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. Finally, we prove that Sparkle+ achieves *full* adaptive security, with a corruption threshold of t, under AOMDL in the algebraic group model (AGM) with random oracles. Importantly, we show adaptive security without requiring secure erasures. Ours is the first proof achieving full adaptive security without exponential tightness loss for *any* threshold Schnorr signature scheme.

Keywords: Threshold Signatures, Schnorr Signatures, Adaptive Security, Random Oracle Model, Algebraic Group Model

Table of Contents

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 [51, 3, 45, 27, 47, 7, 49, 55, 25]. Schnorr signatures admit an efficient and compact representation even in the multi-party setting, which makes them of particular interest for practical use [26, 18].

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 [21]. While there are generic methods for transforming a statically secure scheme into an adaptively secure one [23], 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 grows exponentially in the number of parties, and no adaptive guarantees can be made for larger n. Adaptive security without exponential tightness loss is challenging to achieve. A number of other techniques for proving adaptive security have been proposed, but similarly require undesirable tradeoffs. Prior methods include secure erasure of secret state [23], which is not easily enforced in practice, or heavyweight tools, such as non-committing encryption [43].

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 [18, 19], ideally supporting up to $n = 1024$ or more parties. Our techniques are likely of independent interest, as this paper introduces the first proof achieving full adaptive security without exponential tightness loss for *any* threshold Schnorr signature scheme.

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 DEFKLNS [30] and exhibited in polynomial time by BLLOR [15]. 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 ROS attacks. Combining concurrency and adaptivity in a multi-party, multi-round signing protocol is the main technical achievement of this work.

Scheme	Offline Signing Rounds	Online Signing Rounds	Static Security Assumptions	Proof of Adaptive Security
FROST/2/3 [45, 7, 55]	1	1	AOMDL	✗
Lindell22 [47]	-	3	Schnorr with aborts	✗
Classic S. [49]	-	3	Schnorr with aborts	Exponential to (n, t)
Zero S. [49]	-	3	Schnorr with aborts	Partial; cannot reveal state
SS01 [58]	-	5	Schnorr	✗
Sparkle+	1	2	DL	✓

Fig. 1. Threshold Schnorr signature schemes that are concurrently secure in either the standard model or the random oracle model (ROM). By partial adaptive security, we mean that the reduction can only partially simulate adaptive security, by revealing the signing keys, but not the signing session state maintained by an honest party.

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 are 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 [53]. 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 [51]. 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 [9] because the adversary only queries for the discrete logarithm of linear combinations of its challenge elements, which means the DL oracle runs in polynomial time. This, in conjunction with the verification running in polynomial time, makes AOMDL a falsifiable assumption [6, 41, 44].

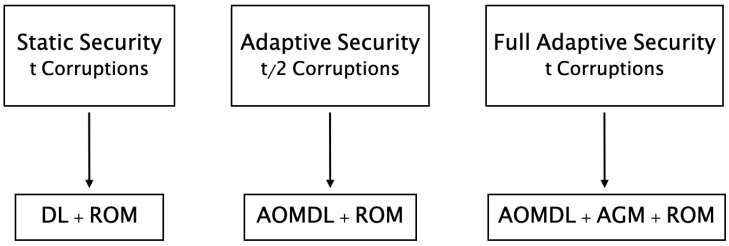

Fig. 2. Comparison of security models and assumptions for our threshold Schnorr signature scheme Sparkle+. The signing threshold is $t + 1$. DL is the discrete logarithm assumption, and AOMDL is the algebraic one-more discrete logarithm assumption. ROM is the random oracle model, and AGM is the algebraic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

In the case where the adversary can corrupt up to a full t 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) [34] 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. In the AGM, we are able to prove full adaptive security of Sparkle+, with corruption threshold t, under the AOMDL assumption and random oracles. Our 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+ is adaptively secure against 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, Drijvers, and Neven [17], the three-round MuSig scheme by Maxwell, Poelstra, Seurin, and Wuille [50], and the 2Schnorr scheme by Nicolosi, Krohn, Dodis, and Mazières [52]. However, MSDL and MuSig consider only the multi-signature setting (n-out-of-n), and 2Schnorr considers only the 2-out-of-2 setting. Note that when proving the security of multi-signatures, there is only one honest signer.

Stinson and Strobl [58] 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) [39] 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.

Komlo and Goldberg [45] present a two-round threshold Schnorr signature FROST. Unlike prior threshold Schnorr schemes in the literature [38], FROST is secure against a concurrent adversary and is not susceptible to ROS attacks [15]. FROST2, proposed by Crites, Komlo, and Maller [27] and refined by Bellare, Tessaro, and Zhu [14] and BCKMTZ [7], 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 RRJSS [55] and proven secure with a DKG by Chu, Gerhart, Ruffing, and Schröder [25]. (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) [7]. 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 [18, 19].

Lindell presents a three-round threshold Schnorr signature [47] with the goal of defining a scheme that is both secure against ROS attacks and secure in the random oracle model under the discrete logarithm assumption only. Security is modeled in the the universally composable framework (UC) [20] and therefore captures a concurrent adversary. However, the security proof assumes the adversary is static, and no claims are made regarding adaptive security. 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 [33], and hence is both significantly more efficient and a simpler design.

Concurrent to this work, Makriyannis [49] defines a commit-reveal threshold Schnorr signature scheme similar to Sparkle+, called Classic S., and proves security with respect to an idealized notion of threshold signatures by CGGMP21 [22]. They consider adaptive security but employ the guessing argument that incurs exponential tightness loss relative to the number of parties.

Adaptive Security of Threshold Signatures. While adaptive security for threshold schemes is a well-known topic in the literature, no proof of adaptive

Scheme	Sign					Combine	
	Performance			Bandwidth		Performance	
	rounds	exp	H	$\mathbb{G}$	$\mathbb{F}$	exp	H
FROST [45]	2	$t+2$	$t+1$	2	1	t	t
FROST2 [7]	2	3	2	2	1	1	1
FROST3 [55]	2	3	2	1	1	1	1
Classic S. [49]	3	1	1	1	2	0	0
Lindell22 [47]	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 [30, 15]. 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 [33], where $r = 8$ is the number of commitments for a Fischlin proof and the length of the zero vector is $b = 16$, such that $b \cdot r = 128$.

security without exponential tightness loss exists for a threshold Schnorr signature scheme that is secure against ROS attacks.

Generalized techniques for transforming statically secure threshold schemes into adaptively secure schemes have been defined in the literature [23, 43, 48]. However, these techniques either require the reduction to guess the corrupted parties ahead of time, require secure erasures, or introduce prohibitive performance overhead by using a robust distributed key generation mechanism (DKG) for nonce generation or heavyweight tools, such as Paillier encryption.

Almansa, Damgård, and Nielsen [2] present a threshold RSA scheme with proactive and adaptive security, but these results do not translate to the discrete logarithm setting. Libert, Joye, and Yung [46] propose a variant of the threshold BLS [16] 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 [4] 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.

Subsequent Work. A recent result by Das and Ren [29] shows 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 [49] introduces a threshold Schnorr signature scheme, Zero S., which relies on Pedersen commitments, Fischlin proofs of knowledge [33], 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 BLTWZ [5], 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 $\mathsf{H}(m)^x$ [24] (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 techniques used for simulation do not allow pre-processing: the message and signing set must be given in the first round. Furthermore, neither scheme is compatible with Schnorr signatures. To date, ours is the only proof of adaptive security for any threshold Schnorr signature scheme.

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 [46]. Alternative definitions of adaptive security in the UC setting have been proposed by CGGMP21 [22]. 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-out-of-n setting unless $\binom{n}{t}$ is small. Our fully adaptive t-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 [47] 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 [22], it is arguably overly strong in the sense that it necessitates certain design decisions, such as incorporating online-extractable zero-knowledge proofs. Indeed, [47] elected for Fischlin proofs (see Table 3). The only method to bias the nonces in both Sparkle+ and [47] is to abort; this is not the case in FROST [45]. However, all three works allow aborts, and therefore the distribution of the secret randomness cannot be considered uniform [31].

Differences to the Original Work. The authors of [5] 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. We provide a simple fix, by having parties sign the view of the commitments they receive from other parties, using any EUF-CMA secure signature scheme. 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 [53]. We see the same technique being used in other

schemes for the same purpose, namely in the blind threshold signature scheme Snowblind [28], and in [7] for "boosting" the security of FROST from TS-UF-3 to TS-UF-4. Indeed, this allows us to prove a stronger result: that adaptive security can be achieved with the message and signing set deferred to the second round. We are able to 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 $\mathbb{G}$.

Random Oracle Model [12]. 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 [34]. 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 $\vec{a} = (a_0, a_1, a_2, \ldots)$ such that $Z = g^{a_0} \prod Y_i^{a_i}$, where $Y_1, Y_2, \cdots \in \mathbb{G}$ 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 [57]). *Shamir secret sharing is an $(n, t+1)$-threshold secret sharing scheme consisting of algorithms* (IssueShares, Recover)*, defined as follows:*

- $\mathsf{IssueShares}(x, n, t+1) \to \{(1, x_1), \ldots, (n, x_n)\}$*: On input a secret x, number of participants n, and threshold $t+1$, perform the following. First, define a polynomial $f(Z) = x + a_1 + a_2 Z^2 + \cdots + a_t Z^t$ by sampling t coefficients at random: $a_1, \ldots, a_t \leftarrow_\$ \mathbb{Z}_p$. 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

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

3.2 Digital Signatures

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

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: On input a security parameter 1^κ, this algorithm outputs public parameters* par *(which are given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$*: This probabilistic algorithm outputs a public key* pk *and secret key* sk*.*
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*: On input a secret key* sk *and a message m, this algorithm outputs a signature σ.*

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

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

– $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \rightarrow 0/1$*: On input a public key* pk*, a message* m*, and a purported signature* σ*, this deterministic algorithm outputs* 1 *(accept) if* σ *is a valid signature on* m *under* 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* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is* correct *if for all security parameters* $\kappa \in \mathbb{N}$*, all key pairs* $(\mathsf{pk}, \mathsf{sk}) \in [\mathsf{KeyGen}()]$*, and all messages* m*, we have:*

$$\Pr[\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1$$

Definition 4 (EUF-CMA [42]). *Let the advantage of an adversary* $\mathcal{A}$ *playing the EUF-CMA game,* $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}$*, as defined in Figure 4, be as follows:*

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

A digital signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *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 [56]). *The Schnorr signature scheme consists of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, defined as follows:*

– $\mathsf{Setup}(1^\kappa) \rightarrow \mathsf{par}$*: On input a security parameter* 1^κ*, run* $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$ *and select a hash function* $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p$*. Output public parameters* $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H})$ *(which are given implicitly as input to all other algorithms).*

– $\mathsf{KeyGen}() \rightarrow (\mathsf{pk}, \mathsf{sk})$*: 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 of the discrete logarithm of the public key X, made non-interactive and bound to the message m by the Fiat-Shamir transform [32]. Schnorr signatures are secure under the discrete logarithm assumption in the random oracle model [53].

3.3 Assumptions

Assumption 1 (Discrete Logarithm Assumption (DL)) *Let the advantage of an adversary* $\mathcal{A}$ *playing the discrete logarithm game,* $\mathsf{Game}^{\mathsf{dl}}$*, as defined in Figure 5, be as follows:*

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

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

Assumption 2 (One-More Discrete Logarithm Assumption (OMDL)) [9] *Let the advantage of an adversary* $\mathcal{A}$ *playing the* $t+1$*-one-more discrete logarithm game,* $\mathsf{Game}^{t+1\text{-}\mathsf{omdl}}$*, as defined in Figure 5, be as follows:*

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

The one-more discrete logarithm assumption holds 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 [51]. We use it as an assumption to prove the adaptive security of Sparkle+, but it actually (implicitly) appears elsewhere in the literature where OMDL is used [11, 52, 13, 35].

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

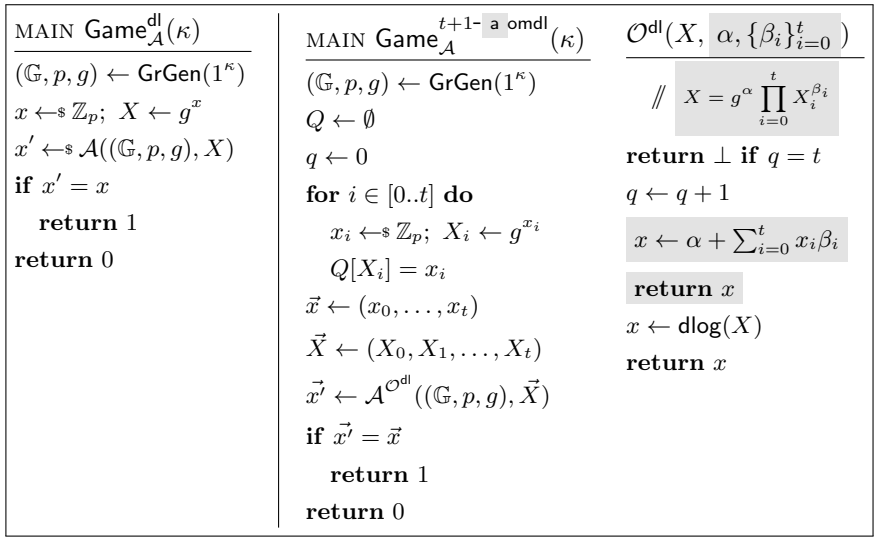

Fig. 5. The Discrete Logarithm (DL), One-More Discrete Logarithm (OMDL), and Algebraic One-More Discrete Logarithm (AOMDL) games. The differences between the OMDL and AOMDL games are highlighted in gray. The key distinction is that in the AOMDL game, the adversary can only query its discrete logarithm 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.

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) [51] *Let the advantage of an adversary $\mathcal{A}$ playing the $t+1$-algebraic one-more discrete logarithm game, $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}$, as defined in Figure 5, be as follows:*

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

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

4 Threshold Signature Schemes

We begin with the definition of a threshold signature scheme. We build upon prior definitions in the literature [37, 40, 36], but define an additional algorithm for combining signatures that is separate from the signing rounds. It may be performed by one of the signers or some external party. Our definition of threshold signatures models a generic 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 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.*
- $(\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3)$ *: 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* $\mathcal{S} \subseteq [n], |\mathcal{S}| = t+1$*, with respect to a message* m*, defined as follows:*

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

  where $\mathsf{pm}_{k,1}, \mathsf{pm}_{k,2}, \mathsf{pm}_{k,3}$ *are protocol messages sent by party* k*, and* $\mathsf{st}_{k,1}, \mathsf{st}_{k,2}, \mathsf{st}_{k,3}$ *is the state of signing 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 (Distributed key generation). Our definition assumes a centralized key generation algorithm KeyGen to generate the joint public key pk and key pairs $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}$. However, our scheme and proofs can be adapted to use a

distributed key generation protocol (DKG). We refer to [4] for a discussion on how to achieve adaptively secure DKGs.

Correctness. A threshold signature scheme is *correct* if for all security parameters $\kappa \in \mathbb{N}$, all $1 \leq t+1 \leq n$, all $\mathcal{S} \subseteq [n]$ such that $|\mathcal{S}| = t+1$, all messages m, and all $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) \in [\mathsf{KeyGen}(n, t+1)]$, the experiment defined by Eq. (2) returns σ such that $\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1$.

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. 6. 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 joint 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 joint 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 Sparkle+ does not rely on secure erasures: the state in each round is carried to all subsequent rounds.

Note also that 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

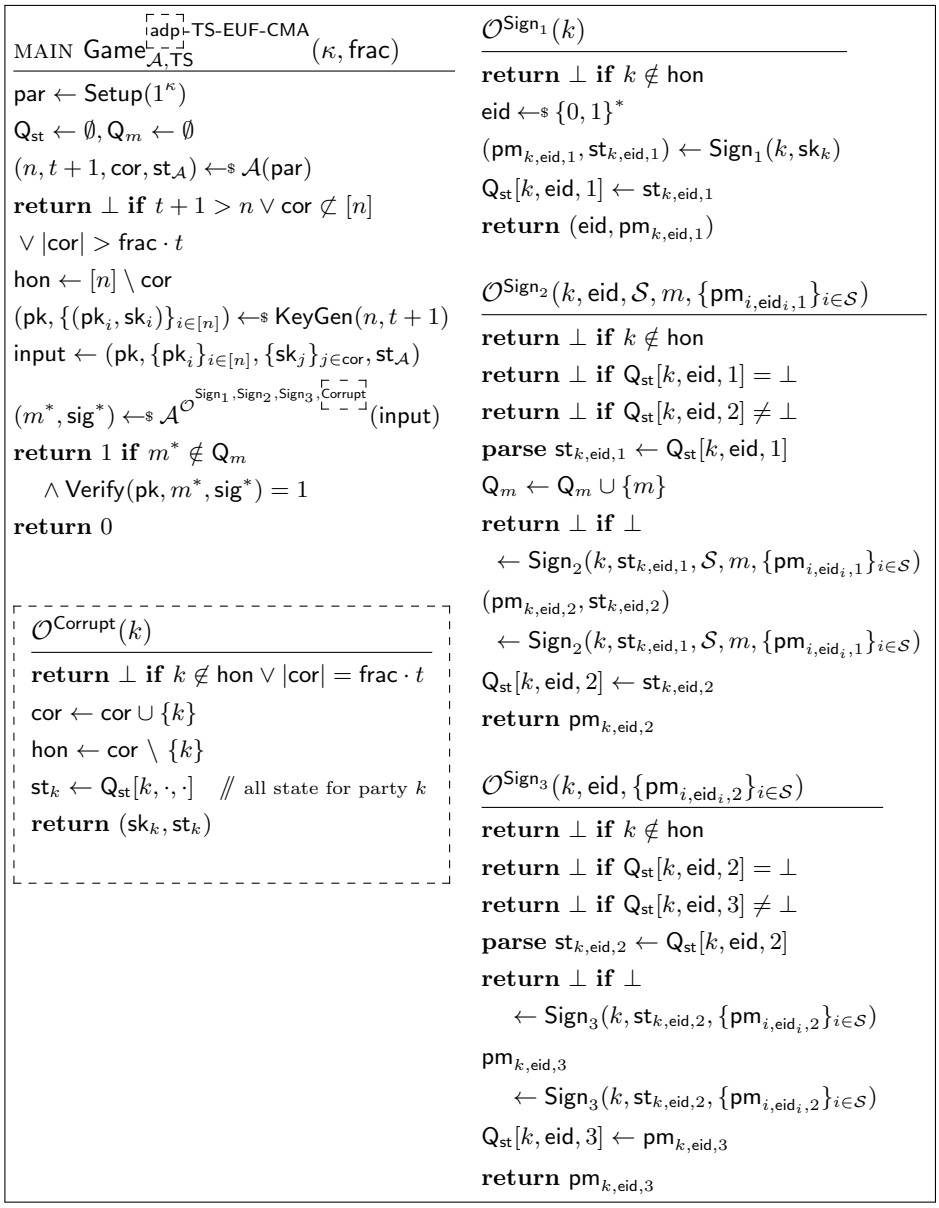

Fig. 6. Static and adaptive unforgeability games for a threshold signature scheme with one preprocessing round followed by two online signing rounds. The public parameters par are implicitly given as input to all algorithms, and $\mathsf{pm}_1, \mathsf{pm}_2, \mathsf{pm}_3$ represent protocol messages 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.

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 [30, 15].

Definition 7 (Static Security). *Let the advantage of an adversary $\mathcal{A}$ playing the static security game,* $\mathsf{Game}^{\mathsf{TS\text{-}EUF\text{-}CMA}}$*, as defined in Figure 6, be as follows:*

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

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 [46]. We provide a formal game-based definition, which is specified in Figure 6. 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* $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}$*, as defined in Figure 6, be as follows:*

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

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 t participants.*

5 Threshold Schnorr Signature Scheme Sparkle+

Sparkle+ is a simple three-round threshold Schnorr signature scheme (Fig. 7). 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 [4] for a discussion on how to achieve adaptively secure DKGs.

We assume that key generation operations are performed over a reliable, authenticated, and private network communication channel. For communication between participants at the time of signing, we assume a network model that is reliable. However, we do not assume consistency or authenticity of messages exchanged between participants. Thus, Sparkle+ additionally makes use of a standard EUF-CMA-secure single-party signature scheme DS to authenticate messages sent in the signing rounds.

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 $\mathsf{Combine}$ 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 [30, 15]. 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.

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.

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 joint 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.

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

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

// 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. 7. Sparkle+ threshold signature scheme with one round of preprocessing. 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. 6) 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.

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 $\{(R_j, \mathsf{cm}_j)\}_{j\in\mathcal{S}}$, 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 joint 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 $R \cdot \mathsf{pk}^c = g^z$.

Remark 2. Key generation outputs the set of public key shares $\{X_i\}_{i\in[n]}$, which allows identification of misbehaving signers (i.e., identifiable abort) by verifying each participant's partial signature share z_i with respect to X_i.

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 joint 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 [53]. We allow a static adversary to control up to t parties, but they must be declared at the beginning of the protocol (Fig. 6).

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. 6), 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* $\mathcal{O}^{\mathsf{Sign}}$*, and a PPT adversary* $\mathcal{B}_2$ *for the game* $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa)$ *(Fig. 5) such that*

$$\mathit{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa) \leq \sqrt{q \cdot \mathit{Adv}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa) + \frac{q^2+4q}{p} + 2n \cdot \mathit{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}_1,\mathsf{DS}}(\kappa)} \tag{3}$$

where n *is the number of signers,* t *is the maximum number of 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 1 in Section A.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the static unforgeability of Sparkle+ (Fig. 6). 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 joint public key $\mathsf{pk} \leftarrow \dot{X}$ and performs a standard simulation of Shamir secret sharing, resulting in public key shares $X_i = g^{x_i}$ 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 Sparkle+ signing without knowing the secret keys $\{x_k\}_{k\in\mathsf{hon}}$ of the honest parties as follows. $\mathcal{B}_2$ 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 [10]. 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 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 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.

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. 6), 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 $\mathcal{O}^{\mathsf{Sign}}$, 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*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, 1/2) \leq \sqrt{q\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}'_2}(\kappa) + \frac{q^2+6q}{p} + \frac{n^2}{(n-t/2)}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}'_1,\mathsf{DS}}(\kappa)} \tag{4}$$

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 Section B.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the $t/2$ adaptive unforgeability of Sparkle+ (Fig. 6). 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. For all $i \in [n]$, $\mathcal{B}'_2$ sets each $\mathsf{Sparkle}+$ public key share as $X_i \leftarrow Y_0 Y_1^i \cdots Y_t^{i^t}$. The joint 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. 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: $\mathsf{Sparkle}+$ 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 $\mathsf{Sparkle}+$ 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 *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. 6), 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* $\mathcal{O}^{\mathsf{Sign}}$*, 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*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, 1) \leq \mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}''_2}(\kappa) + \frac{q^2}{2p} + \frac{3q}{p} + \frac{n^2}{(n-t)}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{B}''_1,\mathsf{DS}}(\kappa) \quad (5)$$

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 Section C.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the full adaptive unforgeability of $\mathsf{Sparkle}+$ (Fig. 6). 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. 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 $k \in [n]$, $X_k = \mathsf{pk}Y_1^kY_2^{k^2}\cdots Y_t^{k^t}$, 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 i^{th} 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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] M. Bellare, W. Dai, and L. Li. "The Local Forking Lemma and Its Application to Deterministic Encryption". In: *ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11923. LNCS. Springer, 2019, pp. 607–636.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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`.

[23] 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.

[24] 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`.

[25] 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`.

[26] 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/`.

[27] 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`.

[28] E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. "Snowblind: A Threshold Blind Signature in Pairing-Free Groups". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14081. Lecture Notes in Computer

Science. Springer, 2023, pp. 710–742. DOI: 10.1007/978-3-031-38557-5_23. URL: https://doi.org/10.1007/978-3-031-38557-5_23.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] R. Gennaro and S. Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by D. Lie, M. Mannan, M. Backes, and X. Wang. ACM, 2018, pp. 1179–1194.

[37] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Robust Threshold DSS Signatures". In: *Inf. Comput.* 164.1 (2001), pp. 54–84.

[38] 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.

[39] 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.

[40] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. "Robust and Efficient Sharing of RSA Functions". In: *J. Cryptol.* 20.3 (2007), p. 393.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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/.

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

[54] 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.

[55] 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.

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

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

[58] 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.

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** $I \neq I'$
return $(I, \mathsf{out}, \mathsf{out}')$

Fig. 8. The forking algorithm $\mathsf{Fork}^{\mathcal{C}}(x)$ associated to an algorithm $\mathcal{C}$ and input x.

A Proof of Static Security

We now formally prove Theorem 1. Since our security reduction makes use of rewinding, we review the general forking lemma by Bellare and Neven [10], which is a formalization of the forking lemma introduced by Pointcheval and Stern [54].

Lemma 1 (General Forking Lemma [10]). *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 $I \geq 1$ 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 input X and proceeds as in Figure 8. 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 6, instantiated with $\mathsf{Sparkle}+$.

Let $\mathcal{A}$ be a PPT adversary attempting to break the static unforgeability of $\mathsf{Sparkle}+$ (Fig. 6) 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 static corruptions. Without loss of generality, we assume $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ 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 simply

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

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. The only difference between Game 0 and Game 1 is the following abort event.

In Game 1, the environment initializes a table $\mathsf{Q}_{\mathsf{DS}} \leftarrow \emptyset$. 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. 6) and Sign_2 (Fig. 7). 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 Q_{DS}. Note that with high probability, each cm_k will only be added to Q_{DS} once.

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 $\mathcal{O}^{\mathsf{Sign}_3}$ and Sign_3. If they pass, then, in particular, $\mathsf{st}_{k,\mathsf{eid},2} = (R_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \mathsf{cm}_k, r_k, \mathsf{sk}_k)$, 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 Equation 9 holds, the environment aborts. We refer to this abort event as Event_1.

$$\forall i \in \mathsf{hon} \cap \mathcal{S}, \mathsf{DS.Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) = 1, \text{but } \exists i \text{ s.t. } \mathsf{msg}_i \notin \mathsf{Q}_{\mathsf{DS}} \tag{9}$$

Reduction to EUF-CMA of DS. We now define a reduction $\mathcal{B}_1$ against the EUF-CMA security of DS (Fig. 4) 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 reduction $\mathcal{B}_1^{\mathcal{O}^{\mathsf{Sign}}}(\mathsf{par}_{\mathsf{DS}}, \hat{X}^\dagger)$ accepts as input public parameters $\mathsf{par}_{\mathsf{DS}}$ and a challenge public key $\hat{X}^\dagger$; its goal is to output a valid forgery $(\mathsf{msg}^*, \hat{\sigma}^*)$ under $\hat{X}^\dagger$, such that $\mathcal{B}_1$ has not queried msg^* to its signing oracle $\mathcal{O}^{\mathsf{Sign}}$.

Setup. $\mathcal{B}_1$ first runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\kappa)$ and sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{par}_{\mathsf{DS}})$. It then initializes the tables $Q, Q_m, \mathsf{Q}_{\mathsf{DS}} \leftarrow \emptyset$. It runs $(n, t+1, \mathsf{cor}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{par})$.

$\mathcal{B}_1$ runs key generation for $\mathsf{Sparkle}+$ honestly: $(\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) \leftarrow\!\!\$\ \mathsf{KeyGen}(n, t{+}1)$. $\mathcal{B}_1$ 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 then 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 $(k, \mathsf{eid}, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, $\mathcal{B}_1$ carries out the appropriate checks, updates $Q_m \leftarrow Q_m \cup \{m\}$, 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 $\mathcal{B}_1$ responds honestly as per Game 0.
  2. If $k = \tau$, then $\mathcal{B}_1$ queries its own signing oracle $\mathcal{O}^{\mathsf{Sign}}(\mathsf{msg}_\tau)$ to obtain $\hat{\sigma}_\tau$. It otherwise follows the protocol honestly.

  Finally, $\mathcal{B}_1$ updates $\mathsf{Q}_{\mathsf{DS}} \leftarrow \mathsf{Q}_{\mathsf{DS}} \cup \{\mathsf{msg}_k\}$.
- $\mathcal{O}^{\mathsf{Sign}_3}$: When $\mathcal{A}$ queries on $(k, \mathsf{eid}, \{(R_i, \hat{\sigma}_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}_1$ carries out the appropriate checks. It then checks if Equation 9 holds. If not, B_1 simply follows the protocol honestly.

  Otherwise, if Equation 9 holds, then Event_1 has occurred. If $\tau \in \mathsf{hon} \cap \mathcal{S}$ and $\mathsf{msg}_\tau \notin \mathsf{Q}_{\mathsf{DS}}$ (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 Event_1 does not occur, then $\mathcal{B}_1$ outputs $\perp$ as its output 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 $\mathcal{O}^{\mathsf{Sign}_1}$ honestly, and so its simulation is perfect. It responds to $\mathcal{O}^{\mathsf{Sign}_2}$ and $\mathcal{O}^{\mathsf{Sign}_3}$ honestly, with the exception of participant τ. However, because it queries its own EUF-CMA signing oracle $\mathcal{O}^{\mathsf{Sign}}$ on msg_τ to obtain $\hat{\sigma}_\tau$, and $\hat{X}_i = \hat{X}^\dagger$, then $\mathcal{A}$'s ability to distinguish the simulation for participant τ is the same as $\mathcal{B}_1$'s ability to distinguish the EUF-CMA simulation of DS.

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 $\mathcal{A}$ forges for participant τ is at least $\frac{1}{n}$. The additional advantage to $\mathcal{A}$ 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 for DS.

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

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

Game 2. We next define a simulating algorithm SIM, which simulates the static unforgeability game $\mathsf{Game}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SIM}}(\kappa, \mathsf{frac} = 1)$, as follows.

Setup. SIM accepts as input an instance X, which is a tuple consisting of the group description $\mathcal{G} = (\mathbb{G}, p, g)$ 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.

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 $\mathsf{BadEvent}_1$.

Otherwise, SIM initializes the following values:

- Parameters for DS by performing $\mathsf{par}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$, and then sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}, \mathsf{par}_{\mathsf{DS}})$.
- The sets $\mathsf{Q}_{\mathsf{cm}} \leftarrow \emptyset, \mathsf{Q}_{\mathsf{sig}} \leftarrow \emptyset$ of $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries and their responses, respectively.
- The set $\mathsf{Q}_{\mathsf{DS}} \leftarrow \emptyset$ 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 participant state for all signing sessions initiated by $\mathcal{A}$.
- The counter $\mathsf{ctr}_h \leftarrow 1$.

SIM may program the random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$.

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 $\mathsf{cor} \leftarrow \{j\}, |\mathsf{cor}| \leq t$, 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. 7) using the 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. (See Section 3 for notation.) 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 $f(0) = \dot{x}$ 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) $0 \cup \mathsf{cor}$.
   (c) For all $1 \leq i \leq t$, SIM computes

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

   where $L'_{j,i}$ is the i^{th} coefficient of $L'_j(Z) = L'_{j,0} + L'_{j,1} Z + \cdots + L'_{j,t} Z^t$.
   (d) For all $1 \leq i \leq n$, SIM computes

$$X_i = \mathsf{pk} Y_1^i Y_2^{i^2} \cdots Y_t^{i^t}$$

   which is implicitly equal to $g^{f(i)}$.

4. Set $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ 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 $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_1}, \mathcal{O}^{\mathsf{Sign}_2}, \mathcal{O}^{\mathsf{Sign}_3}}(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_j\}_{j\in\mathsf{cor}}, \mathsf{st}_{\mathcal{A}})$.

Simulating Random Oracle Queries. When $\mathcal{A}$ queries H_{cm} or $\mathsf{H}_{\mathsf{sig}}$, 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 $(i, R_i, \mathsf{cm}, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, and if so, it returns cm.

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

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

Else, SIM sets $c \leftarrow h_{\mathsf{ctr}_h}$, and sets $\Delta = \mathsf{ctr}_h$. It appends (X, m, R, c, Δ) to $\mathsf{Q}_{\mathsf{sig}}$. It then increments $\mathsf{ctr}_h \leftarrow \mathsf{ctr}_h + 1$. Finally, SIM 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 $\mathcal{O}^{\mathsf{Sign}}$ for $k \in \mathsf{hon}$, SIM first performs the checks in $\mathcal{O}^{\mathsf{Sign}_1}$ (Fig. 6) and Sign_1 (Fig. 7). If they pass, then SIM does the following:

1. Samples $\mathsf{eid} \leftarrow\!\!\$\ \{0,1\}^*$.
2. It sets $\mathsf{cm}_k \leftarrow h_{\mathsf{ctr}_h}$, and increments $\mathsf{ctr}_h \leftarrow \mathsf{ctr}_h + 1$.
3. It 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 $\mathsf{msg}_i = (k, \mathsf{eid}, R_i, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ 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. 6) and Sign_2 (Fig. 7). 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 $(k, \cdot, \mathsf{cm}_k, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$. If the check holds, SIM retrieves $(k, R_k, \mathsf{cm}_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$, and sets R_k accordingly. Then SIM: adds $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ to Q_{DS}; signs msg_k as $\hat{\sigma}_k \leftarrow \mathsf{Sign}(\hat{x}_k, \mathsf{msg}_k)$; 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. Else if not, then, for each commitment cm_i, SIM checks if it has a defined entry $(i, R_i, \mathsf{cm}_i, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$.
   (a) If there exists some $\mathsf{cm}_i \in \mathcal{PM}_1$ 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 $\mathsf{cm}_i \in \mathcal{PM}_1$ such that $i \in \mathcal{S} \cap \mathsf{cor}$ and cm_i does not have a corresponding R_i 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 $R_k \leftarrow\!\!\$\ \mathbb{G}$.
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 $\mathsf{Q}_{\mathsf{DS}} \leftarrow \mathsf{Q}_{\mathsf{DS}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature on msg_k.
5. Set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
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 $c \leftarrow h_{\mathsf{ctr}_h}$, and $\Delta = \mathsf{ctr}_h$. It then increments $\mathsf{ctr}_h \leftarrow \mathsf{ctr}_h + 1$.
2. Then, for all $j \in \mathcal{S} \cap \mathsf{hon}$, SIM samples $z_j \leftarrow\!\!\$\ \mathbb{Z}_p$.
3. For all $j \in \mathcal{S} \cap \mathsf{hon}$, it computes $R_j \leftarrow g^{z_j} X_j^{-c\lambda_j}$, where λ_j is the Lagrange coefficient for party j in the set $\mathcal{S}$.
4. Otherwise, SIM programs $\mathsf{cm}_j \leftarrow \mathsf{H}_{\mathsf{cm}}(j, R_j)$. Specifically SIM adds the entry $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(j, R_j, \mathsf{cm}_j, z_j)\}$ for all $j \in \mathcal{S} \cap \mathsf{hon}$.
5. Next, SIM obtains each corrupted party's commitment by parsing $(j, R_j, \mathsf{cm}_j, \perp) \leftarrow \mathsf{Q}_{\mathsf{cm}}$, for each $j \in \mathcal{S} \cap \mathsf{cor}$.
6. SIM then computes $R = \prod_{i\in\mathcal{S}} R_i$.
7. If $(\mathsf{pk}, m, R, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{sig}}$, then SIM aborts. We refer to this bad event as $\mathsf{BadEvent}_2$.
8. Otherwise, SIM programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. Specifically, SIM adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R, \Delta)\}$.
9. SIM follows the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k.
10. It adds $\mathsf{Q}_{\mathsf{DS}} \leftarrow \mathsf{Q}_{\mathsf{DS}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature on msg_k.
11. Set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, \perp, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
12. Updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$
13. It then 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 $\mathcal{S}$ produces a partial signature on the message m. 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. 6) and Sign_3 (Fig. 7). 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 $\mathsf{Verify}(\hat{X}_i, \mathsf{msg}_i, \hat{\sigma}_i) = 1$ 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 $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}) \in \mathsf{Q}_{\mathsf{DS}}$ and aborts if no such query exists. We refer to this bad event as $\mathsf{BadEvent}_3$.

4. Check that there exists cm_i such that $\mathsf{msg}_i = (i, \mathsf{cm}_i, R_i, \mathcal{S}, m, \{\mathsf{cm}_j\}_{j\in\mathcal{S}}) \in \mathsf{Q}_{\mathsf{DS}}$ for all $i \in \mathcal{S} \cap \mathsf{hon}$ and abort if not. We refer to this bad event as $\mathsf{BadEvent}_4$.
5. Else, SIM retrieves $(k, R_k, \mathsf{cm}_k, z_k) \leftarrow \mathsf{Q}_{\mathsf{cm}}$ and aborts if $z_k = \perp$. We refer to this bad event as $\mathsf{BadEvent}_5$.
6. Updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 3] \leftarrow (z_k)$
7. 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 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. The R_k are 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.

$\mathsf{BadEvent}_2$ occurs with maximum probability $\frac{q}{p}$. Indeed, SIM checks to see if $(\mathsf{pk}, m, R, \cdot) \in \mathsf{Q}_{\mathsf{sig}}$. 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 $(k, R_k, \mathsf{cm}_k, z_k) \leftarrow \mathsf{Q}_{\mathsf{cm}}$, then $R_k = g^{z_k} X_k^{-c\lambda_k}$ where $c = \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. Note that the aggregate R is computed consistently if all honest R_j's are computed in the same $\mathcal{O}^{\mathsf{Sign}_2}$ query during Case 2 (Valid Query), guaranteed by $\mathsf{BadEvent}_4$, and all corrupt R_j's hash to a unique cm_j. If two (j, R_j) and (j', R'_j)'s hash to the same cm, then $\mathsf{BadEvent}_1$ has occurred.

$\mathsf{BadEvent}_3$ and $\mathsf{BadEvent}_4$ cannot occur unless Game 1 also would abort.

$\mathsf{BadEvent}_5$ occurs if some honest R_j is computed in a $\mathcal{O}^{\mathsf{Sign}_2}$ query during Case 1 (Invalid Query). If there exists some $\mathsf{cm}_i \in \mathcal{PM}_1$ such that $i \in \mathcal{S} \cap \mathsf{hon}$ and cm_i has a corresponding R_i defined in Q_{cm}, then $\mathsf{BadEvent}_3$ or $\mathsf{BadEvent}_4$ will occur because $\mathsf{cm}_k \in \mathcal{PM}_1$ is not yet programmed but some other honest $\mathsf{cm}_j \in \mathcal{PM}_1$ has been programmed. Thus, cm_j must have been programmed during a query involving a different set $\mathcal{PM}'_1 \neq \mathcal{PM}_1$ with $\mathsf{cm}_k \notin \mathcal{PM}'_1$. The probability that there exists some $\mathsf{cm}_i \in \mathcal{PM}_1$ such that $i \in \mathcal{S} \cap \mathsf{cor}$ and cm_i does not have a corresponding R_i defined in Q_{cm}, but an adversary later finds R_i such that $\mathsf{H}_{\mathsf{cm}}(i, R_i) = \mathsf{cm}_i$ is bounded by $\frac{q}{p}$.

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

1. $\underline{\mathsf{BadEvent}_1}$: SIM aborts due to $\mathsf{BadEvent}_1$ with probability $\frac{q^2}{2p}$, which occurs when it receives any random oracle outputs $\{h_1, \ldots, h_q\}$ that collide.
2. $\underline{\mathsf{BadEvent}_2}$: SIM aborts due to $\mathsf{BadEvent}_2$ with probability q/p.
3. $\underline{\mathsf{BadEvent}_3}$: When SIM aborts due to $\mathsf{BadEvent}_3$, then Game 1 also aborts.
4. $\underline{\mathsf{BadEvent}_4}$: When SIM aborts due to $\mathsf{BadEvent}_4$, then Game 1 also aborts.
5. $\underline{\mathsf{BadEvent}_5}$: SIM aborts due to $\mathsf{BadEvent}_5$ 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 outputs $\perp$.

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 $\mathsf{BadEvent_1}$, $\mathsf{BadEvent_2}$, and $\mathsf{BadEvent_5}$. As such,

$$\begin{aligned} |\Pr[W_2] - \Pr[W_1]| &= \Pr[\mathsf{BadEvent_1}] + \Pr[\mathsf{BadEvent_2}] + \Pr[\mathsf{BadEvent_5}] \\ &= \frac{q^2}{2p} + \frac{2q}{p} \end{aligned} \tag{11}$$

The Reduction $\mathcal{B}_2$. We now construct a PPT reduction $\mathcal{B}_2$ against the DL assumption (Fig. 5). Specifically, we show how $\mathcal{B}_2$ uses the static unforgeability adversary $\mathcal{A}$ playing against SIM as defined in Game 2 as a subroutine such that

$$\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}_2}(\kappa) \geq \mathsf{accept}(\mathsf{Fork}^{\mathsf{SIM}}(X)) - 2\Pr[\mathsf{BadEvent_1}] - \ldots, -2\Pr[\mathsf{BadEvent_5}] \tag{12}$$

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 $\mathcal{G} = (\mathbb{G}, p, g)$ 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 8, on the simulating algorithm SIM and its instance 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 6, instantiated with $\mathsf{Sparkle}+$, with input $\mathsf{frac} = 1/2$.

Let $\mathcal{A}$ be a PPT adversary attempting to break the adaptive unforgeability of $\mathsf{Sparkle}+$ (Fig. 6) 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 $t/2$ adaptive corruptions. Without loss of generality, we assume $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ 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 simply

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

Game 1. Game 1 is identical to Game 1 in the proof for Theorem 1.

Reduction to EUF-CMA 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 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 $\mathcal{B}'_1$ responds honestly as per Game 0.
2. If $k = \tau$, then $\mathcal{B}'_1$ aborts. We refer to this abort event as Event_2.

Analysis of $\mathcal{B}'_1$'s Simulation. The only difference between the simulation of $\mathcal{B}_1$ given in Game 1 for Theorem 1 and $\mathcal{B}'_1$ is the occurrence of Event_2. We bound the probability of this event occurring as follows.

We calculate the probability of not choosing party τ when selecting $t/2$ values without replacement from a set of size n as follows. The total number of ways to choose $t/2$ values from a set of size n is $\binom{n}{t/2}$. To calculate the number of ways to choose $t/2$ values without selecting τ, we consider the set to be of size $n-1$ because we are excluding the value τ. The total number of ways to choose $t/2$ values from a set of size $n-1$ is $\binom{n-1}{t/2}$. The probability of not choosing τ

(i.e., of not aborting) is thus

$$\frac{\binom{n-1}{t/2}}{\binom{n}{t/2}} = \frac{(n-1)!}{(t/2)!(n-1-t/2)!}\frac{(t/2)!(n-t/2)!}{n!} = \frac{(n-t/2)}{n}$$

Difference between Game 0 and Game 1. If Event_2 does not occur, then Game 0 and Game 1 are identical. If Event_2 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 $\frac{(n-t/2)}{n}$.

The additional advantage to $\mathcal{A}$ 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,

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

Game 2. We next define a simulating algorithm SIM', which simulates the adaptive unforgeability game $\mathsf{Game}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathsf{SIM}',\mathsf{Sparkle}+}(\kappa, \mathsf{frac} = 1/2)$. The simulation by SIM' is similar to that of SIM in Game 2 for Theorem 1, with the exception of the instance that SIM' accepts as input, how key generation is performed, what information SIM' stores in Q_{cm}, and how SIM' additionally simulates $\mathcal{O}^{\mathsf{Corrupt}}$.

Additionally, SIM' is given an oracle $\mathcal{O}^{\mathsf{sdl}}$, which it can query on elements $X \in \mathbb{G}$ together with an algebraic representation, and receive discrete logarithm solutions $x \in \mathbb{Z}_p$. 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 $\mathcal{G} = (\mathbb{G}, p, g)$ 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 $\mathsf{BadEvent}_1$.

Otherwise, SIM' initializes the following values:

- Parameters for DS by performing $\mathsf{par}_{\mathsf{DS}} \leftarrow \mathsf{DS.Setup}(1^\kappa)$, and then sets $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}, \mathsf{par}_{\mathsf{DS}})$.
- The sets $\mathsf{Q}_{\mathsf{cm}} \leftarrow \emptyset, \mathsf{Q}_{\mathsf{sig}} \leftarrow \emptyset$ of $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries and their responses, respectively.
- The set $\mathsf{Q}_{\mathsf{DS}} \leftarrow \emptyset$ 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 participant state for all signing sessions initiated by $\mathcal{A}$.
- The counter $\mathsf{ctr}_h \leftarrow 1$.

SIM' likewise may program the random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$.

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 initially corrupt $\mathsf{cor} \leftarrow \{j\}, |\mathsf{cor}| \leq t/2$ (although it may corrupt additional parties of its choosing later). 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. 7) 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 $1 \leq i \leq n$, SIM' sets the signing public key share as
$$X_i = Y_0 Y_1^i \cdots Y_t^{i^t}$$
which is implicitly equal to $g^{f(i)}$.
3. The threshold public key is $\mathsf{pk} = g^{f(0)} = Y_0$ with corresponding secret key $\mathsf{sk} = y_0$.
4. SIM' obtains the initial corrupt secret key shares by querying $x_j = f(j) \leftarrow \mathcal{O}^{\mathsf{sdl}}(X_j)$ (with representation $(1, j, \ldots, j^t)$) for all $j \in \mathsf{cor}$.
5. Set $\mathsf{pk}_i \leftarrow (X_i, \hat{X}_i)$ 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 $(i, R_i, \cdot, \mathsf{cm}, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$, and if so, it returns cm.

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

$\underline{\mathsf{H}_{\mathsf{sig}}}$: 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 $(k, \cdot, \cdot, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$. If the check holds, SIM' retrieves $(k, R_k, \cdot, \mathsf{cm}_k, c_k, z_k) \in \mathsf{Q}_{\mathsf{cm}}$, and sets R_k accordingly. Then SIM: adds $\mathsf{msg}_k = (k, \mathsf{cm}_k, R_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ to Q_{DS}; signs msg_k as $\hat{\sigma}_k \leftarrow \mathsf{Sign}(\hat{x}_k, \mathsf{msg}_k)$; 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. Else if not, then, for each commitment cm_i, SIM' checks if it has a defined entry $(i, R_i, \cdot, \mathsf{cm}_i, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$.
   (a) If there exists some $\mathsf{cm}_i \in \mathcal{PM}_1$ 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 $\mathsf{cm}_i \in \mathcal{PM}_1$ such that $i \in \mathcal{S} \cap \mathsf{cor}$ and cm_i does not have a corresponding R_i 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 $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(k, R_k, r_k, \mathsf{cm}_k, \bot, \bot)\}$.
3. Follow the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k.
4. Add $\mathsf{Q}_{\mathsf{DS}} \leftarrow \mathsf{Q}_{\mathsf{DS}} \cup \{\mathsf{msg}_k\}$, 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 $c \leftarrow h_{\mathsf{ctr}_h}$, and $\Delta = \mathsf{ctr}_h$. It then increments $\mathsf{ctr}_h \leftarrow \mathsf{ctr}_h + 1$.
2. Then, for all $j \in \mathcal{S} \cap \mathsf{hon}$, SIM' samples $z_j \leftarrow_\$ \mathbb{Z}_p$.
3. For all $j \in \mathcal{S} \cap \mathsf{hon}$, it computes $R_j \leftarrow g^{z_j} X_j^{-c\lambda_j}$, where λ_j is the Lagrange coefficient for party j in the set $\mathcal{S}$.
4. SIM' programs $\mathsf{cm}_j \leftarrow \mathsf{H}_{\mathsf{cm}}(j, R_j)$. Specifically SIM' adds the entry $\mathsf{Q}_{\mathsf{cm}} \leftarrow \mathsf{Q}_{\mathsf{cm}} \cup \{(j, R_j, \bot, \mathsf{cm}_j, c\lambda_j, z_j)\}$ for all $j \in \mathcal{S} \cap \mathsf{hon}$.
5. Next, SIM' obtains each corrupted party's commitment by parsing $(j, R_j, \bot, \mathsf{cm}_j, \bot, \bot) \leftarrow \mathsf{Q}_{\mathsf{cm}}$, for each $j \in \mathcal{S} \cap \mathsf{cor}$.
6. SIM' then computes $R = \prod_{i\in\mathcal{S}} R_i$.
7. If $(\mathsf{pk}, m, R, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{sig}}$, then SIM' aborts. We refer to this bad event as $\mathsf{BadEvent}_2$.
8. Otherwise, SIM' programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\mathsf{pk}, m, R)$. Specifically, SIM' adds $\mathsf{Q}_{\mathsf{sig}} \leftarrow \mathsf{Q}_{\mathsf{sig}} \cup \{(\mathsf{pk}, m, R, \Delta)\}$.
9. SIM' follows the protocol honestly to produce a signature $\hat{\sigma}_k$ on msg_k.
10. It adds $\mathsf{Q}_{\mathsf{DS}} \leftarrow \mathsf{Q}_{\mathsf{DS}} \cup \{\mathsf{msg}_k\}$, indicating that honest party k output a signature on msg_k.
11. Updates $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \hat{\mathsf{sk}}_k)$.
12. Updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \leftarrow \mathsf{st}_{k,\mathsf{eid},2}$.
13. It then returns $(R_k, \hat{\sigma}_k)$.

Round 3 ($\mathcal{O}^{\mathsf{Sign}_3}(k, \mathsf{eid}, \{(R_i, \hat{\sigma_i})\}_{i\in\mathcal{S}})$): SIM' simulates $\mathcal{A}$'s queries to $\mathcal{O}^{\mathsf{Sign}_3}$ identically to SIM as in the proof for Theorem 1, with exception that SIM' retrieves $(k, R_k, \cdot, \mathsf{cm}_k, c_k, z_k) \leftarrow \mathsf{Q}_{\mathsf{cm}}$ (instead of $(k, R_k, \mathsf{cm}_k, z_k)$).

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 $\bot$ if not. Otherwise, SIM' queries its DL oracle $\mathcal{O}^{\mathsf{sdl}}$ 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:

– 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. Update $\hat{\mathsf{sk}}_k$ to sk_k and set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \mathsf{sk}_k)$ and $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.
– 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 r_k as

$$r_k = z_k - c_k x_k$$

Update $\hat{\mathsf{sk}}_k$ to sk_k and set $\mathsf{st}_{k,\mathsf{eid},2} \leftarrow (R_k, r_k, \mathsf{cm}_k, \hat{\sigma}_k, \mathcal{S}, m, \{\mathsf{cm}_i\}_{i\in\mathcal{S}}, \mathsf{sk}_k)$ and $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.
– If there does not exist R_k such that $(k, R_k, \cdot, \mathsf{cm}_k, \cdot, \cdot) \in \mathsf{Q}_{\mathsf{cm}}$ then 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 $(k, R_k, r_k, \mathsf{cm}_k, c_k, z_k)$ to Q_{cm}. Update $\hat{\mathsf{sk}}_k$ to sk_k and set $\mathsf{st}_{k,\mathsf{eid},1} \leftarrow (\mathsf{cm}_k, R_k, r_k, \mathsf{sk}_k)$.
– Else, if there exists R_k such that $(k, R_k, \perp, \mathsf{cm}_k, \perp, \perp) \in \mathsf{Q}_{\mathsf{cm}}$, then SIM' aborts. We refer to this bad event as $\mathsf{BadEvent}_6$.

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 $\mathcal{O}^{\mathsf{Corrupt}}$ is perfect.

– 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 $R_k = g^{r_k}$.
– 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.
– 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 $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(k, R_k)$.

SIM''s simulation of key generation and $\mathcal{O}^{\mathsf{Sign}_1}, \mathcal{O}^{\mathsf{Sign}_2}, \mathcal{O}^{\mathsf{Sign}_3}$ is the same as SIM's simulation for Theorem 1 so is perfect. The abort conditions are the same, with the addition of $\mathsf{BadEvent}_6$.

$\mathsf{BadEvent}_6$ occurs with maximum probability $\frac{q}{p}$. 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 happens 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 if $m^* \notin \mathsf{Q}_m$ and $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$. If either check fails, SIM' outputs $\perp$.

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 $\mathsf{BadEvent}_1$, $\mathsf{BadEvent}_2$, $\mathsf{BadEvent}_5$, $\mathsf{BadEvent}_6$. As such,

$$\begin{aligned}|\Pr[W_2] - \Pr[W_1]| &= \Pr[\mathsf{BadEvent}_1] + \Pr[\mathsf{BadEvent}_2] + \Pr[\mathsf{BadEvent}_5] \\ &\quad + \Pr[\mathsf{BadEvent}_6] \\ &= \frac{q^2}{2p} + \frac{3q}{p}\end{aligned} \tag{15}$$

The Reduction $\mathcal{B}_2'$. We now construct a PPT reduction $\mathcal{B}_2'$ against the AOMDL assumption (Fig. 5). Specifically, we show how $\mathcal{B}_2'$ uses the adaptive unforgeability adversary $\mathcal{A}$ playing against SIM' as defined in Game 2 as a subroutine such that

$$\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}_2'}(\kappa) \geq \mathsf{accept}(\mathsf{Fork}^{\mathsf{SIM}'}(X)) - 2\Pr[\mathsf{BadEvent}_1] - \cdots - 2\Pr[\mathsf{BadEvent}_6] \tag{16}$$

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 $\mathcal{G} = (\mathbb{G}, p, g)$ 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 AOMDL solution oracle.

To simulate $\mathcal{O}^{\mathsf{sdl}}$ queries by SIM', $\mathcal{B}_2'$ initializes a table Q_{dl}, which it uses to cache $\mathcal{O}^{\mathsf{sdl}}$ 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 8, on the simulating algorithm SIM' and its instance X.

When SIM' queries $\mathcal{O}^{\mathsf{sdl}}$, $\mathcal{B}_2'$ 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 $\mathcal{O}^{\mathsf{sdl}}$ 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 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 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 6, instantiated with $\mathsf{Sparkle}+$, with input $\mathsf{frac} = 1$.

Let $\mathcal{A}$ be an algebraic adversary attempting to break the adaptive unforgeability of $\mathsf{Sparkle}+$ (Fig. 6). Because $\mathcal{A}$ is algebraic, when it queries its oracles on a group element $X \in \mathbb{G}$, it additionally supplies a representation of that element with respect to all other group elements it has seen previously. $\mathcal{A}$ is allowed to make 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. Without loss of generality, we assume $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ 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 simply

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

Game 1. Game 1 is identical to Game 1 in the proof for Theorem 1.

Reduction to EUF-CMA 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. The probability of not choosing a specific value τ when selecting t values without replacement from a set of size n is $\frac{(n-t)}{n}$. (Set $t := t/2$ in the combinatorial argument in Appendix B.)

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 to $\mathcal{A}$ 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.

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

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

The Reduction $\mathcal{B}''_2$. 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

$$\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}''_2}(\kappa) \geq \mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sparkle}+}(\kappa, 1) - \Pr[\mathsf{BadEvent}_1] - \cdots - \Pr[\mathsf{BadEvent}_6] \tag{19}$$

where X is the instance given by the AOMDL challenger.

Initialization. $\mathcal{B}_2''$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and an AOMDL challenge of $t+1$ values $(Y_0, \ldots, Y_t)$. As in $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}_2''}(\kappa)$, $\mathcal{B}_2''$ 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 AOMDL solution oracle.

Simulation. The simulation by $\mathcal{B}_2''$ is identical to that of SIM' as in Game 2 for Theorem 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 if $m^* \notin \mathsf{Q}_m$ and $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$. If either check fails, $\mathcal{B}_2''$ outputs $\perp$. Otherwise, $\mathcal{B}_2''$ looks up the entry $(\mathsf{pk}, m^*, R^*, c^*) \in \mathsf{Q}_{\mathsf{sig}}$ 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 $\Pr[\mathsf{BadEvent}_1], \ldots, \Pr[\mathsf{BadEvent}_6]$, as given in Equation 18 and Equation 19.

Extracting an AOMDL solution given a discrete logarithm of Y_0. Assume without loss of generality that $\mathcal{A}$ makes t corruptions over the course of the protocol. (If not, $\mathcal{B}_2''$ can corrupt the remaining number at the end, by querying its DL oracle until it reaches t secret keys.) Recall that $\mathcal{B}_2''$ set $X_i = Y_0 Y_1^i \cdots Y_t^{i^t}$ and made t 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,

$$\begin{pmatrix} x \\ x_{i_1} \\ \vdots \\ x_{i_t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & i_1 & \cdots & {i_1}^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_t & \cdots & {i_t}^t \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{pmatrix} \tag{20}$$

$\mathcal{B}_2''$ knows all of the values $\{x_j\}_{j \in \mathsf{cor}} = \{x_{i_1}, \ldots, x_{i_t}\}$ on the left-hand side, but not x. However, $\mathcal{B}_2''$ can compute x as follows.

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

$$R^* = g^{z^*}\,\mathsf{pk}^{-c^*} \tag{21}$$

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:

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

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. (21), 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:

$$\begin{aligned} &g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1}\rho_{i,1}^*} \cdots g^{-z_{i,n}\rho_{i,n}^*} \\ &\qquad = \mathsf{pk}^{c^*}\,\mathsf{pk}^{\xi^*} X_1^{\xi_1^*} \cdots X_n^{\xi_n^*} \prod_{i=1}^{q_S} (X_1^{-c_i\lambda_{i,1}})^{\rho_{i,1}^*} \cdots (X_n^{-c_i\lambda_{i,n}})^{\rho_{i,n}^*} \end{aligned} \tag{22}$$

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. (22) can be rewritten as:

$$g^{\eta^*} = \mathsf{pk}^{c^*+\xi^*} X_1^{\zeta_1^*} \cdots X_n^{\zeta_n^*} \tag{23}$$

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

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

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

$$X_k = \mathsf{pk}^{\mu_k^*} g^{\nu_k^*}$$

and Eq. (23) can be rewritten as:

$$g^{\eta^*} = \mathsf{pk}^{c^* + \xi^*} \prod_{k=1}^{n} X^{\mu_k^*} g^{\nu_k^*}$$

Rearranging, we have:

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

and

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

$\mathcal{A}$ 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.

The matrix

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

in Equation 20 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 17, 18, and 19 gives Equation 5. This completes the proof. □