$(I', \mathsf{out}') \leftarrow \mathcal{C}\big(X, (h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q); \rho\big)$
return $\bot$ **if** $I \neq I' \vee h_I = h'_I$
return $\mathsf{output} \leftarrow (I, \mathsf{out}, \mathsf{out}')$

Fig. 9. The forking algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$ associated to an algorithm $\mathcal{C}$ and input X.

way, static security can be viewed as a special case of adaptive security where all corrupt parties are corrupted at the onset of the protocol. This approach limits redundancy in the three proofs we present in this work and modularizes changes made in subsequent games when adaptive corruptions are added.

Our static and $t/2$-adaptive security reductions make use of rewinding. We recall the general forking lemma by Bellare and Neven [12], which is a formalization of the forking lemma introduced by Pointcheval and Stern [61].

Lemma 1 (General Forking Lemma [12]). *Let $q \in \mathbb{N}$ and H be a finite, non-empty set. Let IG be a randomized algorithm that generates an input X. Let $\mathcal{C}$ be a randomized algorithm that takes as input X and $h_1, \ldots, h_q \in H$ and random coins ρ and outputs an index $I \in [0..q]$ and side output out. Let $\mathsf{accept}(\mathcal{C})$ be the probability that $\mathcal{C}$ outputs an index $I \neq 0$ in the following experiment:*

$$\mathsf{accept}(\mathcal{C}) := \Pr[X \leftarrow\!\!\$\ \mathsf{IG};\ h_1, \ldots, h_q \leftarrow\!\!\$\ H;\ (I, \mathsf{out}) \leftarrow \mathcal{C}\big(X, (h_1, \ldots, h_q); \rho\big) : I \neq 0]$$

Let the forking algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$ associated to $\mathcal{C}$ be the randomized algorithm that takes as input X and proceeds as in Figure 9. Let

$$\mathsf{accept}(\mathsf{Fork}^{\mathcal{C}}) := \Pr[X \leftarrow\!\!\$\ \mathsf{IG};\ \mathsf{output} \leftarrow\!\!\$\ \mathsf{Fork}^{\mathcal{C}}(X) : \mathsf{output} \neq \bot]$$

Then, $\mathsf{accept}(\mathsf{Fork}^{\mathcal{C}})$ is bounded by

$$\mathsf{accept}(\mathsf{Fork}^{\mathcal{C}}) \geq \mathsf{accept}(\mathcal{C}) \cdot \left(\frac{\mathsf{accept}(\mathcal{C})}{q} - \frac{1}{|H|} \right). \tag{6}$$

or, alternatively,

$$\mathsf{accept}(\mathcal{C}) \leq \frac{q}{|H|} + \sqrt{q \cdot \mathsf{accept}(\mathsf{Fork}^{\mathcal{C}})} \tag{7}$$

Proof. (of Theorem 1.) We prove the theorem by a sequence of games.

Game 0. This is the static unforgeability game as defined in Figure 7, instantiated with $\mathsf{Sparkle}+$.

Let $\mathcal{A}$ be a PPT adversary attempting to break the static unforgeability of $\mathsf{Sparkle}+$ (Fig. 7) that makes up to q_H queries to H_{cm} and $\mathsf{H}_{\mathsf{sig}}$, and q_S queries to its signing oracles, each time it is executed. We assume, without loss of generality, that $\mathcal{A}$ makes a query to $\mathsf{H}_{\mathsf{sig}}$ on its forgery (pk, m^*, R^*). Then, let $q = q_H + q_S + 1$.

Let W_0 be the event that $\mathcal{A}$ wins Game 0. Then, the advantage of $\mathcal{A}$ is:

$$\mathsf{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1) = |\Pr[W_0]| \tag{8}$$

Here, $\mathsf{frac} = 1$ to allow $\mathcal{A}$ to corrupt a full $\mathsf{frac} \cdot t = t$ parties, which must be declared by $\mathcal{A}$ at the beginning of its execution.

Game 1. In this game, we rule out the event that $\mathcal{A}$ forges a DS signature for some honest party. That is, the game aborts if the following event $\mathsf{ForgeDS}$ occurs.

In Game 1, the environment initializes a table $\mathsf{Q}_{\mathsf{msg}} \leftarrow \emptyset$. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_2}$ on inputs $(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, the environment first performs the checks in $\mathcal{O}^{\mathsf{Sign}_2}$ (Fig. 7) and Sign_2 (Fig. 8). If they pass, then, in particular, $\mathsf{st}_{k,\mathsf{eid},1} = (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$, where $\mathsf{cm}_k \in \{\mathsf{cm}_i\}_{i\in\mathcal{S}}$. The environment can then form the message $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ and complete Round 2 honestly. Finally, the environment adds msg_k to $\mathsf{Q}_{\mathsf{msg}}$.

When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_3}$ on inputs $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$, the environment first performs the checks in $\mathcal{O}^{\mathsf{Sign}_3}$ (Fig. 7) and Sign_3 (Fig. 8). If they pass, then, in particular, $\mathsf{st}_{k,\mathsf{eid},2} = (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \mathsf{sk}_k)$, where $(R_k, \hat{\sigma}_k) \in \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}}$. The environment forms, for all $i \in \mathcal{S}$, the messages $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j\in\mathcal{S}})$. If the following holds, the environment aborts:

$$\forall i \in \mathsf{hon} \cap \mathcal{S}, \mathsf{DS.Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) = 1, \text{but } \exists i' \text{ s.t. } \mathsf{msg}_{i'} \notin \mathsf{Q}_{\mathsf{msg}} \tag{9}$$

That is, $\mathcal{A}$ has forged a signature $\hat{\sigma}_{i'}$ on a message $\mathsf{msg}_{i'}$ that honest party i' never signed. We refer to this abort event as $\mathsf{ForgeDS}$.

Reduction $\mathcal{B}_1$ to the EUF-CMA Security of DS. The event that $\mathsf{ForgeDS}$ occurs implies $\mathcal{A}$ breaks EUF-CMA security of DS (Fig. 4) for some public key $\hat{\mathsf{pk}}_{i'}$. Therefore, we can define a reduction $\mathcal{B}_1$ against the EUF-CMA game. We show that when $\mathcal{A}$ wins Game 1, then $\mathcal{B}_1$ wins its game with non-negligible probability.

The reduction $\mathcal{B}_1^{\mathcal{O}^{\mathsf{DS.Sign}}}(\mathsf{par}_{\mathsf{DS}}, \hat{X}^\dagger)$ accepts as input public parameters $\mathsf{par}_{\mathsf{DS}}$ and a challenge public key $\hat{\mathsf{pk}} \leftarrow \hat{X}^\dagger$; its goal is to output a valid forgery $(\mathsf{msg}^*, \hat{\sigma}^*)$ under $\hat{\mathsf{pk}}$ such that $\mathcal{B}_1$ has not queried msg^* to its signing oracle $\mathcal{O}^{\mathsf{DS.Sign}}$.

Setup. $\mathcal{B}_1$ initializes the following values:

- $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$ and parameters for DS by performing $\mathsf{par}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$. $\mathcal{B}_1$ then sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{par}_{\mathsf{DS}})$.
- The set $\mathsf{Q}_{\mathsf{msg}} \leftarrow \emptyset$ of messages msg_k signed by honest parties during $\mathcal{O}^{\mathsf{Sign}_2}$.
- The set $\mathsf{Q}_m \leftarrow \emptyset$ of messages queried by $\mathcal{A}$ to $\mathcal{O}^{\mathsf{Sign}_2}$.
- The set $\mathsf{Q}_{\mathsf{st}} \leftarrow \emptyset$ of participants' states for all signing sessions initiated by $\mathcal{A}$.

$\mathcal{B}_1$ runs $(n, t+1, \mathsf{cor}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{par})$, and sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$. $\mathcal{A}$ is given access to random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$.

$\mathcal{B}_1$ runs key generation for Sparkle+ honestly: $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) \leftarrow\!\!\$\ \mathsf{KeyGen}(n, t+1)$. Then, it samples $\tau \leftarrow\!\!\$\ \mathsf{hon}$ and parses $(X_\tau, \hat{X}_\tau) \leftarrow \mathsf{pk}_\tau$ and $(x_\tau, \hat{x}_\tau, \mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}) \leftarrow \mathsf{sk}_\tau$. It replaces $\hat{X}_\tau$ with its challenge: $\mathsf{pk}_\tau \leftarrow (X_\tau, \hat{X}^\dagger)$ and $\mathsf{sk}_\tau \leftarrow (x_\tau, \perp, \mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$.

$\mathcal{B}_1$ then runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_1}, \mathcal{O}^{\mathsf{Sign}_2}, \mathcal{O}^{\mathsf{Sign}_3}}(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in\mathsf{cor}}, \mathsf{st}_{\mathcal{A}})$.

Simulating Random Oracle Queries. When $\mathcal{A}$ queries H_{cm} or $\mathsf{H}_{\mathsf{sig}}$, $\mathcal{B}_1$ follows the protocol honestly.

Simulating Signing Oracle Queries. When $\mathcal{A}$ queries its signing oracles, $\mathcal{B}_1$ does as follows:

- $\mathcal{O}^{\mathsf{Sign}_1}$: When $\mathcal{A}$ queries on $k \in \mathsf{hon}$, $\mathcal{B}_1$ follows the protocol honestly.
- $\mathcal{O}^{\mathsf{Sign}_2}$: When $\mathcal{A}$ queries on $(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, $\mathcal{B}_1$ first performs the checks in $\mathcal{O}^{\mathsf{Sign}_2}$ (Fig. 7) and Sign_2 (Fig. 8). If they pass, then $\mathcal{B}_1$ updates $\mathsf{Q}_m \leftarrow \mathsf{Q}_m \cup \{m\}$, and computes $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$. It then does the following:
  1. If $k \neq \tau$, then $\mathcal{B}_1$ follows the protocol honestly and computes the DS signature $\hat{\sigma}_k$ on msg_k.
  2. If $k = \tau$, then $\mathcal{B}_1$ queries its own signing oracle $\mathcal{O}^{\mathsf{DS.Sign}}(\mathsf{msg}_\tau)$ to obtain $\hat{\sigma}_\tau$. It otherwise follows the protocol honestly.

  Finally, $\mathcal{B}_1$ updates $\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature $\hat{\sigma}_k$ on msg_k.
- $\mathcal{O}^{\mathsf{Sign}_3}$: When $\mathcal{A}$ queries on $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}_1$ first performs the checks in $\mathcal{O}^{\mathsf{Sign}_3}$ (Fig. 7) and Sign_3 (Fig. 8). If they pass, then $\mathcal{B}_1$ checks if Condition 9 holds. If it does not hold, B_1 follows the protocol honestly.
  Otherwise, if Condition 9 holds, then ForgeDS has occurred. If ForgeDS has occurred for $i' = \tau$ (i.e., $\mathcal{A}$ has output a forgery for the simulated player τ), then $\mathcal{B}_1$ terminates the execution of $\mathcal{A}$ and returns $(\mathsf{msg}_\tau, \hat{\sigma}_\tau)$ as its forgery for the EUF-CMA game. Otherwise, $\mathcal{B}_1$ aborts.

If $\mathcal{A}$ terminates and ForgeDS does not occur, then $\mathcal{B}_1$ outputs 0 for the EUF-CMA game.

Analysis of $\mathcal{B}_1$'s Simulation. $\mathcal{B}_1$ simulates H_{cm} and $\mathsf{H}_{\mathsf{sig}}$ perfectly because it follows the protocol honestly.

$\mathcal{B}_1$ responds to $\mathcal{O}^{\mathsf{Sign}_1}$ queries honestly, so its simulation is perfect. It also responds to $\mathcal{O}^{\mathsf{Sign}_2}$ and $\mathcal{O}^{\mathsf{Sign}_3}$ queries honestly, with the exception of participant τ. However, because $\mathcal{B}_1$ queries its own EUF-CMA signing oracle $\mathcal{O}^{\mathsf{DS.Sign}}$ to receive signatures that verify under $\hat{\mathsf{pk}}_\tau = \hat{X}^\dagger$, $\mathcal{B}_1$'s simulation is perfect.

Difference between Game 0 and Game 1. If ForgeDS does not occur, then Game 0 and Game 1 are identical. If ForgeDS occurs, then the probability that $\mathcal{A}$ forges for participant τ is at least $\frac{1}{n}$. The additional advantage for $\mathcal{A}$ in Game 1, assuming that ForgeDS occurs for participant τ, is upper-bounded by the advantage that $\mathcal{B}_1$ wins its EUF-CMA game against DS.

Let W_1 be the event that $\mathcal{A}$ wins Game 1. Then:

$$|\Pr[W_0] - \Pr[W_1]| \leq n\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1,\mathsf{DS}}(\kappa) \tag{10}$$

Game 2. Next, we define a game that rules out several bad events: $\mathsf{ROCollision}$, $\mathsf{H_{sig}Guess}, \mathsf{ForgeDS}_k, \mathsf{ForgeDS}_i, \mathsf{FromInvalid}_k$. That is, the game aborts if any of these events occurs.

We next define a simulator algorithm SIM that simulates Game 1 to a static unforgeability adversary $\mathcal{A}$.

Setup. SIM accepts as input an instance X, which is a tuple consisting of the group description $(\mathbb{G}, p, g)$ and a discrete logarithm (DL) challenge $\dot{X} \in \mathbb{G}$. In addition, SIM accepts as input a set of $q = q_H + q_S + 1$ values $\{h_1, \ldots, h_q\}$, which it will use to program its random oracles responses (i.e., $\mathsf{SIM} = \mathcal{C}$ in the forking lemma notation of Fig. 9).

Next, SIM checks for collisions $h_i = h_j$ for $i, j \in [q], i \neq j$. If so, SIM aborts. We refer to this bad event as $\mathsf{ROCollision}$.

Otherwise, SIM initializes the following values:

- Parameters for DS by performing $\mathsf{par_{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$. SIM then sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{par_{DS}})$.
- The sets $\mathsf{Q_{cm}} \leftarrow \emptyset, \mathsf{Q_{sig}} \leftarrow \emptyset$ of $\mathsf{H_{cm}}, \mathsf{H_{sig}}$ queries and their responses, respectively.
- The set $\mathsf{Q_{msg}} \leftarrow \emptyset$ of messages msg_k signed by honest parties during $\mathcal{O}^{\mathsf{Sign}_2}$.
- The set $\mathsf{Q}_m \leftarrow \emptyset$ of messages queried by $\mathcal{A}$ to $\mathcal{O}^{\mathsf{Sign}_2}$.
- The set $\mathsf{Q_{st}} \leftarrow \emptyset$ of participants' states for all signing sessions initiated by $\mathcal{A}$.
- The set $\mathsf{Q_{cm\text{-}wait}} \leftarrow \emptyset$ of potential $\mathsf{Q_{cm}}$ stored values.
- The counter $\mathsf{ctr} \leftarrow 1$.

$\mathcal{A}$ is given access to random oracles $\mathsf{H_{cm}}, \mathsf{H_{sig}}$, which SIM may program.

Static Corruption. SIM then runs $\mathcal{A}(\mathsf{par})$. $\mathcal{A}$ chooses the total number of potential signers n, the threshold $t + 1 \leq n$, and the set of corrupt parties $\mathsf{cor} \leftarrow \{j\}, j \in [n], |\mathsf{cor}| \leq t$, which are fixed for the rest of the protocol. If $t+1 > n$ or $\mathsf{cor} \not\subset [n]$ or $|\mathsf{cor}| > t$, SIM outputs $\perp$. Otherwise, SIM sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which SIM does in the next step.

Simulating KeyGen. SIM simulates the key generation algorithm (Fig. 8) using its DL challenge $\dot{X}$ as follows.

1. SIM generates public and private DS keys honestly $(\hat{X}_i, \hat{x}_i) \leftarrow\!\!\$\ \mathsf{DS.KeyGen}()$, for all $i \in [n]$.
2. SIM sets the threshold public key $\mathsf{pk} \leftarrow \dot{X}$.
3. SIM simulates a Shamir secret sharing of the discrete logarithm of pk by performing the following steps. Assume without loss of generality that $|\mathsf{cor}| = t$.
   (a) SIM samples t random values $x_j \leftarrow\!\!\$\ \mathbb{Z}_p$ for $j \in \mathsf{cor}$.

(b) Let f be the polynomial whose constant term is the challenge $f(0) = \dot{x}$ (which SIM does not know) and for which $f(j) = x_j$ for all $j \in \mathsf{cor}$. SIM computes the $t+1$ Lagrange polynomials $\{L'_0(Z), \{L'_j(Z)\}_{j\in\mathsf{cor}}\}$ relating to the set (of x-coordinates) $0 \cup \mathsf{cor}$. (See Section 3 for notation.)

(c) For all $j \in \mathsf{cor}$, SIM sets the public key share as $X_j \leftarrow g^{x_j}$. For all $i \in \mathsf{hon}$, SIM computes:

$$X_i = \mathsf{pk}^{L'_{0,i}} \prod_{j\in\mathsf{cor}} g^{x_j L'_{j,i}}$$

where $L'_{j,i}$ is the ith coefficient of $L'_j(Z) = L'_{j,0} + L'_{j,1}Z + \cdots + L'_{j,t}Z^t$.

4. SIM sets $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ and $\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i)$ for all $i \in [n]$.

The threshold public key is $\mathsf{pk} = g^{f(0)} = \dot{X}$ with corresponding secret key $\mathsf{sk} = \dot{x}$. SIM then runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_1},\mathsf{Sign}_2,\mathsf{Sign}_3}(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in\mathsf{cor}}, \mathsf{st}_{\mathcal{A}})$.

Simulating Random Oracle Queries. When $\mathcal{A}$ queries H_{cm} or $\mathsf{H}_{\mathsf{sig}}$, SIM simulates the response as follows.

$\underline{\mathsf{H}_{\mathsf{cm}}:}$ When $\mathcal{A}$ queries H_{cm} on (i, R_i), SIM checks whether Q_{cm} contains an entry $(i, R_i, \mathsf{cm}_i, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, and if so, it returns cm_i. (Note: $\cdot$ means a defined entry z_i, or an undefined entry $\perp$.)

Else, SIM sets $\mathsf{cm}_i \leftarrow h_{\mathsf{ctr}}$ and updates $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(i, R_i, \mathsf{cm}_i, \perp)\}$. It then increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and returns cm_i.

$\underline{\mathsf{H}_{\mathsf{sig}}:}$ When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), SIM checks whether $(X, m, R, c, \Delta) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c.

Else, SIM sets $c \leftarrow h_{\mathsf{ctr}}$ and $\Delta \leftarrow \mathsf{ctr}$. It appends (X, m, R, c, Δ) to $\mathsf{Q}_{\mathsf{sig}}$. It then increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and returns c.

Simulating Sparkle+ Signing. SIM handles $\mathcal{A}$'s signing queries as follows.

Round 1 ($\mathcal{O}^{\mathsf{Sign}_1}(k)$): In the first round of signing, parties form commitments cm_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_1}$ for participant identifier k, SIM first checks that $k \in \mathsf{hon}$. If so, then SIM does the following:

1. Samples $\mathsf{eid} \leftarrow\!\!\$\ \{0,1\}^*$.
2. Sets $\mathsf{cm}_k \leftarrow h_{\mathsf{ctr}}$, and increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$.
3. Sets $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, \perp, \perp, \hat{\mathsf{sk}}_k)$, where $\hat{\mathsf{sk}}_k := (\perp, \hat{x}_k, \mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$.
4. Updates $\mathsf{Q}_{\mathsf{st}} \leftarrow \mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] \leftarrow \mathsf{st}_{k,\mathsf{eid},1}$.
5. Finally, SIM returns cm_k to $\mathcal{A}$.

Round 2 ($\mathcal{O}^{\mathsf{Sign}_2}(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$): In the second round of signing, each party i in the signing set $\mathcal{S}$ takes as input a set of commitments $\{\mathsf{cm}_j\}_{j\in\mathcal{S}}$ and reveals its nonce R_i such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(i, R_i)$. Each party also signs their message $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j\in\mathcal{S}})$ using DS.

When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_2}$ on $(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ for $k \in \mathsf{hon}$, SIM first performs the checks in $\mathcal{O}^{\mathsf{Sign}_2}$ (Fig. 7) and Sign_2 (Fig. 8). If they pass, then, in particular $\mathsf{st}_{k,\mathsf{eid},1} = (\mathsf{cm}_k, \perp, \perp, \hat{\mathsf{sk}}_k)$. SIM then does the following:

1. Updates $\mathsf{Q}_m \leftarrow \mathsf{Q}_m \cup \{m\}$.

2. Next, it checks if $(k, R_k, \mathsf{cm}_k, z_k, (\mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})) \in \mathsf{Q}_{\mathsf{cm\text{-}wait}}$ for some R_k, z_k. If the check holds, SIM sets R_k accordingly. Then, SIM adds $(k, R_k, \mathsf{cm}_k, z_k)$ to Q_{cm} and $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ to $\mathsf{Q}_{\mathsf{msg}}$, signs msg_k as $\hat{\sigma}_k \leftarrow \mathsf{DS.Sign}(\hat{x}_k, \mathsf{msg}_k)$, updates $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$, and returns $(R_k, \hat{\sigma}_k)$.
3. If the check fails, then for each commitment $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$, SIM checks if it has an entry $(i, R_i, \mathsf{cm}_i, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$.
   (a) If there exists some $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $i \in \mathcal{S} \cap \mathsf{hon}$ and cm_i has a corresponding R_i defined in Q_{cm}, then SIM goes to Case 1.
   (b) If there exists some $\mathsf{cm}_j \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $j \in \mathcal{S} \cap \mathsf{cor}$ and cm_j does not have a corresponding R_j defined in Q_{cm}, then SIM goes to Case 1.
   (c) Otherwise, SIM goes to Case 2.

Case 1 (Invalid Query). In this case, SIM does the following:

1. Sample random $R_k \leftarrow_\$ \mathbb{G}$.
2. Program $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(k, R_k, \mathsf{cm}_k, \perp)\}$.
3. Follow the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k.
4. Add $\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature $\hat{\sigma}_k$ on msg_k.
5. Set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
6. Update $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$.
7. Return $(R_k, \hat{\sigma}_k)$.

Case 2 (Valid Query). In this case, SIM does the following:

1. Sets $c \leftarrow h_{\mathsf{ctr}}$ and $\Delta \leftarrow \mathsf{ctr}$ and increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$.
2. For all $i \in \mathcal{S} \cap \mathsf{hon}$, SIM samples random $z_i \leftarrow_\$ \mathbb{Z}_p$ and computes $R_i \leftarrow g^{z_i} X_i^{-c\lambda_i}$, where λ_i is the Lagrange coefficient for party i in the set $\mathcal{S}$.
3. Next, SIM programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(k, R_k)$. That is, SIM adds the entry $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(k, R_k, \mathsf{cm}_k, z_k)\}$. Additionally, for all $i \in \mathcal{S} \cap \mathsf{hon}$, $i \neq k$, SIM adds the entry $\mathsf{Q}_{\mathsf{cm\text{-}wait}} \leftarrow \mathsf{Q}_{\mathsf{cm\text{-}wait}} \cup \{(i, R_i, \mathsf{cm}_i, z_i, (\mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}))\}$.
4. Next, for each $j \in \mathcal{S} \cap \mathsf{cor}$, SIM obtains R_j by looking up $(j, R_j, \mathsf{cm}_j, \perp) \in \mathsf{Q}_{\mathsf{cm}}$.
5. SIM then computes $R = \prod_{\ell\in\mathcal{S}} R_\ell$.
6. If there is a defined entry $(\mathsf{pk}, m, R, c', \Delta') \in \mathsf{Q}_{\mathsf{sig}}$ for some c', Δ', then SIM aborts. We refer to this bad event as $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$.
7. Otherwise, SIM programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. Specifically, SIM adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R, c, \Delta)\}$.
8. SIM follows the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k and adds $\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature $\hat{\sigma}_k$ on msg_k.
9. It then sets $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$ and updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$.
10. Finally, SIM returns $(R_k, \hat{\sigma}_k)$.

Round 3 ($\mathcal{O}^{\mathsf{Sign}_3}(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$): In the third round of signing, each party $i \in \mathcal{S}$ outputs the remaining part of its partial signature, namely z_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_3}$ on $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$ for $k \in \mathsf{hon}$, SIM first performs the checks in $\mathcal{O}^{\mathsf{Sign}_3}$ (Fig. 7) and Sign_3 (Fig. 8). If they pass, then, in particular $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \bot, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$. SIM then does the following:

1. Checks that $\mathsf{DS.Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) = 1$ for all $i \in \mathcal{S}$, where $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}})$, and returns $\bot$ if not.
2. Checks whether $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(i, R_i)$ for all $i \in \mathcal{S}$ and returns $\bot$ if not.
3. Looks up $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}) \in \mathsf{Q}_{\mathsf{msg}}$ and aborts if no such entry exists. We refer to this bad event as $\mathsf{ForgeDS}_k$.
4. Check that there exists cm_i such that $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}) \in \mathsf{Q}_{\mathsf{msg}}$ for all $i \in \mathcal{S} \cap \mathsf{hon}$ and abort if not. We refer to this bad event as $\mathsf{ForgeDS}_i$.
5. Else, SIM retrieves $(k, R_k, \mathsf{cm}_k, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ and aborts if $\cdot = \bot$. We refer to this bad event as $\mathsf{FromInvalid}_k$. Otherwise, SIM retrieves z_k from $(k, R_k, \mathsf{cm}_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$.
6. SIM updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 3] \leftarrow z_k$.
7. Finally, SIM returns z_k.

Analysis of SIM's Simulation. SIM's simulation of key generation and $\mathcal{O}^{\mathsf{Sign}_1}$ is perfect and no abort conditions occur.

SIM's simulation of $\mathcal{O}^{\mathsf{Sign}_2}$ is perfect. Before revealing an honest R_k, SIM always programs H_{cm} such that $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(k, R_k)$, as would be the case in Game 2. Each R_k is randomized by z_k. SIM responds if and only if $\mathsf{st}_{k,\mathsf{eid},1}$ exists as in Game 2. The signatures $\hat{\sigma}_k$ verify on msg_k.

$\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$ occurs with maximum probability $\frac{q}{p}$. Indeed, SIM checks to see if there is a defined entry $(\mathsf{pk}, m, R, c', \Delta') \in \mathsf{Q}_{\mathsf{sig}}$ for some c', Δ' before attempting to program $\mathsf{H}_{\mathsf{sig}}$. The probability that $\mathcal{A}$ guesses R in some hash query is bounded by $\frac{q}{p}$ because R is randomized by c.

SIM's simulation of $\mathcal{O}^{\mathsf{Sign}_3}$ is perfect. If there is a defined entry $(k, R_k, \mathsf{cm}_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$, then $R_k = g^{z_k} X_k^{-c\lambda_k}$ for $c = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. Note that the aggregate R is computed consistently if all honest R_i's are computed in the same $\mathcal{O}^{\mathsf{Sign}_2}$ query with the same $(m, \mathcal{S}, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}})$ during Case 2 (Valid Query). Due to $\mathsf{ForgeDS}_i$, SIM only needs to respond to such queries. Each corrupt R_j hashes to a unique cm_j.

If two (i, R_i) and (i', R_i')'s hash to the same cm_i, then ROCollision has occurred.

$\mathsf{ForgeDS}_k$ and $\mathsf{ForgeDS}_i$ cannot occur unless Game 1 also would abort.

$\mathsf{FromInvalid}_k$ occurs in Round 3 if some honest R_k is computed in an $\mathcal{O}^{\mathsf{Sign}_2}$ query during Case 1 (Invalid Query). If this occurs in Step 3 (a) of the simulator's Round 2, then there exists some $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $i \in \mathcal{S} \cap \mathsf{hon}$ and cm_i has a corresponding R_i defined in Q_{cm}. Then $\mathsf{ForgeDS}_k$ or $\mathsf{ForgeDS}_i$ will occur because either (i) cm_i was determined according to a different Round 2 query $(m', \mathcal{S}', \{\mathsf{cm}_i'\}_{i\in\mathcal{S}'})$ to cm_k, or else (ii) $\mathcal{A}$ chose R_i and queried $\mathsf{H}_{\mathsf{cm}}(i, R_i)$ to get cm_i. If this occurs in Step 3 (b), then there exists some $\mathsf{cm}_j \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $j \in \mathcal{S} \cap \mathsf{cor}$ and cm_j does not have a corresponding R_j defined in Q_{cm},

but an adversary later finds R_j such that $\mathsf{H}_{\mathsf{cm}}(j, R_j) = \mathsf{cm}_j$. This occurs with maximum probability $\frac{q}{p}$.

In summary, SIM aborts with negligible probability in the following cases:

1. $\underline{\mathsf{ROCollision}}$: SIM aborts due to $\mathsf{ROCollision}$ with probability $\frac{q^2}{2p}$, which occurs when it receives any inputs $\{h_1, \ldots, h_q\}$ that collide.
2. $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$: SIM aborts due to $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$ with probability q/p.
3. $\underline{\mathsf{ForgeDS}_k}$: When SIM aborts due to $\mathsf{ForgeDS}_k$, then Game 1 also aborts.
4. $\underline{\mathsf{ForgeDS}_i}$: When SIM aborts due to $\mathsf{ForgeDS}_i$, then Game 1 also aborts.
5. $\underline{\mathsf{FromInvalid}_k}$: SIM aborts due to $\mathsf{FromInvalid}_k$ with probability q/p.

Output. When $\mathcal{A}$ terminates with output (m^*, σ^*), SIM first checks that it is a valid forgery, by checking that $m^* \notin \mathsf{Q}_m$ and $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$. If either check fails, SIM returns 0.

Otherwise, SIM looks up the entry $(\mathsf{pk}, m^*, R^*, c^*, I) \in \mathsf{Q}_{\mathsf{sig}}$ corresponding to $\mathcal{A}$'s forgery. It sets $\mathsf{out} \leftarrow \sigma^*$. SIM then outputs (I, out).

Difference between Game 2 and Game 1. Let W_2 be the event that $\mathcal{A}$ wins Game 2. The only additional advantage in the simulation by SIM over Game 1 is given by bad events $\mathsf{ROCollision}$, $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$, and $\mathsf{FromInvalid}_k$. As such,

$$\begin{aligned} |\Pr[W_1] - \Pr[W_2]| &= \Pr[\mathsf{ROCollision}] + \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}] + \Pr[\mathsf{FromInvalid}_k] \\ &= \frac{q^2}{2p} + \frac{2q}{p} \end{aligned} \tag{11}$$

Reduction $\mathcal{B}_2$ Against DL. Finally, we construct our PPT reduction $\mathcal{B}_2$ against the DL assumption (Fig. 5). Specifically, we show how $\mathcal{B}_2$ uses $\mathsf{Fork}^{\mathsf{SIM}}$ as a subroutine such that:

$$\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa) \geq \mathsf{accept}(\mathsf{Fork}^{\mathsf{SIM}}(X)) - 2\Pr[\mathsf{ROCollision}] - \ldots - 2\Pr[\mathsf{FromInvalid}_k] \tag{12}$$

where X is the instance given by the challenger in the DL game.

Initialization. $\mathcal{B}_2$ is initialized by the DL challenger with input the group description $(\mathbb{G}, p, g)$ and a DL challenge $\dot{X} \in \mathbb{G}$. The goal of $\mathcal{B}_2$ is to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$.

Execution. $\mathcal{B}_2$ then runs the general forking algorithm $\mathsf{Fork}^{\mathsf{SIM}}(X)$ as described in Figure 9, on the simulator algorithm SIM and its instance $X = ((\mathbb{G}, p, g), \dot{X})$. With non-negligible probability lower-bounded by Equation 6, $\mathsf{Fork}^{\mathsf{SIM}}(X)$ will output the accepting answer $(I, \mathsf{out}, \mathsf{out}')$ such that:

$$\begin{gathered} h_I = c^*,\ h'_I = c^{**}, \\ \sigma^* \leftarrow \mathsf{out},\ \sigma^{**} \leftarrow \mathsf{out}' \\ (R^*, z^*) \leftarrow \sigma^*,\ (R^*, z^{**}) \leftarrow \sigma^{**} \end{gathered}$$

$\mathcal{B}_2$ then solves for $\mathsf{sk} = \frac{z^* - z^{**}}{c^* - c^{**}}$, where $\mathsf{sk} = \dot{x}$ and where the DL challenge is $\dot{X} = g^{\dot{x}}$. $\mathcal{B}_2$ then returns $\dot{x}$ as its output to the DL challenger.

Finishing the Proof. Equation 6 lower bounds $\mathsf{accept}(\mathsf{Fork}^{\mathsf{SIM}}(X))$. The probability that $\mathcal{B}_2$ succeeds is therefore defined by combining Equations 6, 8, 10, 11, and 12, which gives Equation 3. This completes the proof. □

B Proof of Adaptive Security up to $t/2$ Corruptions

In this section, we prove the adaptive security of Sparkle+ against up to $t/2$ corruptions under the algebraic one-more discrete logarithm assumption (AOMDL) (Fig. 5) in the random oracle model (ROM). The reason the allowed corruption is $t/2$ is that, in order to extract an AOMDL solution, the reduction needs to rewind the adversary once, and there is no guarantee that the adversary will corrupt the same set of parties after the fork as it did during the first iteration of the adversary. When the adversary can corrupt only $t/2$ parties, this causes no issues, as the total number of corruptions over both iterations does not exceed t. If the adversary could corrupt more parties, then the reduction would query its discrete logarithm oracle more than t times and would lose the $t+1$-aomdl game.

Proof. (of Theorem 2.) We prove the theorem by a sequence of games.

Game 0. This is the adaptive unforgeability game as defined in Figure 7, instantiated with Sparkle+.

Let $\mathcal{A}$ be a PPT adversary attempting to break the adaptive unforgeability of Sparkle+ (Fig. 7) that makes up to q_H queries to H_{cm} and $\mathsf{H}_{\mathsf{sig}}$, and q_S queries to its signing oracles, each time it is executed. $\mathcal{A}$ is allowed to make $t/2$ adaptive corruptions. We assume, without loss of generality, that $\mathcal{A}$ makes a query to $\mathsf{H}_{\mathsf{sig}}$ on its forgery (pk, m^*, R^*). Then, let $q = q_H + q_S + 1$.

Let W_0 be the event that $\mathcal{A}$ wins Game 0. Then, the advantage of $\mathcal{A}$ is:

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1/2) = |\Pr[W_0]| \tag{13}$$

Here, $\mathsf{frac} = 1/2$ to allow $\mathcal{A}$ to corrupt a $\mathsf{frac} \cdot t = t/2$ parties.

Game 1. The changes from Game 0 to Game 1 are the same as in the proof of Theorem 1.

Reduction $\mathcal{B}'_1$ to the EUF-CMA Security of DS. We now define a reduction $\mathcal{B}'_1$ against the EUF-CMA security of DS that uses $\mathcal{A}$ as a subroutine. We show that when $\mathcal{A}$ wins Game 1, then $\mathcal{B}'_1$ wins the EUF-CMA game with non-negligible probability.

The only difference in $\mathcal{B}'_1$'s simulation to the reduction $\mathcal{B}_1$ in Game 1 for Theorem 1 is that $\mathcal{B}'_1$ must additionally respond to corruption queries $\mathcal{O}^{\mathsf{Corrupt}}$.

Simulating Corruption Oracle Queries. When $\mathcal{A}$ queries its corruption oracle on $k \in \mathsf{hon}$, $\mathcal{B}'_1$ does as follows:

1. If $k \neq \tau$, then $\mathcal{B}_1'$ follows the protocol honestly.
2. If $k = \tau$, then $\mathcal{B}_1'$ aborts. We refer to this abort event as $\mathsf{CorruptParty}\tau$.

Analysis of $\mathcal{B}_1'$'s Simulation. The only difference between $\mathcal{B}_1$'s simulation in Game 1 of Theorem 1 and $\mathcal{B}_1'$'s simulation is the occurrence of $\mathsf{CorruptParty}\tau$. At the end of the game, there are $(n - t/2)$ non-corrupted parties. The probability that τ is one of them (i.e., that $\mathsf{CorruptParty}\tau$ does not occur) is $\frac{(n-t/2)}{n}$.

Difference between Game 0 and Game 1. If $\mathsf{ForgeDS}$ does not occur, then Game 0 and Game 1 are identical. If $\mathsf{ForgeDS}$ occurs, then the probability that $k = \tau$ is at least $\frac{1}{n}$. The probability that $\mathsf{CorruptParty}\tau$ does not occur, i.e., τ is not corrupted, is greater than $\frac{(n-t/2)}{n}$.

The additional advantage for $\mathcal{A}$ in Game 1, assuming that $\mathsf{ForgeDS}$ occurs for participant τ, is upper-bounded by the advantage that $\mathcal{B}_1'$ wins its EUF-CMA game against DS, and the probability that $\mathsf{CorruptParty}\tau$ does not occur.

Let W_1 be the event that $\mathcal{A}$ wins Game 1. Then:

$$|\Pr[W_0] - \Pr[W_1]| \leq \frac{n^2}{(n - t/2)} \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1', \mathsf{DS}}(\kappa) \tag{14}$$

Game 2. Next, we define a game that rules out several bad events: $\mathsf{ROCollision}$, $\mathsf{H_{sig}Guess}$, $\mathsf{ForgeDS}_k$, $\mathsf{ForgeDS}_i$, $\mathsf{FromInvalid}_k$, $\mathsf{H_{sig}Guess_{OneHon}}$, $\mathsf{H_{sig}Guess_{LastHonCor}}$, $\mathsf{H_{cm}Guess}$. That is, the game aborts if any of these events occurs.

We next define a simulator algorithm SIM' that simulates Game 1 to an adaptive unforgeability adversary $\mathcal{A}$. The simulation by SIM' is similar to that of SIM in Game 2 for Theorem 1, with the exception of the instance X that SIM' accepts as input, how key generation is performed, what information SIM' stores in $\mathsf{Q_{cm}}$, and how SIM' additionally simulates $\mathcal{O}^{\mathsf{Corrupt}}$.

Additionally, SIM' is given an oracle $\mathcal{O}^{\mathsf{dl}}$, which it can query on a group element, together with an algebraic representation, and receive its discrete logarithm. SIM' can query this oracle up to t times.

Setup. SIM' accepts as input an instance X, which is a tuple consisting of the group description $(\mathbb{G}, p, g)$ and $t + 1$ AOMDL challenges $(Y_0, \ldots, Y_t) \in \mathbb{G}^{t+1}$. In addition, SIM' accepts as input a set of $q = q_H + q_S + 1$ values $\{h_1, \ldots, h_q\}$, which it will use to program its random oracles responses.

Next, SIM' checks for collisions $h_i = h_j$ for $i, j \in [q], i \neq j$. If so, SIM' aborts. We refer to this bad event as $\mathsf{ROCollision}$.

Otherwise, SIM' initializes the following values:

- Parameters for DS by performing $\mathsf{par_{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$. SIM then sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{par_{DS}})$.
- The sets $\mathsf{Q_{cm}} \leftarrow \emptyset, \mathsf{Q_{sig}} \leftarrow \emptyset$ of $\mathsf{H_{cm}}, \mathsf{H_{sig}}$ queries and their responses, respectively.
- The set $\mathsf{Q_{msg}} \leftarrow \emptyset$ of messages msg_k signed by honest parties during $\mathcal{O}^{\mathsf{Sign}_2}$.
- The set $\mathsf{Q}_m \leftarrow \emptyset$ of messages queried by $\mathcal{A}$ to $\mathcal{O}^{\mathsf{Sign}_2}$.

– The set $\mathsf{Q}_{\mathsf{st}} \leftarrow \emptyset$ to be the participants' states for all signing sessions initiated by $\mathcal{A}$.
– The set $\mathsf{Q}_{\mathsf{cm\text{-}wait}} \leftarrow \emptyset$ of potential Q_{cm} and $\mathsf{Q}_{\mathsf{sig}}$ stored values.
– The counter $\mathsf{ctr} \leftarrow 1$.

$\mathcal{A}$ is given access to random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$, which SIM' may program.

Static Corruption. SIM' then runs $\mathcal{A}(\mathsf{par})$. $\mathcal{A}$ chooses the total number of potential signers n and the threshold $t+1 \leq n$. Additionally, $\mathcal{A}$ may choose a set of parties to corrupt initially: $\mathsf{cor} \leftarrow \{j\}, j \in [n], |\mathsf{cor}| \leq t/2$. If $t+1 > n$ or $\mathsf{cor} \not\subset [n]$ or $|\mathsf{cor}| > t/2$, SIM' outputs $\perp$. Otherwise, SIM' sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which SIM' does in the next step.

Simulating KeyGen. SIM' simulates the key generation algorithm (Fig. 8) using its AOMDL challenge $(Y_0, \ldots, Y_t)$ as follows.

1. SIM' generates public and private DS keys honestly $(\hat{X}_i, \hat{x}_i) \leftarrow\!\!\$\ \mathsf{DS.KeyGen}()$, for all $i \in [n]$.
2. For all $i \in [n]$, SIM' sets the public key share as:

$$X_i = Y_0 Y_1^i \cdots Y_t^{i^t}$$

   which is implicitly equal to $g^{f(i)}$ for the polynomial $f(Z) = y_0 + y_1 Z + \cdots + y_t Z^t$. The threshold public key is $\mathsf{pk} = g^{f(0)} = Y_0$ with corresponding secret key $\mathsf{sk} = y_0$.
3. SIM' obtains the initial corrupt secret key shares by querying $x_j = f(j) \leftarrow \mathcal{O}^{\mathsf{dl}}(X_j)$ (with representation $(1, j, \ldots, j^t)$) for all $j \in \mathsf{cor}$.
4. SIM' sets $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ and $\mathsf{sk}_i \leftarrow (x_i, \hat{x}_i)$ for all $i \in [n]$.

SIM' then runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_1}, \mathsf{Sign}_2, \mathsf{Sign}_3, \mathsf{Corrupt}}(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in\mathsf{cor}}, \mathsf{st}_{\mathcal{A}})$.

Simulating Random Oracle Queries. When $\mathcal{A}$ queries H_{cm} or $\mathsf{H}_{\mathsf{sig}}$, SIM' simulates the response as follows.

$\underline{\mathsf{H}_{\mathsf{cm}}:}$ When $\mathcal{A}$ queries H_{cm} on (i, R_i), SIM' checks whether Q_{cm} contains an entry $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, and if so, it returns cm_i.

Else, SIM' sets $\mathsf{cm}_i \leftarrow h_{\mathsf{ctr}}$ and updates $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(i, R_i, \perp, \mathsf{cm}_i, \perp, \perp)\}$. It then increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and returns cm_i.

$\underline{\mathsf{H}_{\mathsf{sig}}:}$ SIM' simulates $\mathcal{A}$'s queries to $\mathsf{H}_{\mathsf{sig}}$ identically to SIM for Theorem 1.

Simulating Sparkle+ Signing. SIM' handles $\mathcal{A}$'s signing queries as follows.

Round 1 ($\mathcal{O}^{\mathsf{Sign}_1}(k)$): SIM' simulates $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Sign}_1}$ identically to SIM for Theorem 1, setting additional empty entries in Q_{cm} as required.

Round 2 ($\mathcal{O}^{\mathsf{Sign}_2}(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$): When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_2}$ on $(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ for $k \in \mathsf{hon}$, SIM' performs the same steps as SIM for Theorem 1, with the following exceptions:

1. Updates $\mathsf{Q}_m \leftarrow \mathsf{Q}_m \cup \{m\}$.
2. Next, it checks if $(k, R_k, \mathsf{cm}_k, z_k, (\mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}), c_k, c) \in \mathsf{Q}_{\mathsf{cm\text{-}wait}}$ for some R_k, z_k, c_k, c. If the check holds, SIM' sets R_k accordingly and:
   (a) Adds $(k, R_k, \perp, \mathsf{cm}_k, c_k, z_k)$ to Q_{cm} and $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ to $\mathsf{Q}_{\mathsf{msg}}$, signs msg_k as $\hat{\sigma}_k \leftarrow \mathsf{DS.Sign}(\hat{x}_k, \mathsf{msg}_k)$, and updates $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
   (b) Checks if there exists $\{R_i\}_{i\in\mathcal{S}, i\neq k}$ such that $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}$, $i \neq k$. If the check holds, then SIM' computes $R = \prod_{i\in\mathcal{S}} R_i$. If $(\mathsf{pk}, m, R, c', \Delta') \in \mathsf{Q}_{\mathsf{sig}}$ for some c', Δ', then SIM' aborts. We refer to this bad event as $\mathsf{H_{sig}Guess}$. Otherwise, SIM' programs $c \leftarrow \mathsf{H_{sig}}(\mathsf{pk}, m, R)$. That is, SIM' adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R, c, \Delta)\}$.

   Finally, SIM' returns $(R_k, \hat{\sigma}_k)$.
3. Else, suppose the check fails, i.e., that $(k, \cdot, \mathsf{cm}_k, \cdot, (\mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}), \cdot, \cdot) \notin \mathsf{Q}_{\mathsf{cm\text{-}wait}}$. For each commitment $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$, SIM' checks if it has an entry $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$.
   (a) If there exists some $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $i \in \mathcal{S} \cap \mathsf{hon}$ and cm_i has a corresponding R_i defined in Q_{cm}, then SIM goes to Case 1.
   (b) If there exists some $\mathsf{cm}_j \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ such that $j \in \mathcal{S} \cap \mathsf{cor}$ and cm_j does not have a corresponding R_j defined in Q_{cm}, then SIM' goes to Case 1.
   (c) Otherwise, SIM' goes to Case 2.

Case 1 (Invalid Query). In this case, SIM' does the following:

1. Sample random $r_k \leftarrow_\$ \mathbb{Z}_p$ and set $R_k \leftarrow g^{r_k}$.
2. Program $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(k, R_k, r_k, \mathsf{cm}_k, \perp, \perp)\}$.
3. Follow the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k.
4. Add $\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature on msg_k.
5. Set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
6. Update $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$.
7. Return $(R_k, \hat{\sigma}_k)$.

Case 2 (Valid Query). In this case, SIM' does the following:

1. Sets $c \leftarrow h_{\mathsf{ctr}}$ and $\Delta \leftarrow \mathsf{ctr}$ and increments $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$.
2. For all $i \in \mathcal{S} \cap \mathsf{hon}$, SIM' samples random $z_i \leftarrow_\$ \mathbb{Z}_p$ and computes $R_i \leftarrow g^{z_i} X_i^{-c\lambda_i}$, where λ_i is the Lagrange coefficient for party i in the set $\mathcal{S}$.
3. SIM' programs $\mathsf{cm}_k \leftarrow \mathsf{H_{cm}}(k, R_k)$. That is, SIM' adds the entry $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(k, R_k, \perp, \mathsf{cm}_k, c\lambda_k, z_k)\}$. Additionally, for all $i \in \mathcal{S} \cap \mathsf{hon}$, $i \neq k$, SIM' adds the entry $\mathsf{Q}_{\mathsf{cm\text{-}wait}} \leftarrow \mathsf{Q}_{\mathsf{cm\text{-}wait}} \cup \{(i, R_i, \mathsf{cm}_i, z_i, (\mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}), \lambda_i c, c)\}$.
4. SIM' follows the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k, where $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, and adds $\mathsf{Q}_{\mathsf{msg}} \leftarrow \mathsf{Q}_{\mathsf{msg}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature on msg_k.
5. It then sets $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$ and updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$.

6. Next, if k is the only honest party in $\mathcal{S}$ then SIM' looks up $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}, i \neq k$. SIM' then computes $R = \prod_{\ell \in \mathcal{S}} R_\ell$. If $(\mathsf{pk}, m, R, c', \Delta') \in \mathsf{Q}_{\mathsf{sig}}$ for some c', Δ', then SIM' aborts. We refer to this bad event as $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}$. Otherwise, SIM' programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. That is SIM' adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R, c, \Delta)\}$.
7. Finally, SIM' returns $(R_k, \hat{\sigma}_k)$.

Round 3 ($\mathcal{O}^{\mathsf{Sign}_3}(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i \in \mathcal{S}})$): In the third round of signing, each party i in the signing set $\mathcal{S}$ outputs the remaining part of its partial signature, namely z_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_3}$ on $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i \in \mathcal{S}})$ for $k \in \mathsf{hon}$, SIM' first performs the checks in $\mathcal{O}^{\mathsf{Sign}_3}$ (Fig. 7) and Sign_3 (Fig. 8). If they pass, then, in particular $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i \in \mathcal{S}}, \hat{\mathsf{sk}}_k)$. SIM' then does the following:

1. Checks that $\mathsf{DS.Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) = 1$ for all $i \in \mathcal{S}$, where $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j \in \mathcal{S}})$, and returns $\perp$ if not.
2. Checks whether $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(i, R_i)$ for all $i \in \mathcal{S}$ and returns $\perp$ if not.
3. Looks up $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i \in \mathcal{S}}) \in \mathsf{Q}_{\mathsf{msg}}$ and aborts if no such entry exists. We refer to this bad event as $\mathsf{ForgeDS}_k$.
4. Check that there exists cm_i such that $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_\ell\}_{j\ell \in \mathcal{S}}) \in \mathsf{Q}_{\mathsf{msg}}$ for all $i \in \mathcal{S} \cap \mathsf{hon}$ and abort if not. We refer to this bad event as $\mathsf{ForgeDS}_i$.
5. Else, SIM retrieves $(k, R_k, \cdot, \mathsf{cm}_k, \cdot, \cdot) \leftarrow \mathsf{Q}_{\mathsf{cm}}$ and aborts if $\cdot = \perp$. We refer to this bad event as $\mathsf{FromInvalid}_k$. Otherwise SIM' retrieves z_k from $(k, R_k, \cdot, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$.
6. SIM' updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 3] \leftarrow z_k$.
7. Finally, SIM returns z_k.

Simulating Corruption Oracle Queries. $\mathcal{A}$ may at any time corrupt an honest party k by querying $\mathcal{O}^{\mathsf{Corrupt}}(k)$. Upon receiving a corruption query, SIM' first checks that $k \in \mathsf{hon}$, returning $\perp$ if not. Otherwise, SIM' queries its DL oracle $\mathcal{O}^{\mathsf{dl}}$ on $X_k = g^{f(k)}$ (with representation $(1, k, \ldots, k^t)$) to obtain the secret key $x_k = f(k)$. Then, for each $\mathsf{st}_{k,\mathsf{eid},\cdot}$, SIM' chooses R_k and r_k as follows:

1. If there exists R_k, r_k such that $(k, R_k, r_k, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, then SIM' sets R_k and r_k accordingly. SIM' updates $\hat{\mathsf{sk}}_k$ to sk_k and sets $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i \in \mathcal{S}}, \mathsf{sk}_k)$ and $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.
2. If there exists R_k, c_k, z_k such that $(k, R_k, \perp, \mathsf{cm}_k, c_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$, then SIM' sets R_k accordingly and sets r_k as:

$$r_k = z_k - c_k x_k$$

   SIM' updates $\hat{\mathsf{sk}}_k$ to sk_k and sets $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i \in \mathcal{S}}, \mathsf{sk}_k)$ and $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.
3. If there does not exist R_k such that $(k, R_k, \cdot, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, then
   (a) SIM' samples $r_k \leftarrow_{\$} \mathbb{Z}_p$, sets $R_k \leftarrow g^{r_k}$, and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(k, R_k)$ by adding $(k, R_k, r_k, \mathsf{cm}_k, \cdot, \cdot)$ to Q_{cm}.

(b) For each $R'_k, z'_k, (\mathcal{S}', m', \{\mathsf{cm}'_i\}_{i\in\mathcal{S}}), c'_k, c')$ with

$$(k, R'_k, \mathsf{cm}_k, z'_k, (\mathcal{S}', m', \{\mathsf{cm}'_i\}_{i\in\mathcal{S}}), c'_k, c') \in \mathsf{Q}_{\mathsf{cm\text{-}wait}}$$

SIM' tests if there exists $\{R'_i\}_{i\in\mathcal{S}',i\neq k}$ such that $(i, R'_i, \cdot, \mathsf{cm}'_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}'$, $i \neq k$. If the check holds, then SIM' computes $R' \leftarrow R_k \prod_{\ell\in\mathcal{S}',\ell\neq k} R'_\ell$. If $(\mathsf{pk}, m, R', \cdot, \cdot) \in \mathsf{Q}_{\mathsf{sig}}$, then SIM' aborts. We refer to this bad event as $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}$. Otherwise, SIM' programs $c' \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R')$. That is, SIM' adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R', c', \Delta')\}$.

(c) SIM' updates $\hat{\mathsf{sk}}_k$ to sk_k and sets $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.

4. Else, if there exists R_k such that $(k, R_k, \bot, \mathsf{cm}_k, \bot, \bot) \in \mathsf{Q}_{\mathsf{cm}}$, then SIM' aborts. We refer to this bad event as $\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}$.

Finally, SIM' sets $\mathsf{st}_k \leftarrow \mathsf{Q}_{\mathsf{st}}[k, \cdot, \cdot]$, and outputs $(x_k, \hat{x}_k, \mathsf{st}_k)$.

Analysis of SIM''s Simulation.

SIM''s simulation of $\mathsf{H}_{\mathsf{sig}}$. SIM''s simulation of $\mathsf{H}_{\mathsf{sig}}$ is perfect because each c set in $\mathcal{O}^{\mathsf{Sign}_2}$, Step 3, Case 2, Item 1, is only added to $\mathsf{Q}_{\mathsf{sig}}$ once. Indeed, c can be added to $\mathsf{Q}_{\mathsf{sig}}$ either:

1. In $\mathcal{O}^{\mathsf{Sign}_2}$, Step 2 (b): When $(k, R_k, \mathsf{cm}_k, z_k, (\mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}), c_k, c) \in \mathsf{Q}_{\mathsf{cm\text{-}wait}}$, $\mathsf{H}_{\mathsf{sig}}$ is only programmed if there exists $\{R_i\}_{i\in\mathcal{S},i\neq k}$ such that $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}$, $i \neq k$.
2. In $\mathcal{O}^{\mathsf{Sign}_2}$, Step 3, Case 2, Item 6. When k is the only honest party in $\mathcal{S}$ then we have $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}$, $i \neq k$.
3. In $\mathcal{O}^{\mathsf{Corrupt}}$, Step 3 (b). When $(k, R'_k, \mathsf{cm}_k, z'_k, (\mathcal{S}', m', \{\mathsf{cm}'_i\}_{i\in\mathcal{S}}), c'_k, c') \in \mathsf{Q}_{\mathsf{cm\text{-}wait}}$, $\mathsf{H}_{\mathsf{sig}}$ is only programmed if there exists $\{R'_i\}_{i\in\mathcal{S}',i\neq k}$ such that $(i, R'_i, \cdot, \mathsf{cm}'_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S}'$, $i \neq k$.

Thus, we see that c from $\mathsf{Q}_{\mathsf{cm\text{-}wait}}$ is only added to $\mathsf{Q}_{\mathsf{sig}}$ when the final $\mathsf{cm}_i \in \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}$ is added to Q_{cm}.

SIM''s simulation of KeyGen and $\mathcal{O}^{\mathsf{Sign}_1}$. SIM''s simulation of key generation and $\mathcal{O}^{\mathsf{Sign}_1}$ is perfect and no abort conditions occur.

SIM''s simulation of $\mathcal{O}^{\mathsf{Sign}_2}$. SIM''s simulation of $\mathcal{O}^{\mathsf{Sign}_2}$ is perfect. SIM' responds if and only if $\mathsf{st}_{k,\mathsf{eid},1}$ exists as in Game 2. The signatures $\hat{\sigma}_k$ verify on msg_k. Before revealing an honest R_k, SIM always programs H_{cm} such that $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(k, R_k)$, as would be the case in Game 2. In the real $\mathcal{O}^{\mathsf{Sign}_2}$, the R_k is chosen as a uniformly random group element (during $\mathcal{O}^{\mathsf{Sign}_1}$ but only revealed now). In the simulation, R_k is chosen either:

1. In Step 2 (a). Here, the R_k has previously been randomized by some unique z_k during an alternative $\mathcal{O}^{\mathsf{Sign}_2}$ query that went to Step 3, Case 2, Item 3.
2. In Step 3, Case 1, Item 2. Here, the R_k is sampled uniformly at random.
3. In Step 3, Case 2, Item 4. Here the R_k is randomized by z_k.

It remains to show that $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$ and $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}$ occur with maximum probability $\frac{q}{p}$. Indeed, SIM checks to see if there is a defined entry $(\mathsf{pk}, m, R, c', \Delta') \in \mathsf{Q}_{\mathsf{sig}}$ for some c', Δ'. In both cases, the probability that $\mathcal{A}$ guesses R in some hash query is bounded by $\frac{q}{p}$ because R is randomized R_k, which we have seen is distributed uniformly at random.

$\underline{\mathsf{SIM}'\text{'s simulation of } \mathcal{O}^{\mathsf{Sign}_3}}$. SIM''s simulation of $\mathcal{O}^{\mathsf{Sign}_3}$ is perfect.

Recall SIM' is required to respond only to valid queries $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$, where $\mathsf{st}_{k,\mathsf{eid},2} = (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.

For all $i \in \mathcal{S} \cap \mathsf{hon}$ we have $(i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}}) \in \mathsf{Q}_{\mathsf{msg}}$. This means that R_i was returned during an $\mathcal{O}^{\mathsf{Sign}_2}$ query with respect to the same $(\mathcal{S}, m, \{\mathsf{cm}_\ell\}_{\ell\in\mathcal{S}})$. Thus, there exists $(i, R_i, \perp, \mathsf{cm}_i, c_i, z_i) \in \mathsf{Q}_{\mathsf{cm}}$ for all $i \in \mathcal{S} \cap \mathsf{hon}$ such that $R_i = g^{z_i} X_i^{-c_i}$.

For all $j \in \mathcal{S} \cap \mathsf{cor}$, we have that there exists $(j, R_j, \cdot, \mathsf{cm}_j, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$. Each corrupt R_j's hashes to a unique cm_j.

Thus, some $i' \in \mathcal{S}$ must have been the last value $(i', R_{i'}, \cdot, \mathsf{cm}_{i'}, \cdot, \cdot)$ added to Q_{cm}. Upon this occurring, we have that $(\mathsf{pk}, m, R, c, \Delta)$ was added to $\mathsf{Q}_{\mathsf{sig}}$ with $R = \prod_{\ell\in\mathcal{S}} R_\ell$ and with $c_i = \lambda_i c$ for all $i \in \mathcal{S} \cap \mathsf{hon}$.

Thus $R_k = g^{z_k} X_k^{-c\lambda_k}$ and the response is correctly distributed.

$\mathsf{ForgeDS}_k$ and $\mathsf{ForgeDS}_i$ cannot occur unless Game 1 also would abort. $\mathsf{FromInvalid}_k$ occurs if some honest R_i is computed in an $\mathcal{O}^{\mathsf{Sign}_2}$ query during Case 1 (Invalid Query). By the same argument as in the proof of Theorem 1, this occurs with maximum probability $\frac{q}{p}$.

$\underline{\mathsf{SIM}'\text{'s simulation of } \mathcal{O}^{\mathsf{Corrupt}}}$. SIM''s simulation of $\mathcal{O}^{\mathsf{Corrupt}}$ is perfect.

1. If there exists R_k, r_k such that $(k, R_k, r_k, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, then R_k and r_k were chosen during Case 1. This means that r_k is randomly distributed and $R_k = g^{r_k}$.
2. If there exists R_k, c_k, z_k such that $(k, R_k, \perp, \mathsf{cm}_k, c_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$, then R_k, c_k and z_k were chosen during Case 2. This means that $R_k \leftarrow g^{z_k} X_k^{-c\lambda_k}$, $c_k = c\lambda_k$ and thus:
$$R_k = g^{z_k - x_k c_k} = g^{r_k}$$
Further, r_k is randomly distributed because z_k is randomly chosen.
3. If there does not exist R_k such that $(k, R_k, \cdot, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, then by design $R_k = g^{r_k}$ for random r_k, and $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(k, R_k)$.

The probability that $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}$ occurs, i.e., $(\mathsf{pk}, m, R', \cdot, \cdot) \in \mathsf{Q}_{\mathsf{sig}}$, is bounded by $\frac{q}{p}$. This is because the aggregated R' is randomized by R_k (as discussed in the analysis of $\mathsf{H}_{\mathsf{sig}}$).

The probability that $\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}$ occurs is bounded by $\frac{q}{p}$. Indeed, the only way in which $(k, R_k, \perp, \mathsf{cm}_k, \perp, \perp) \in \mathsf{Q}_{\mathsf{cm}}$ is if the adversary queries H_{cm} on (k, R_k) and gets response cm_k. This happens if and only if there is a collision with some honest cm_k, which occurs with maximum probability $\frac{q}{p}$.

Output. When $\mathcal{A}$ terminates with output (m^*, σ^*), SIM' first checks that it is a valid forgery, by checking that $m^* \notin \mathsf{Q}_m$ and $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$. If either check fails, SIM' returns 0.

Otherwise, SIM' looks up the entry $(\mathsf{pk}, m^*, R^*, c^*, I) \in \mathsf{Q}_{\mathsf{sig}}$ corresponding to $\mathcal{A}$'s forgery. It sets $\mathsf{out} \leftarrow \sigma^*$. SIM' then outputs (I, out).

Difference between Game 2 and Game 1. Let W_2 be the event that $\mathcal{A}$ wins Game 2. The only additional advantage in the simulation by SIM' over Game 1 is given by bad events $\mathsf{ROCollision}$, $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}$, $\mathsf{FromInvalid}_k$, $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}$, $\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}$, $\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}$ (recalling that $\mathsf{ForgeDS}_k$ and $\mathsf{ForgeDS}_i$ cannot occur). As such,

$$\begin{aligned}|\Pr[W_1] - \Pr[W_2]| &= \Pr[\mathsf{ROCollision}] + \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}] + \Pr[\mathsf{FromInvalid}_k] \\ &\quad + \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}] + \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}] + \Pr[\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}] \\ &= \frac{q^2}{2p} + \frac{5q}{p}\end{aligned} \tag{15}$$

Reduction $\mathcal{B}'_2$ Against AOMDL. Finally, we construct our PPT reduction $\mathcal{B}'_2$ against the AOMDL assumption (Fig. 5). Specifically, we show how $\mathcal{B}'_2$ uses $\mathsf{Fork}^{\mathsf{SIM}}$ as a subroutine such that:

$$\begin{aligned}\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}'_2}(\kappa) \geq\ & \mathsf{accept}(\mathsf{Fork}^{\mathsf{SIM}'}(X)) - 2\Pr[\mathsf{ROCollision}] - 2\Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}] \\ & - 2\Pr[\mathsf{FromInvalid}_k] - 2\Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}] \\ & - 2\Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}] - 2\Pr[\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}]\end{aligned} \tag{16}$$

where X is the instance given by the AOMDL challenger.

Initialization. $\mathcal{B}'_2$ is initialized by the AOMDL challenger with input the group description $(\mathbb{G}, p, g)$ and $t+1$ AOMDL challenges $(Y_0, \ldots, Y_t) \in \mathbb{G}^{t+1}$. $\mathcal{B}'_2$ has access to a discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to t times. $\mathcal{B}'_2$ aims to output $(y_0, \ldots, y_t)$ such that $Y_i = g^{y_i}$ for all $0 \leq i \leq t$, with only t queries to its DL oracle.

To simulate $\mathcal{O}^{\mathsf{dl}}$ queries by SIM', $\mathcal{B}'_2$ initializes a table Q_{dl}, which it uses to cache $\mathcal{O}^{\mathsf{dl}}$ queries made by SIM' and responses from $\mathcal{O}^{\mathsf{dl}}$.

Execution. $\mathcal{B}'_2$ then runs the general forking algorithm $\mathsf{Fork}^{\mathsf{SIM}'}(X)$ as described in Figure 9, on the simulator algorithm SIM' and its instance X.

When SIM' queries $\mathcal{O}^{\mathsf{dl}}$, $\mathcal{B}'_2$ queries its own oracle $\mathcal{O}^{\mathsf{dl}}$. $\mathcal{B}'_2$ caches the request and response in Q_{dl}, and then returns the response to SIM'.

With non-negligible probability lower-bounded by Equation 6, $\mathsf{Fork}^{\mathsf{SIM}'}(X)$ will output the accepting answer $(I, \mathsf{out}, \mathsf{out}')$, such that:

$$\begin{gathered}h_I = c^*,\ h'_I = c^{**}, \\ \sigma^* \leftarrow \mathsf{out},\ \sigma^{**} \leftarrow \mathsf{out}' \\ (R^*, z^*) \leftarrow \sigma^*,\ (R^*, z^{**}) \leftarrow \sigma^{**}\end{gathered}$$

Extracting the Discrete Logarithm of Y_0. We show that $\mathcal{B}_2'$ can extract the discrete logarithm of Y_0 from $\mathcal{A}$'s two valid forgeries. We assume without loss of generality that $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (pk, R^*, m^*) on its forgery for each of its executions.

With overwhelming probability, $c^* \neq c^{**}$, and $\mathcal{B}_2'$ can solve for $y_0 = \frac{z^* - z^{**}}{c^* - c^{**}}$. If $\mathcal{B}_2'$ extracts y_0, then we use this to extract a full AOMDL solution as follows.

Extracting an AOMDL Solution. The reduction $\mathcal{B}_2'$ must now extract the remaining $y_1, \ldots, y_t$ such that $Y_i = g^{y_i}$. Assume without loss of generality that $\mathcal{A}$ makes t corruptions over the two iterations. (If not, SIM' can corrupt the remaining number at the end, by querying its DL oracle until it reaches t secret keys.) Recall that SIM' sets $X_i = Y_0 Y_1^i \cdots Y_t^{i^t}$, and made t $\mathcal{O}^{\mathsf{dl}}$ queries $g^{x_{i_1}}, \ldots, g^{x_{i_t}}$:

$$\begin{aligned} g^{x_{i_1}} &= Y_0 Y_1^{i_1} \cdots Y_t^{{i_1}^t} \\ &\vdots \\ g^{x_{i_t}} &= Y_0 Y_1^{i_t} \cdots Y_t^{{i_t}^t} \end{aligned}$$

Recall also that $\mathsf{pk} = Y_0$. This forms the following system of linear equations:

$$\begin{aligned} x &= y_0 \\ x_{i_1} &= y_0 + i_1 y_1 \cdots + {i_1}^t y_t \\ &\vdots \\ x_{i_t} &= y_0 + i_t y_1 \cdots + {i_t}^t y_t \end{aligned}$$

Equivalently,

$$\begin{pmatrix} x \\ x_{i_1} \\ \vdots \\ x_{i_t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & i_1 & \cdots & {i_1}^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_t & \cdots & {i_t}^t \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{pmatrix}$$

$\mathcal{B}_2'$ knows all of the values on the left-hand side. The matrix

$$V = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & i_1 & \cdots & {i_1}^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_t & \cdots & {i_t}^t \end{pmatrix}$$

is a Vandermonde matrix and is therefore invertible. Thus, $\mathcal{B}_2'$ can solve for $(y_0, y_1, \ldots, y_t)$ and win the $t+1$-aomdl game.

Finishing the Proof. Combining Equations 6, 13, 14, 15, and 16 gives Equation 4. This completes the proof. □

C Proof of Adaptive Security up to t Corruptions

We now prove our strongest result: that Sparkle+ is secure against t adaptive corruptions. In particular, if $t+1$ parties engage in signing, all but one of them could be malicious and the unforgeability of Sparkle+ would still hold. We prove this result under the EUF-CMA security of DS and under the AOMDL assumption (Fig. 5) in the AGM with random oracles. For simplicity, we assume that the signature scheme used, if it is group based, is implemented over a different group than the threshold signature.

Proof. (of Theorem 3.) We prove the theorem by a sequence of games.

Game 0. This is the adaptive unforgeability game as defined in Figure 7, instantiated with Sparkle+.

Let $\mathcal{A}$ be an algebraic PPT adversary attempting to break the adaptive unforgeability of Sparkle+ (Fig. 7) that makes up to q_H queries to H_{cm} and $\mathsf{H}_{\mathsf{sig}}$, and q_S queries to its signing oracles. $\mathcal{A}$ is allowed to make t adaptive corruptions. We assume, without loss of generality, that $\mathcal{A}$ makes a query to $\mathsf{H}_{\mathsf{sig}}$ on its forgery (pk, m^*, R^*). Then, let $q = q_H + q_S + 1$.

Let W_0 be the event that $\mathcal{A}$ wins Game 0. Then, the advantage of $\mathcal{A}$ is:

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1) = |\Pr[W_0]| \tag{17}$$

Here, $\mathsf{frac} = 1$ to allow $\mathcal{A}$ to corrupt a full $\mathsf{frac} \cdot t = t$ parties.

Game 1. The changes from Game 0 to Game 1 are the same as in the proof of Theorem 1.

Reduction $\mathcal{B}_1''$ to EUF-CMA Security of DS. We now define a reduction $\mathcal{B}_1''$ against the EUF-CMA security of DS that uses $\mathcal{A}$ as a subroutine. $\mathcal{B}_1''$ is identical to $\mathcal{B}_1'$ in the proof for Theorem 2. The only difference is the probability of Event_2 occurring. At the end of the game, there are $(n-t)$ non-corrupted parties. The probability that τ is one of them (i.e., that Event_2 does not occur) is $\frac{(n-t)}{n}$.

Difference between Game 0 and Game 1. If Event_1 does not occur, then Game 0 and Game 1 are identical. If Event_1 occurs, then the probability that $k = \tau$ is at least $\frac{1}{n}$. The probability that Event_2 does not occur, i.e., τ is not corrupted, is greater than $\frac{(n-t)}{n}$.

The additional advantage for $\mathcal{A}$ in Game 1, assuming that Event_1 occurs for participant τ, is upper-bounded by the advantage that $\mathcal{B}_1'$ wins its EUF-CMA game against DS, and the probability that Event_2 does not occur.

Let W_1 be the event that $\mathcal{A}$ wins Game 1. Then,

$$|\Pr[W_0] - \Pr[W_1]| \leq \frac{n^2}{(n-t)} \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1'',\mathsf{DS}}(\kappa) \tag{18}$$

Reduction $\mathcal{B}_2''$ against AOMDL. We now construct a PPT reduction $\mathcal{B}_2''$ against the AOMDL assumption (Fig. 5). Specifically, we show how $\mathcal{B}_2''$ uses the algebraic adaptive unforgeability adversary $\mathcal{A}$ as a subroutine such that:

$$\begin{aligned}\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}_2''}(\kappa) \geq \mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa,1) - \Pr[\mathsf{ROCollision}] - \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}] \\ - \Pr[\mathsf{FromInvalid}_k] - \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}] \\ - \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}] - \Pr[\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}]\end{aligned} \tag{19}$$

where X is the instance given by the AOMDL challenger.

Initialization. $\mathcal{B}_2''$ is initialized by the AOMDL challenger with input the group description $(\mathbb{G}, p, g)$ and $t+1$ AOMDL values $(Y_0, \ldots, Y_t) \in \mathbb{G}^{t+1}$. $\mathcal{B}_2''$ has access to a discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to t times. $\mathcal{B}_2''$ aims to output $(y_0, \ldots, y_t)$ such that $Y_i = g^{y_i}$ for all $0 \leq i \leq t$, with only t queries to its DL oracle.

Simulation. The simulation by $\mathcal{B}_2''$ is the same as that of SIM' in Game 2 for Theorem 2, with the exception that SIM' takes input values $\{h_1, \ldots, h_q\}$, whereas these should be sampled by $\mathcal{B}_2''$. As such, the difference between the simulation by $\mathcal{B}_2''$ and Game 1 is likewise upper-bounded by Equation 15.

When $\mathcal{A}$ terminates with output (m^*, σ^*), $\mathcal{B}_2''$ first checks that it is a valid forgery, by checking that $m^* \notin \mathsf{Q}_m$ and $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$. If either check fails, $\mathcal{B}_2''$ returns 0. Otherwise, $\mathcal{B}_2''$ looks up the entry $(\mathsf{pk}, m^*, R^*, c^*, \Delta^*) \in \mathsf{Q}_{\mathsf{sig}}$ corresponding to $\mathcal{A}$'s forgery.

The probability that $\mathcal{A}$ is successful and outputs a valid forgery but $\mathcal{B}_2''$ aborts is upper bounded by the difference between Game 0, Game 1, and the probability of additional bad events $\Pr[\mathsf{ROCollision}], \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}], \Pr[\mathsf{ForgeDS}_k]$, $\Pr[\mathsf{ForgeDS}_i], \Pr[\mathsf{FromInvalid}_k], \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{OneHon}}], \Pr[\mathsf{H}_{\mathsf{sig}}\mathsf{Guess}_{\mathsf{LastHonCor}}], \Pr[\mathsf{H}_{\mathsf{cm}}\mathsf{Guess}]$, as given in Equation 18 and Equation 19.

Extracting the Discrete Logarithm of Y_0. $\mathcal{A}$'s forgery verifies as:

$$R^* = g^{z^*}\,\mathsf{pk}^{-c^*} \tag{20}$$

where $c^* = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m^*, R^*)$. On the other hand, when $\mathcal{A}$ made its query $\mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m^*, R^*)$, it provided a representation of R^* in terms of all of the group elements it had seen so far, namely $(g, \mathsf{pk}, X_1, \ldots, X_n, \{R_{i,1}, \ldots, R_{i,n}\}_{i\in[q_S]})$, where $\{R_{i,k}\}_{i\in[q_S]}$ are the honest nonces returned by $\mathcal{O}^{\mathsf{Sign}_1}$ over the q_S signing queries that $\mathcal{A}$ makes. We assume without loss of generality that $\mathcal{A}$ completes every signing session. (Otherwise, $\mathcal{B}_2''$ can perform any unmet $\mathcal{O}^{\mathsf{Sign}_2}$ and $\mathcal{O}^{\mathsf{Sign}_3}$ queries itself.) Thus, $\mathcal{A}$ provided $(\gamma^*, \xi^*, \xi_1^*, \ldots, \xi_n^*, \{\rho_{i,1}^*, \ldots, \rho_{i,n}^*\}_{i\in[q_S]})$ such that:

$$R^* = g^{\gamma^*}\,\mathsf{pk}^{\xi^*}\,X_1^{\xi_1^*}\cdots X_n^{\xi_n^*}\prod_{i=1}^{q_S} R_{i,1}^{\rho_{i,1}^*}\cdots R_{i,n}^{\rho_{i,n}^*}$$

where $\rho^*_{i,k} = 0$ for k not in the signing set for query i. Each $R_{i,k}$ verifies as $R_{i,k} = g^{z_{i,k}} X_k^{-c_i \lambda_{i,k}}$, where $c_i = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m_i, R_i)$. Thus,

$$R^* = g^{\gamma^*} \mathsf{pk}^{\xi^*} X_1^{\xi^*_1} \cdots X_n^{\xi^*_n} \prod_{i=1}^{q_S} (g^{z_{i,1}} X_1^{-c_i \lambda_{i,1}})^{\rho^*_{i,1}} \cdots (g^{z_{i,n}} X_n^{-c_i \lambda_{i,n}})^{\rho^*_{i,n}}$$

Equating this with Eq. (20), we have:

$$g^{z^*} \mathsf{pk}^{-c^*} = g^{\gamma^*} \mathsf{pk}^{\xi^*} X_1^{\xi^*_1} \cdots X_n^{\xi^*_n} \prod_{i=1}^{q_S} (g^{z_{i,1}} X_1^{-c_i \lambda_{i,1}})^{\rho^*_{i,1}} \cdots (g^{z_{i,n}} X_n^{-c_i \lambda_{i,n}})^{\rho^*_{i,n}}$$

Rearranging, we have:

$$g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1}\rho^*_{i,1}} \cdots g^{-z_{i,n}\rho^*_{i,n}} = \mathsf{pk}^{c^*} \mathsf{pk}^{\xi^*} X_1^{\xi^*_1} \cdots X_n^{\xi^*_n} \prod_{i=1}^{q_S} (X_1^{-c_i \lambda_{i,1}})^{\rho^*_{i,1}} \cdots (X_n^{-c_i \lambda_{i,n}})^{\rho^*_{i,n}} \quad (21)$$

Let $\eta^* = z^* - \gamma^* - \sum_{i=1}^{q_S} (z_{i,1}\rho^*_{i,1} + \cdots + z_{i,n}\rho^*_{i,n})$ and $\zeta^*_k = \xi^*_k - \sum_{i=1}^{q_S} c_i \lambda_{i,k} \rho^*_{i,k}$ for all $k \in [n]$. Then Eq. (21) can be rewritten as:

$$g^{\eta^*} = \mathsf{pk}^{c^* + \xi^*} X_1^{\zeta^*_1} \cdots X_n^{\zeta^*_n} \quad (22)$$

Recall that $X_i = \mathsf{pk} Y_1^i Y_2^{i^2} \cdots Y_t^{i^t}$ for all $i \in [n]$. We have that $Y_i = \mathsf{pk}^{L'_{0,i}} \prod_{j \in \mathsf{cor}} g^{x_j L'_{j,i}}$ for all $i \in [t]$.[5] Thus,

$$X_k = \mathsf{pk} \prod_{i=1}^{t} \Big(\mathsf{pk}^{L'_{0,i}} \prod_{j \in \mathsf{cor}} g^{x_j L'_{j,i}} \Big)^{k^i}$$

Let $\mu^*_k = 1 + \sum_{i=1}^{t} L'_{0,i} k^i$ and $\nu^*_k = \sum_{i=1}^{t} \Big(\sum_{j \in \mathsf{cor}} x_j L'_{j,i} \Big) k^i$. Then X_k can be rewritten as:

$$X_k = \mathsf{pk}^{\mu^*_k} g^{\nu^*_k}$$

and Eq. (22) can be rewritten as:

$$g^{\eta^*} = \mathsf{pk}^{c^* + \xi^*} \prod_{k=1}^{n} (\mathsf{pk}^{\mu^*_k} g^{\nu^*_k})^{\zeta^*_k}$$

[5] There exists a unique degree-t polynomial $f(Z) = x + y_1 Z + \cdots + y_t Z^t$ such that $f(0) = x$ and $f(j) = x_j$ for all $j \in \mathsf{cor}$. It holds that $y_i = L'_{0,i} x + \sum_{j \in \mathsf{cor}} L'_{j,i} x_j$ for the appropriate Lagrange coefficients $L'_{0,i}, L'_{j,i}$. In general, for public key shares $X_i = \mathsf{pk} Y_1^i Y_2^{i^2} \cdots Y_t^{i^t}$ for $i \in [n]$, it holds that $Y_i = \mathsf{pk}^{L'_{0,i}} \prod_{j \in \mathsf{cor}} g^{x_j L'_{j,i}}$ for $i \in [t]$.

Rearranging, we have:

$$g^{\eta^* - \sum_{k=1}^{n} \nu_k^* \zeta_k^*} = \mathsf{pk}^{c^* + \xi^* + \sum_{k=1}^{n} \mu_k^* \zeta_k^*}$$

and

$$x = \frac{\eta^* - \sum_{k=1}^{n} \nu_k^* \zeta_k^*}{c^* + \xi^* + \sum_{k=1}^{n} \mu_k^* \zeta_k^*}$$

$\mathcal{A}$ fixed R^* and thus $\eta^*, \{\nu_i^*\}_{i \in [n]}, \xi^*, \{\mu_i^*\}_{i \in [n]}$ as it queried $\mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m^*, R^*)$ to receive random c^*. Thus, the denominator is nonzero with overwhelming probability and $\mathcal{B}_2''$ can solve for x.

Extracting an AOMDL Solution. As in the proof of Theorem 2, $\mathcal{B}_2''$ can solve for $(y_0, y_1, \ldots, y_t)$ and win the $t+1$-aomdl game.

Finishing the Proof. Combining Equations 17, 18, and 19 gives Equation 5. This completes the proof. □