Fully Adaptive Schnorr Threshold Signatures*

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

[1] University of Edinburgh, UK
[2] University of Waterloo & Zcash Foundation
[3] Ethereum Foundation & PQShield, UK
ecrites@ed.ac.uk, 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; moreover, the reduction is tight.

* A preliminary version of this paper appears in the proceedings of CRYPTO 2023. This is the full version.

Table of Contents

1 Introduction

A threshold signature scheme allows a set of n possible signers to jointly produce a signature over a message and with respect to a single public key, so long as at least a threshold $t+1$ of signers participate. Importantly, threshold signature schemes are secure even if t signers are under adversarial control. A recent line of work has explored multi-party signature schemes whose output is a standard (single-party) Schnorr signature [39, 33, 35]. Schnorr signatures admit an efficient and compact representation even in the multi-party setting, which makes them of particular interest for practical use [17].

Static vs. Adaptive Security. Most threshold signature schemes in the literature are proven *statically* secure. 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 [14]. While there are generic methods for transforming a statically secure scheme into an adaptively secure one [16], 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 erasure of secret state [16], which is not easily enforced in practice, or heavyweight tools, such as non-committing encryption [31].

In this work, we investigate the adaptive security of a simple three-round threshold Schnorr signature scheme, which we call $\mathsf{Sparkle}$, under different corruption models and security assumptions. Achieving adaptive security is not just of theoretical interest: NIST recently published a call for threshold EdDSA/Schnorr schemes and included adaptive security as a main goal [11, 12], 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; moreover, the reduction is tight.

Concurrent Security. A *concurrent* adversary may open an arbitrary number of signing sessions simultaneously and unforgeability should still hold. Concurrent security 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 DEFKLNS19 [19] and exhibited in polynomial time by BLLOR21 [8]. Our security reductions for $\mathsf{Sparkle}$ hold against a concurrent and adaptive adversary. Combining concurrency and adaptivity in a multi-party, multi-round signature protocol is the main technical achievement of this work.

Schnorr Threshold Signature $\mathsf{Sparkle}$. In this paper, we prove the adaptive security of a simple three-round threshold Schnorr signature scheme $\mathsf{Sparkle}$ that

Scheme	Static Assumptions	Sign					Combine	
		Performance			Bandwidth		Performance	
		rounds	exp	H	$\mathbb{G}$	$\mathbb{F}$	exp	H
FROST [33]	OMDL+ROM	2	$t+2$	$t+1$	2	1	t	t
FROST2 [5]	OMDL+ROM	2	3	2	2	1	1	1
Lindell22 [35]	DL+ROM	3	$11t+1$	$18t+10^7$	11	25	0	0
Sparkle	DL+ROM	3	1	$t+2$	1	2	0	0

Fig. 1. Efficiency of Two- and Three-Round Schnorr Threshold Signature Schemes. All output a standard Schnorr signature. We only compare schemes that are secure against ROS attacks [8]. 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 [22], where $r = 8$ is the number of commitments for a Fischlin proof and the length of the zero vector is $b = 16$, such that $b \cdot r = 128$.

follows a commit-reveal paradigm. 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. Our security reduction incurs no additional tightness loss compared to the security reduction for plain Schnorr signatures [41]. We compare the efficiency of Sparkle with existing two- and three-round threshold Schnorr signatures in Table 1.

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 [39]. Importantly, we do so without requiring secure erasure of secret state. In the $t+1$-aomdl game, the adversary is given as input an AOMDL challenge that is a vector of group elements of length $t+1$. The adversary is 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 [7] because the adversary only receives the discrete logarithm of linear combinations of its challenge elements, which allows the oracle to run in polynomial time and makes it a falsifiable assumption [4, 29, 32].

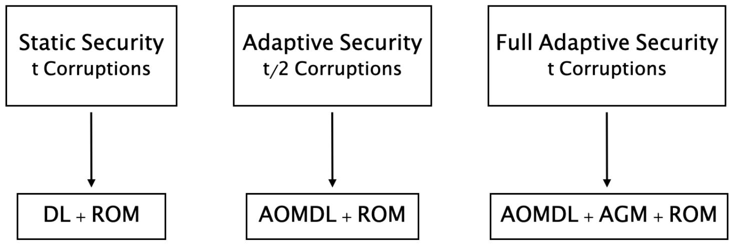

Fig. 2. Comparison of security models and assumptions for our threshold Schnorr signature scheme Sparkle. The signing threshold is $t+1$. DL is the Discrete Logarithm Assumption, and AOMDL is the Algebraic One-More Discrete Logarithm Assumption. ROM is the Random Oracle Model, and AGM is the Algebraic 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) [23] 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.

2 Related Work

Threshold Schnorr Signatures. Closest to the design of Sparkle is the MSDL scheme presented by Boneh, Drijvers, and Neven [10], the three-round MuSig scheme by Maxwell, Poelstra, Seurin, and Wuille [38], and the 2Schnorr scheme by Nicolosi, Krohn, Dodis, and Mazières [40]. However, MSDL and MuSig consider only the multi-signature setting (n-out-of-n), and 2Schnorr considers only the 2-out-of-2 setting. Concurrent to this work, Makriyannis [37] defines a commit-reveal threshold Schnorr signature scheme similar to Sparkle and proves security with respect to an idealized notion of threshold signatures by CGGMP21 [15].

They consider adaptive security but employ the guessing argument that incurs exponential tightness loss relative to the number of parties.

Stinson and Strobl [44] present a threshold Schnorr 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) [27] to generate the nonce for each signature, which adds considerable network overhead: at a minimum, it requires participants to perform four rounds in total. Further, the proof of security assumes only a static adversary.

Komlo and Goldberg [33] present a two-round threshold Schnorr signature FROST. Unlike prior threshold Schnorr schemes in the literature [26], FROST is secure against a concurrent adversary and is not susceptible to ROS attacks [8]. FROST2, introduced in [5], is an optimized version of FROST that reduces the number of exponentiations required for signing from $t+1$ to one. (See Table 1 for a comparison of efficiency.) However, both FROST and FROST2 are proven secure assuming a static adversary and the one-more discrete logarithm assumption (OMDL) [5]. While Sparkle adds an additional round of communication, it only requires a single exponentiation per signer and can be proven secure under standard assumptions, which are also criteria of interest in [11, 12].

Lindell presents a three-round threshold Schnorr signature [35] with the goal of defining a threshold scheme that is secure against ROS attacks and proven secure in the random oracle model under the discrete logarithm assumption only. Security is modeled in the the universally composable framework (UC) [13] 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 ROM and DL assumptions, but does not require the use of online-extractable zero-knowledge proofs [22], and hence is both significantly more efficient and a simpler design.

Adaptive Security of Threshold Signatures. While adaptive security for threshold schemes is a well-known topic in the literature, to the best of our knowledge, no proof of adaptive 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 [16, 31, 36]. However, these techniques introduce prohibitive performance overhead, such as requiring a robust distributed key generation mechanism (DKG) for nonce generation.

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 [34] present a variant of the threshold BLS [9] scheme that is adaptively secure. However, their variant is incompatible with single-party BLS verification, an often-critical goal for threshold schemes in practice. Bacho and Loss [3] demonstrate the adaptive security of threshold BLS 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.

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 [34]. Alternative definitions of adaptive security in the UC setting have been proposed by CGGMP21 [15]. 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 [35] 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 [15], it is arguably overly strong in the sense that it necessitates certain design decisions, such as incorporating online-extractable zero-knowledge proofs. Indeed, [35] elected for Fischlin proofs (see Table 1). The only method to bias the nonces in both $\mathsf{Sparkle}$ and [35] is to abort; this is not the case in FROST [33]. However, all three works allow aborts, and therefore the distribution of the secret randomness cannot be considered uniform [20].

3 Preliminaries

3.1 General Notation

Let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ its unary representation. A function $f : \mathbb{N} \to \mathbb{R}$ is called *negligible* if for all $c \in \mathbb{R}, c > 0$, there exists $k_0 \in \mathbb{N}$ such that $|f(k)| < \frac{1}{k^c}$ 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 $\boldsymbol{a} = (a_1, \ldots, a_n)$.

Let PPT denote probabilistic polynomial time. Algorithms are randomized unless explicitly noted otherwise. Let $y \leftarrow A(x; \omega)$ denote running algorithm A on input x and randomness ω and assigning its output to y. Let $y \leftarrow\!\!\$\, A(x)$ denote $y \leftarrow A(x; \omega)$ for a uniformly random ω. 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 $\mathbb{G}$.

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:

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

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:

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

3.2 Definitions and 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 3, be as follows:*

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

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

Assumption 2 (One-More Discrete Logarithm Assumption (OMDL)) [7] *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 3, be as follows:*

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

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

Algebraic One-More Discrete Logarithm Assumption (AOMDL). The algebraic one-more discrete logarithm assumption (AOMDL) was formalized by Jonas, Ruffing, and Seurin [39]. We highlight differences with the OMDL assumption in Figure 3. Note that while the OMDL assumption is not falsifiable (because the adversary is allowed to query for the discrete logarithm of any arbitrary group element), the AOMDL assumption is falsifiable. This is because the input to the discrete logarithm oracle is a linear combination of all challenges issued to the adversary, so it is guaranteed that the environment runs in PPT when its respective adversary runs in PPT. Hence, the AOMDL assumption is a strictly weaker assumption than standard OMDL.

An adversary in the AOMDL game is initialized in exactly the same manner as an OMDL adversary, with the same winning condition. The only distinction between the games is the inputs to and outputs from the discrete logarithm oracle. In the AOMDL setting, the adversary provides a vector $(\alpha, \beta_0 \ldots, \beta_t)$ of coefficients to the discrete logarithm oracle. The oracle then responds with the integer linear combination of these coefficients with respect to the discrete

$\text{MAIN } \mathsf{Game}^{\mathsf{dl}}_{\mathcal{A}}(\kappa)$

$(\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GrGen}(1^\kappa)$
$x \leftarrow_\$ \mathbb{Z}_p;\ X \leftarrow g^x$
$x' \leftarrow_\$ \mathcal{A}((\mathbb{G}, p, g), X)$
if $x' = x$
  return 1
return 0

$\text{MAIN } \mathsf{Game}^{t+1\text{-}\boxed{\mathsf{a}}\,\mathsf{omdl}}_{\mathcal{A}}(\kappa)$

$(\mathbb{G}, p, g) \leftarrow_\$ \mathsf{GrGen}(1^\kappa)$
$Q \leftarrow \emptyset$
$q \leftarrow 0$
for $i \in [0..t]$ **do**
  $x_i \leftarrow_\$ \mathbb{Z}_p;\ X_i \leftarrow g^{x_i}$
  $Q[X_i] = x_i$
$\boldsymbol{x} \leftarrow (x_0, \ldots, x_t)$
$\boldsymbol{X} \leftarrow (X_0, X_1, \ldots, X_t)$
$\boldsymbol{x'} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{dl}}}((\mathbb{G}, p, g), \boldsymbol{X})$
if $\boldsymbol{x'} = \boldsymbol{x} \ \wedge\ q < t + 1$
  return 1
return 0

$\mathcal{O}^{\mathsf{dl}}(X, \boxed{\alpha, \{\beta_i\}_{i=0}^{t}})$

$\boxed{/\!/ \ X = g^{\alpha} \prod_{i=0}^{t} X_i^{\beta_i}}$
$q \leftarrow q + 1$
$\boxed{x \leftarrow \alpha + \sum_{i=0}^{t} x_i \beta_i}$
$\boxed{\textbf{return } x}$
$x \leftarrow \mathsf{dlog}(X)$
return x

Fig. 3. The Discrete Logarithm (DL), One-More Discrete Logarithm (OMDL), and Algebraic One-More Discrete Logarithm (AOMDL) games. $\mathbb{G}$ is a cyclic group with prime order p and generator g. dlog is an algorithm that finds the discrete logarithm relation of the input X and g. 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 for linear combinations of its challenge group elements; in the OMDL game, the environment must return the discrete logarithm of an arbitrary group element.

---

logarithm of the set of challenges $(X_0, \ldots, X_t)$. It is easy to see that the AOMDL game can be used to win the OMDL game, by the adversary simply querying $\mathcal{O}^{\mathsf{dl}}$ with a bit vector, to return the discrete logarithm of the term whose coefficient is set to one.

Assumption 3 (Algebraic One-More Discrete Logarithm Assumption) [39] *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 3, be as follows:*

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

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

Assumption 4 (Algebraic Group Model (AGM) [23]) *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 $\boldsymbol{a} = (a_0, a_1, a_2, \ldots)$ such that $Z = g^{a_0} \prod Y_i^{a_i}$, where $Y_1, Y_2, \dots \in \mathbb{G}$ are group elements that the adversary has seen thus far.*

Definition 1 (Schnorr Signatures [42]). *The Schnorr signature scheme consists of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, defined as follows:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: On input a security parameter, run* $(\mathbb{G}, p, g) \leftarrow\$\, \mathsf{GrGen}(1^\kappa)$ *and select a hash function* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$*. Output public parameters* $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H})$ *(which are given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$*: On input a security parameter, sample a secret key* $x \leftarrow\$\, \mathbb{Z}_p$ *and compute a public key* $X \leftarrow g^x$*. Output key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow (X, x)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*: On input the secret key* $\mathsf{sk} = x$ *and a message* m*, sample a nonce* $r \leftarrow\$\, \mathbb{Z}_p$ *and 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 signature* $\sigma \leftarrow (R, z)$*.*
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 0/1$*: On input the 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* $RX^c = g^z$*; 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 [21]. Schnorr signatures are secure under the discrete logarithm assumption in the random oracle model [41].

Definition 2 (Shamir Secret Sharing [43]). *Shamir secret sharing is an* $(n, t+1)$*-threshold secret sharing scheme consisting of algorithms* $(\mathsf{IssueShares}, \mathsf{Recover})$*, defined as follows:*

- $\mathsf{IssueShares}(x, n, t+1) \to \{(1, x_1), \ldots, (n, x_n)\}$*: On input a secret* x*, number of participants* n*, and threshold* $t+1$*, perform the following. First, define a polynomial* $f(Z) = x + a_1 + a_2Z^2 + \cdots + a_tZ^t$ *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*
$$\lambda_i = \prod_{j \in \mathcal{S}, j \neq i} \frac{j}{i - j}$$

4 Threshold Signature Schemes

We begin with the definition of a threshold signature scheme. We build upon prior definitions in the literature [25, 28, 24], 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. We then provide game-based definitions of static and adaptive security for threshold signatures. Our definitions model a three-round signing protocol but can be adapted to schemes with fewer or more rounds.

Definition 3 (Threshold Signatures). *A threshold signature scheme* TS *is a tuple of polynomial-time algorithms* $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, (\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}''), \mathsf{Combine}, \mathsf{Verify})$*, defined as follows:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: Takes as input a security parameter and outputs public parameters* par *(which are given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}(n, t+1) \to (\tilde{X}, \{\tilde{X}_i\}_{i\in[n]}, \{x_i\}_{i\in[n]})$*: A probabilistic algorithm that takes as input the number of signers* n *and the threshold* $t+1$ *and outputs the public key* $\tilde{X}$ *representing the set of* n *signers, the set* $\{\tilde{X}_i\}_{i\in[n]}$ *of public keys for each signer, and the set* $\{x_i\}_{i\in[n]}$ *of secret shares for each signer. It is assumed that participant* i *is sent its respective share* x_i *privately.*
- $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'') \to \{\rho_i, \rho_i', \rho_i''\}_{i\in\mathcal{S}}$*: A set of probabilistic algorithms where each subsequent algorithm represents a single stage in an interactive signing protocol, performed by each signing party in a signing set* $\mathcal{S} \subseteq [n], |\mathcal{S}| \geq t+1$ *with respect to a message* m*, defined as follows:*

$$\begin{aligned}(\rho_i, \mathsf{st}_i) &\leftarrow \mathsf{Sign}(m, \mathcal{S})\\ (\rho_i', \mathsf{st}_i) &\leftarrow \mathsf{Sign}'(\mathsf{st}_i, x_i, \{\rho_j\}_{j\in\mathcal{S}})\\ \rho_i'' &\leftarrow \mathsf{Sign}''(\mathsf{st}_i, x_i, \{\rho_j'\}_{j\in\mathcal{S}})\end{aligned}$$

  $\rho_i, \rho_i', \rho_i''$ *are protocol messages, and* st_i *is the state of signing party* i *in* $\mathcal{S}$*.*
- $\mathsf{Combine}(\{(\rho_i', \rho_i'')\}_{i\in\mathcal{S}}) \to (m, \sigma)$*: A deterministic algorithm that takes as input the set of protocol messages* (ρ_i', ρ_i'') *sent by each party during the* Sign' *and* Sign'' *stages and outputs a signature* σ*.*
- $\mathsf{Verify}(\tilde{X}, m, \sigma) \to 0/1$*: A deterministic algorithm that takes as input the public key* $\tilde{X}$*, a message* m*, and purported signature* σ *and outputs* 1 *(accept) if the signature verifies; else, it outputs* 0 *(reject).*

Remark 1 (Distributed key generation). Our definition assumes a centralized key generation algorithm KeyGen to generate the public key $\tilde{X}$ and set of shares $\{\tilde{X}_i, x_i\}_{i\in[n]}$. However, our scheme and proofs can be adapted to use a fully decentralized distributed key generation protocol (DKG). We refer to [3] for a discussion on how to achieve adaptively secure DKGs.

Correctness. A threshold signature scheme is *correct* if for all security parameters κ, all allowable $1 \leq t+1 \leq n$, all $\mathcal{S}$ such that $t+1 \leq |\mathcal{S}| \leq n$, all messages $m \in \{0,1\}^*$, and for $(\tilde{X}, \{\tilde{X}_i, x_i\}_{i\in[n]}) \leftarrow\$ \mathsf{KeyGen}(n, t+1)$, if all signers in $\mathcal{S}$ input (x_i, m) to the signing protocol $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'')$, then every signer will output a signature share that, when combined with all other shares, results in a signature σ satisfying $\mathsf{Verify}(\tilde{X}, m, \sigma) = 1$.

4.1 Static Security

We present a game-based definition of static security for threshold signatures analogous to existential unforgeability against chosen message attack (EUF-CMA) for standard signature schemes [30]. The static security game is defined formally in Figure 4, where it contains all but the dashed boxes. In addition to the security parameter κ, the game additionally accepts a parameter τ, which specifies the fraction of t signers that the adversary may corrupt. A static adversary can corrupt a full t signers out of at least $t+1$ signers in a session, so the fraction is $\tau = 1$.

In the static unforgeability game, the challenger generates public parameters par and returns them to the adversary. The adversary must now choose the set of corrupt participants cor, which are fixed for the duration of the protocol. The challenger then runs KeyGen to derive the joint public key $\tilde{X}$ representing the set of n signers, the individual public key shares $\{\tilde{X}_i\}_{i \in [n]}$, and the secret key shares $\{x_i\}_{i \in [n]}$. It returns $\tilde{X}$, $\{\tilde{X}_i\}_{i \in [n]}$, and the set of corrupt shares $\{x_j\}_{j \in \mathsf{cor}}$ to the adversary.

After key generation has concluded, the adversary can query honest signers $k \in \mathsf{hon}$ at each step in the signing protocol by querying oracles $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}$, and has full power over choosing the set of signers $\mathcal{S}$ and the message m to be signed. The signing session identifier is denoted by ssid.

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

The adversary may be *rushing*, meaning it can wait to produce its outputs after having seen the honest outputs first (represented by protocol messages ρ, ρ', ρ'' in Fig. 4). The adversary may also be *concurrent*, meaning it can open simultaneous signing sessions at once, or choose not to complete a signing session (e.g., the adversary might query $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}$, but not $\mathcal{O}^{\mathsf{Sign}''}$, before it starts a new session). By modeling a concurrent adversary, we ensure that our notion of security protects against practical attacks that can occur in such a setting [8].

Definition 4 (Static Security). *Let the advantage of an adversary $\mathcal{A}$ playing the static security game $\mathsf{Game}^{\mathsf{UF}}_{\mathcal{A}}(\kappa, \tau)$, as defined in Figure 4, be as follows:*

$$\mathsf{Adv}^{st\text{-}sec}_{\mathcal{A},\mathsf{TS}}(\kappa, \tau) = \left|\Pr[\mathsf{Game}^{\mathsf{UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, \tau) = 1]\right|$$

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

4.2 Adaptive Security

We build upon the notion of adaptive security for threshold signature schemes by Libert, Joye, and Yung [34]. We provide a formal game-based definition, which is specified in Figure 4. The adaptive unforgeability game contains the same inputs and algorithms as the static game, but additionally includes a corruption oracle $\mathcal{O}^{\mathsf{Corrupt}}$.

$\textsc{main}\ \mathsf{Game}^{\mathsf{UF}}_{\mathcal{A},\mathsf{TS}}(\kappa,\tau)\ \boxed{\mathsf{Game}^{\mathsf{adp\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa,\tau)}$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^{\kappa})$
$W \leftarrow \emptyset$ // open signing sessions for $\mathcal{O}^{\mathsf{Sign}}$
$W' \leftarrow \emptyset$ // open signing sessions for $\mathcal{O}^{\mathsf{Sign}'}$
$W'' \leftarrow \emptyset$ // signing sessions for $\mathcal{O}^{\mathsf{Sign}''}$
$Q \leftarrow \emptyset$ // set of queried messages
$(n, t+1, \mathsf{cor}, \mathsf{st}_{\mathcal{A}}) \leftarrow\$\ \mathcal{A}(\mathsf{par})$
$\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$
$(\tilde{X}, \{\tilde{X}_i, x_i\}_{i\in[n]}) \leftarrow\$\ \mathsf{KeyGen}(n, t+1)$
$\mathsf{input} \leftarrow (\tilde{X}, \{\tilde{X}_i\}_{i\in[n]}, \{x_j\}_{j\in\mathsf{cor}}, \mathsf{st}_{\mathcal{A}})$
$(m^*, \sigma^*) \leftarrow\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}'',\boxed{\mathsf{Corrupt}}}}(\mathsf{input})$
if $|\mathsf{cor}| \leq \tau \cdot t$ // τ is a fraction
  if $m^* \notin Q \wedge \mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$
    return 1
return 0

$\boxed{\mathcal{O}^{\mathsf{Corrupt}}(k)}$

if $k \notin \mathsf{hon}$ **return** $\perp$
$\mathsf{cor} \leftarrow \mathsf{cor} \cup \{k\}$
$\mathsf{hon} \leftarrow \mathsf{cor} \setminus \{k\}$
return $(x_k, \{\mathsf{st}_{k,\ell}\}_{\ell\in[\mathsf{ssid}]})$

$\mathcal{O}^{\mathsf{Sign}}(k, \mathsf{ssid}, m, \mathcal{S})$

// k denotes the participant identifier
$Q \leftarrow Q \cup \{m\}$
$W \leftarrow W \cup \{(k, \mathsf{ssid})\}$
$(\rho_k, \mathsf{st}_{k,\mathsf{ssid}}) \leftarrow \mathsf{Sign}(m, \mathcal{S})$
return ρ_k

$\mathcal{O}^{\mathsf{Sign}'}(k, \mathsf{ssid}, \{\rho_i\}_{i\in\mathcal{S}})$

if $(k, \mathsf{ssid}) \notin W$ **return** $\perp$
if $(k, \mathsf{ssid}) \in W'$ **return** $\perp$
$W' \leftarrow W' \cup \{(k, \mathsf{ssid})\}$
$(\rho'_k, \mathsf{st}_{k,\mathsf{ssid}}) \leftarrow \mathsf{Sign}'(\mathsf{st}_{k,\mathsf{ssid}}, x_k, \{\rho_i\}_{i\in\mathcal{S}})$
return ρ'_k

$\mathcal{O}^{\mathsf{Sign}''}(k, \mathsf{ssid}, \{\rho'_i\}_{i\in\mathcal{S}})$

if $(k, \mathsf{ssid}) \notin W'$ **return** $\perp$
if $(k, \mathsf{ssid}) \in W''$ **return** $\perp$
$W'' \leftarrow W'' \cup \{(k, \mathsf{ssid})\}$
$\rho''_k \leftarrow \mathsf{Sign}''(\mathsf{st}_{k,\mathsf{ssid}}, x_k, \{\rho'_i\}_{i\in\mathcal{S}})$
return ρ''_k

Fig. 4. Static and adaptive unforgeability games for a threshold signature scheme with three signing rounds. The public parameters par are implicitly given as input to all algorithms, and ρ, ρ', ρ'' represent protocol messages defined within the construction. The static game contains all but the dashed boxes, and the adaptive game adds the dashed boxes. τ specifies the fraction of t signers that the adversary may corrupt. For example, $\tau = 1$ means the adversary can corrupt a full t signers, and $\tau = 1/2$ means it can corrupt $t/2$ signers.

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 $\tau \cdot t$, with respect to the corruption ratio τ. An adaptive adversary may, for example, corrupt $t/2$ (i.e., $\tau = 1/2$) or a full t (i.e., $\tau = 1$) signers out of at least $t+1$ signers in a session.

Definition 5 (Adaptive Security). *Let the advantage of an adversary $\mathcal{A}$ playing the adaptive security game $\mathsf{Game}^{\mathsf{adp\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, \tau)$, as defined in Figure 4, be as follows:*

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

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

5 Schnorr Threshold Signature Scheme Sparkle

Sparkle is a simple three-round Schnorr threshold signature scheme (Fig. 5). We employ a centralized key generation algorithm KeyGen to generate the public key $\tilde{X}$ and set of shares $\{\tilde{X}_i, x_i\}_{i \in [n]}$. The public parameters par generated during setup are provided as input to all other algorithms and protocols. We assume an external mechanism to choose the set of signers $\mathcal{S} \subseteq \{1, \ldots, n\}$, where $t+1 \le |\mathcal{S}| \le n$ and $\mathcal{S}$ is ordered to ensure consistency.

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 $\mathsf{cm}_i \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$, where m is the message. In the second round, each participant reveals R_i in the clear. In the third round, each participant computes the aggregate nonce $\tilde{R} = \prod_{i \in \mathcal{S}} R_i$, the Schnorr challenge $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ using $\tilde{R}$, and their signature share z_i. The signature shares are additively combined via the Combine algorithm to produce the Schnorr signature $\sigma = (\tilde{R}, z = \sum_{i \in \mathcal{S}} z_i)$.

This commit-reveal strategy ensures two security properties. First, requiring each participant to publish a commitment in the first round, before revealing their R_i, prevents a rushing adversary from adaptively choosing R_j as a function of other participants' R_i values. 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 a 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 outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}})$.

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

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

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

// select two hash functions

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

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

return par

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

$x \leftarrow\$ \mathbb{Z}_p;\ \tilde{X} \leftarrow g^x$

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

// Shamir secret sharing of x

for $i \in [n]$ $\tilde{X}_i \leftarrow g^{x_i}$

return $(\tilde{X}, \{\tilde{X}_i, x_i\}_{i \in [n]})$

$\mathsf{Sign}(m, \mathcal{S})$

// local signer has index k

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

$\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$

$\rho_k \leftarrow \mathsf{cm}_k$

$\mathsf{st}_k \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$

return (ρ_k, st_k)

$\mathsf{Sign}'(\mathsf{st}_k, x_k, \{\rho_i\}_{i \in \mathcal{S}})$

parse $\mathsf{cm}_i \leftarrow \rho_i$ for all $i \in \mathcal{S}$

parse $(\mathsf{cm}_k, R_k, r_k, m, \mathcal{S}) \leftarrow \mathsf{st}_k$

return $\perp$ **if** $\mathsf{cm}_k \notin \{\mathsf{cm}_i\}_{i \in \mathcal{S}}$

$\rho'_k \leftarrow R_k$

$\mathsf{st}_k \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S}, \{\rho_i\}_{i \in \mathcal{S}})$

return (ρ'_k, st_k)

$\mathsf{Sign}''(\mathsf{st}_k, x_k, \{\rho'_i\}_{i \in \mathcal{S}})$

// Sign'' must be called once per st_k

parse $(\mathsf{cm}_k, R_k, r_k, m, \mathcal{S}, \{\rho_i\}_{i \in \mathcal{S}}) \leftarrow \mathsf{st}_k$

parse $R_i \leftarrow \rho'_i$ for all $i \in \mathcal{S}$

return $\perp$ **if** $R_k \notin \{R_i\}_{i \in \mathcal{S}}$

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

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

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

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

$z_k \leftarrow r_k + c\lambda_k x_k$

// λ_k is the Lagrange coefficient for k w.r.t. $\mathcal{S}$

$\rho''_k \leftarrow z_k$

return ρ''_k

$\mathsf{Combine}(\{(\rho'_i, \rho''_i)\}_{i \in \mathcal{S}})$

parse $R_i \leftarrow \rho'_i, z_i \leftarrow \rho''_i$ for all $i \in \mathcal{S}$

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

$\sigma \leftarrow (\tilde{R}, z)$

return σ

$\mathsf{Verify}(\tilde{X}, m, \sigma)$

parse $(\tilde{R}, z) \leftarrow \sigma$

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

if $\tilde{R}\tilde{X}^c = g^z$

  return 1

return 0

Fig. 5. The Sparkle threshold signature scheme. The public parameters par are implicitly given as input to all algorithms and protocols. We assume an external mechanism to choose the set of signers $\mathcal{S} \subseteq \{1, \ldots, n\}$, where $t + 1 \leq |\mathcal{S}| \leq n$ and $\mathcal{S}$ is ordered to ensure consistency. Note that verification is identical to a standard (single-party) Schnorr signature as in Definition 1.

corresponding public key as $\tilde{X} \leftarrow g^x$. It then performs a Shamir secret sharing of x (Def. 2): $\{(i, x_i)\}_{i\in[n]} \leftarrow_{\$} \mathsf{IssueShares}(x, n, t+1)$. Each participant's public key is computed as $\tilde{X}_i \leftarrow g^{x_i}$ and it outputs $(\tilde{X}, \{\tilde{X}_i, x_i\}_{i\in[n]})$.

Signing Round 1 (Sign)**.** On input a message m and a signing set $\mathcal{S}$, each participant $i \in \mathcal{S}$ samples $r_i \leftarrow_{\$} \mathbb{Z}_p$, computes $R_i \leftarrow g^{r_i}$ and $\mathsf{cm}_i \leftarrow \mathsf{H_{cm}}(m, \mathcal{S}, R_i)$, and outputs their commitment cm_i.

Signing Round 2 (Sign')**.** On input commitments $\{\mathsf{cm}_j\}_{j\in\mathcal{S}}$, each participant $i \in \mathcal{S}$ outputs their nonce R_i.

Signing Round 3 (Sign'')**.** On input nonces $\{R_j\}_{j\in\mathcal{S}}$, each participant $i \in \mathcal{S}$ first checks that the commitments received in the first round are valid, i.e., $\mathsf{cm}_j = \mathsf{H_{cm}}(m, \mathcal{S}, R_j)$ for all $j \in \mathcal{S}$. If not, return $\perp$. Else, each participant computes the aggregate nonce $\tilde{R} = \prod_{j\in\mathcal{S}} R_j$, $c \leftarrow \mathsf{H_{sig}}(\tilde{X}, m, \tilde{R})$, 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 $\mathcal{S}$. Each participant outputs z_i.

Combining Signatures. On input nonces $\{R_j\}_{j\in\mathcal{S}}$ and partial signatures $\{z_j\}_{j\in\mathcal{S}}$, the combiner computes $\tilde{R} \leftarrow \prod_{j\in\mathcal{S}} R_j$ and $z \leftarrow \sum_{j\in\mathcal{S}} z_j$, and outputs $\sigma \leftarrow (\tilde{R}, z)$.

Verification. On input the joint public key $\tilde{X}$, a message m, and a purported signature $\sigma = (\tilde{R}, z)$, the verifier computes $c \leftarrow \mathsf{H_{sig}}(\tilde{X}, m, \tilde{R})$ and accepts if $\tilde{R}\tilde{X}^c = g^z$.

Correctness of $\mathsf{Sparkle}$ is straightforward to verify. Note that verification of the signature σ is identical to verification of a standard (single-party) Schnorr signature (Def. 1) with respect to the joint public key $\tilde{X}$ and aggregate nonce $\tilde{R}$.

6 Static Security Under Standard Assumptions

In this section, we show that $\mathsf{Sparkle}$ is statically secure under the discrete logarithm assumption (DL) in the random oracle model (ROM). Static security allows an adversary to control up to t parties, but they must be declared at the beginning of the protocol (Fig. 4).

Theorem 1. $\mathsf{Sparkle}$ *is statically secure under DL in the ROM.*

We formally prove Theorem 1 in Section 9.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the static unforgeability of $\mathsf{Sparkle}$ (Fig. 4). We construct a PPT reduction $\mathcal{B}$ against the DL assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine as follows. $\mathcal{B}$ takes as input a DL challenge $\dot{X}$ and aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$. $\mathcal{B}$ sets the joint public key $\tilde{X} \leftarrow \dot{X}$ and performs a standard simulation of Shamir secret sharing. $\mathcal{B}$ then returns $\tilde{X}$, public key shares $\{\tilde{X}_i\}_{i\in[n]}$, and secret key shares $\{x_j\}_{j\in\mathsf{cor}}$ to $\mathcal{A}$.

$\mathcal{B}$ simulates signing without knowing the secret keys $\{x_k\}_{k\in\mathsf{hon}}$ of the honest parties as follows. $\mathcal{B}$ can simulate R_k for all $k \in \mathsf{hon}$ as $R_k \leftarrow g^{z_k}\tilde{X}_k^{-c\lambda_k}$ for random $z_k \leftarrow_{\$} \mathbb{Z}_p$ so that the partial signature z_k output in Round 3 verifies.

However, c must equal $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ for aggregate $\tilde{R}$. Luckily, $\mathcal{B}$ can compute $\tilde{R} = \prod_{i \in \mathcal{S}} R_i$, where $\mathcal{S}$ is the signing set, by extracting R_j for all $j \in \mathsf{cor}$ from $\mathcal{A}$'s $\mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_j)$ queries in Round 1. So, $\mathcal{B}$ samples a random value for $c \leftarrow\!\!\$\, \mathbb{Z}_p$, computes R_k for all $k \in \mathsf{hon}$ as above, and programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ – all before $\mathcal{A}$ sees the honest R_k's output at the end of Round 2. This leaves just one problem: $\mathcal{B}$ needs to output $\mathsf{cm}_k = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ in Round 1 *before* being able to carry out the above steps. But $\mathcal{B}$ can simply output a random value $\mathsf{cm}_k \leftarrow\!\!\$\, \mathbb{Z}_p$ in Round 1 and program $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ in Round 2.

$\mathcal{B}$ then *rewinds* $\mathcal{A}$, programming the single point $c' \leftarrow\!\!\$\, \mathbb{Z}_p; c' \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$ on the second iteration of $\mathcal{A}$. By the local forking lemma [6], $\mathcal{A}$'s two forgeries $(m^*, (\tilde{R}^*, z^*)), (m', (\tilde{R}', z'))$ satisfy $(\tilde{R}^*, m^*) = (\tilde{R}', m')$ with non-negligible probability. Thus, $\mathcal{B}$ can compute $\dot{x} = \frac{z^* - z'}{c^* - c'}$ and win the DL game.

7 Adaptive Security Against up to $t/2$ Corruptions

In this section, we prove the adaptive security of $\mathsf{Sparkle}$ against up to $t/2$ corruptions under the algebraic one-more discrete logarithm assumption (AOMDL) (Fig. 3) 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. $\mathsf{Sparkle}$ *is adaptively secure against* $t/2$ *corruptions under AOMDL in the ROM.*

We formally prove Theorem 2 in Section 9.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the adaptive unforgeability of $\mathsf{Sparkle}$ (Fig. 4). We construct a PPT reduction $\mathcal{B}$ against the $t+1$-aomdl assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine as follows. $\mathcal{B}$ 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 $i \in [n]$, $\mathcal{B}$ sets each public key share as $\tilde{X}_i \leftarrow Y_0 Y_1^i \cdots Y_t^{i^t}$. The joint public key is $\tilde{X} \leftarrow Y_0$. $\mathcal{B}$ queries its DL oracle on $\tilde{X}_j$ (with representation $(1, j, \ldots, j^t)$) to get x_j for the initial corrupted set cor. $\mathcal{B}$ then returns $(\tilde{X}, \{\tilde{X}_i\}_{i \in [n]}, \{x_j\}_{j \in \mathsf{cor}})$ to $\mathcal{A}$.

$\mathcal{B}$'s simulation of signing is similar to the proof of static security. In particular, $\mathcal{B}$ again simulates R_k for honest $k \in \mathsf{hon}$ as $R_k \leftarrow g^{z_k} \tilde{X}_k^{-c\lambda_k}$ in Round 2. If $\mathcal{A}$ corrupts an honest party k after Round 2 has begun, then $\mathcal{B}$ queries its DL oracle on $\tilde{X}_k$ (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 along with the honest party's state $\mathsf{st}_{k,\mathsf{ssid}} = \{\mathsf{cm}_k, R_k, r_k, \ldots\}$ 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, $\mathcal{B}$ samples a random $r_k \leftarrow_\$ \mathbb{Z}_p$, sets $R_k \leftarrow g^{r_k}$, and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ for the random cm_k it output in Round 1. It then queries its DL oracle on $\tilde{X}_k$ (with rep. $(1, k, \ldots, k^t)$) to get x_k and returns it and $\mathsf{st}_{k,\mathsf{ssid}} = \{\mathsf{cm}_k, R_k, r_k, ...\}$ across all signing sessions to $\mathcal{A}$. As in the proof of static security, $\mathcal{B}$ 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 $\mathcal{B}$ can corrupt the remaining itself.) For simplicity, say the corrupt indices are $\mathsf{cor} = \{1, \ldots, t\}$. Then $\mathcal{B}$ 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. $\mathcal{B}$ 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.

8 Adaptive Security Against t Corruptions

We now prove our strongest result: that $\mathsf{Sparkle}$ 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 $\mathsf{Sparkle}$ would still hold. We prove this result under the AOMDL assumption (Fig. 3) in the AGM with random oracles. We also provide a proof of static security under the DL assumption in the AGM and ROM, intended as a warm-up for our adaptive proof, in Appendix A.

Theorem 3. $\mathsf{Sparkle}$ *is adaptively secure against* t *corruptions under the AOMDL assumption in the AGM and ROM.*

We formally prove Theorem 3 in Section 9.

Proof Outline. Let $\mathcal{A}$ be an algebraic adversary against the adaptive unforgeability of $\mathsf{Sparkle}$ (Fig. 4). We construct a PPT reduction $\mathcal{B}$ against the $t+1$-aomdl assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine as follows. $\mathcal{B}$ 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. $\mathcal{B}$ 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 $(m^*, (\tilde{R}^*, z^*))$ verifies as $\tilde{R}^* = g^{z^*} \tilde{X}^{-c^*}$, where $\mathcal{A}$ provided a representation of $\tilde{R}^*$ when querying $c^* = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$:

$$\tilde{R}^* = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} R_{i,1}^{\rho_{i,1}^*} \cdots R_{i,n}^{\rho_{i,n}^*}$$

where $\{R_{i,k}\}_{i\in[q_S]}$ are the honest nonces returned by the Sign' oracle over q_S signing queries. Each $R_{i,k}$ verifies as $R_{i,k} = g^{z_{i,k}} \tilde{X}_k^{-c_i \lambda_{i,k}}$, where $c_i = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m_i, \tilde{R}_i)$. Equating the two expressions for $\tilde{R}^*$ and rearranging, we have:

$$g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1}\rho^*_{i,1}} \cdots g^{-z_{i,n}\rho^*_{i,n}}$$
$$= \tilde{X}^{c^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (\tilde{X}_1^{-c_i\lambda_{i,1}})^{\rho^*_{i,1}} \cdots (\tilde{X}_n^{-c_i\lambda_{i,n}})^{\rho^*_{i,n}}$$

$\mathcal{B}$ queries its DL oracle on $\tilde{X}_j$ (with representation $(1, j, \ldots, j^t)$) to obtain $\{x_j\}_{j\in\mathsf{cor}}$ for the t corrupt parties. For all $k \in [n]$, $\tilde{X}_k = \tilde{X} Y_1^k Y_2^{k^2} \cdots Y_t^{k^t}$, and for all $i \in [t]$, $Y_i = \tilde{X}^{L'_{0,i}} \prod_{j\in\mathsf{cor}} g^{x_j L'_{j,i}}$, 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:

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

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 $\tilde{R}^*$ and thus $\eta^*, \{\nu_i^*\}_{i\in[n]}, \xi^*, \{\mu_i^*\}_{i\in[n]}$ as it queried $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$ to receive random c^*. Thus, the denominator is nonzero with overwhelming probability and $\mathcal{B}$ can solve for x. $\mathcal{B}$ can then compute $(y_0, y_1, \ldots, y_t)$ as in the $t/2$-adaptive proof to win the $t+1$-aomdl game.

9 Static and Adaptive Security Proofs

In this section, we provide formal proofs of static and adaptive security for $\mathsf{Sparkle}$. In particular, we prove the following: (1) static security (against up to t corruptions) (Theorem 1) under DL+ROM, (2) adaptive security against up to $t/2$ corruptions (Theorem 2) under AOMDL+ROM, and (3) adaptive security against up to t corruptions (Theorem 3) under AOMDL+AGM+ROM.

9.1 Proof of Static Security

Proof. (of Theorem 1.) Let $\mathcal{A}$ be a PPT adversary attempting to break the static unforgeability of $\mathsf{Sparkle}$ (Fig. 4) that makes up to q_H queries to H_{cm} and $\mathsf{H}_{\mathsf{sig}}$, and q_S queries to its signing oracles. Without loss of generality, we assume $\mathcal{A}$

queries $\mathsf{H_{sig}}$ on its forgery $(\tilde{X}, m^*, \tilde{R}^*)$. Then, let $q = q_H + q_S + 1$. We construct a PPT reduction $\mathcal{B}$ against the DL assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}^{st\text{-}sec}_{\mathcal{A}}(\kappa, 1) \leq \sqrt{q\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\kappa) + \mathsf{negl}(\kappa)}$$

Here, $\tau = 1$ to allow $\mathcal{A}$ to corrupt a full $\tau \cdot t = t$ parties. The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times. On the second iteration, $\mathcal{B}$ programs $\mathsf{H_{sig}}$ to output a different random value on a single point so that it can extract a discrete logarithm solution from $\mathcal{A}$'s two forgeries. $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{UF}}_{\mathcal{A}}(\kappa, 1)$. However, $\mathcal{B}$ can only extract a discrete logarithm if $\mathcal{A}$'s forgery $(m^*, (\tilde{R}^*, z^*))$ at the end of each iteration verifies and includes the same nonce $\tilde{R}^*$. By the local forking lemma [6], this occurs with probability less than $\frac{1}{q}(\mathsf{Adv}^{st\text{-}sec}_{\mathcal{A}}(\kappa, 1))^2$.

The Reduction $\mathcal{B}$: We define the reduction $\mathcal{B}$ playing game $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}}(\kappa)$ as follows. $\mathcal{B}$ is responsible for simulating key generation and oracle responses for queries to $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}, \mathsf{H_{cm}}$, and $\mathsf{H_{sig}}$. Let $\mathsf{Q_{cm}}, \mathsf{Q_{sig}}$ be the set of $\mathsf{H_{cm}}, \mathsf{H_{sig}}$ queries and their responses, respectively. $\mathcal{B}$ initializes them to the empty set and maintains them across both iterations of the adversary. $\mathcal{B}$ may program the random oracles $\mathsf{H_{cm}}, \mathsf{H_{sig}}$. Let Q be the set of messages that have been queried in $\mathcal{O}^{\mathsf{Sign}}$ as in $\mathsf{Game}^{\mathsf{UF}}_{\mathcal{A}}(\kappa, 1)$. $\mathcal{B}$ initializes Q to the empty set. At the beginning of the second iteration of $\mathcal{A}$, $\mathcal{B}$ resets Q to the empty set.

DL Input. $\mathcal{B}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and a DL challenge $\dot{X}$. $\mathcal{B}$ aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$.

Static Corruption. $\mathcal{B}$ runs $\mathcal{A}()$. $\mathcal{A}$ 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. $\mathcal{B}$ sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which $\mathcal{B}$ does in the next step.

Simulating KeyGen. $\mathcal{B}$ simulates the key generation algorithm (Fig. 5) using its DL challenge $\dot{X}$ as follows.

1. $\mathcal{B}$ sets the joint public key $\tilde{X} \leftarrow \dot{X}$.
2. $\mathcal{B}$ simulates a Shamir secret sharing of the discrete logarithm of $\tilde{X}$ by performing the following steps. (See Section 3 for notation.) Assume without loss of generality that $|\mathsf{cor}| = t$.
   (a) $\mathcal{B}$ 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 $j \in \mathsf{cor}$. $\mathcal{B}$ 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 $1 \leq i \leq t$, $\mathcal{B}$ computes

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

   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 $1 \leq i \leq n$, $\mathcal{B}$ computes

$$\tilde{X}_i = \tilde{X} Y_1^i Y_1^{i^2} \cdots Y_t^{i^t}$$

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

The joint public key is $\tilde{X} = g^{f(0)} = \dot{X}$ with corresponding secret key $x = \dot{x}$. $\mathcal{B}$ runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}''}}(\tilde{X}, \{\tilde{X}_i\}_{i\in[n]}, \{x_j\}_{j\in\mathsf{cor}})$.

Simulating Random Oracle Queries. $\mathcal{B}$ handles $\mathcal{A}$'s random oracle queries throughout the protocol by lazy sampling, as follows.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on $(m, \mathcal{S}, R)$, $\mathcal{B}$ checks whether $(m, \mathcal{S}, R, \mathsf{cm}) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm. Else, $\mathcal{B}$ samples $\mathsf{cm} \leftarrow\!\!\$\, \mathbb{Z}_p$, appends $(m, \mathcal{S}, R, \mathsf{cm})$ to Q_{cm}, and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\!\$\, \mathbb{Z}_p$, appends (X, m, R, c) to $\mathsf{Q}_{\mathsf{sig}}$, and returns c.

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

Round 1 ($\mathcal{O}^{\mathsf{Sign}}$): In the first round of signing for session ssid, each party i in the signing set $\mathcal{S}$ sends a commitment cm_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}}$ on $(k, \mathsf{ssid}, m, \mathcal{S})$ for honest $k \in \mathsf{hon}$, $\mathcal{B}$ samples $\mathsf{cm}_k \leftarrow\!\!\$\, \mathbb{Z}_p$, appends $(m, \mathcal{S}, \cdot, \mathsf{cm}_k)$ to Q_{cm}, sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, \cdot, \cdot, m, \mathcal{S})$, and returns cm_k to $\mathcal{A}$.

Round 2 ($\mathcal{O}^{\mathsf{Sign}'}$): In the second round of signing for session ssid, each party i in the signing set $\mathcal{S}$ takes as input the set of commitments $\{\mathsf{cm}_i\}_{i\in\mathcal{S}}$ and reveals its nonce R_i such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}'}$ on $(k, \mathsf{ssid}, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ for $k \in \mathsf{hon}$, then for all $j \in \mathcal{S} \cap \mathsf{cor}$, $\mathcal{B}$ looks up cm_j for a record $(m, \mathcal{S}, R_j, \mathsf{cm}_j) \in \mathsf{Q}_{\mathsf{cm}}$.

If there exists $j' \in \mathcal{S}$ such that $(m', \mathcal{S}', \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ and $(m', \mathcal{S}') \neq (m, \mathcal{S})$, or if there exists some $j' \in \mathcal{S}$ for which no record $(m, \mathcal{S}, \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ exists, then $\mathcal{B}$ chooses $R_k \leftarrow\!\!\$\, g^{r_k}$ randomly, updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$, programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ and returns R_k.

Else if this is the first query in the signing session, $\mathcal{B}$ samples $c \leftarrow\!\!\$\, \mathbb{Z}_p$. For all $k \in \mathsf{hon}$, $\mathcal{B}$ samples $z_k \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $R_k \leftarrow g^{z_k} \tilde{X}_k^{-c\lambda_k}$ (λ_k is the Lagrange coefficient for party k in the set $\mathcal{S}$), and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ (updating $(m, \mathcal{S}, R_k, \mathsf{cm}_k) \in \mathsf{Q}_{\mathsf{cm}}$). Then $\mathcal{B}$ computes $\tilde{R} = \prod_{i\in\mathcal{S}} R_i$ and programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$. (However, if $\mathcal{A}$ has already queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$, then $\mathcal{B}$ aborts.)

Finally, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, and returns R_k to $\mathcal{A}$. If this is not the first query, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$ and returns R_k.

Round 3 ($\mathcal{O}^{\mathsf{Sign}''}$): In the third round of signing for session ssid, each party i in the signing set $\mathcal{S}$ produces a partial signature on the message m. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}''}$ on $(k, \mathsf{ssid}, \{R_i\}_{i\in\mathcal{S}})$ for $k \in \mathsf{hon}$, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m,$

$\mathcal{S}, \{\mathsf{cm}_i\}_{i\in\mathcal{S}})$, checks whether $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ for all $i \in \mathcal{S}$ and returns $\perp$ if not. If $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ but $\mathcal{A}$ never queried H_{cm} on input $(m, \mathcal{S}, R_i)$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow ()$ and returns z_k.

Output. At the end of the game, $\mathcal{A}$ produces a forgery $(m^*, \sigma^*) = (m^*, (\tilde{R}^*, z^*))$ and wins if $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$ and $m^* \notin Q$.

Extracting the Discrete Logarithm of $\dot{X}$. $\mathcal{B}$'s simulation of key generation and signing is perfect, and $\mathcal{B}$ aborts with negligible probability. Indeed, $\mathcal{B}$ aborts in Round 3 if $\mathcal{A}$ reveals R_j such that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_j)$ but $\mathcal{A}$ never queried H_{cm} on $(m, \mathcal{S}, R_j)$. This requires $\mathcal{A}$ to have guessed cm_j ahead of time, which occurs with negligible probability $1/p$.

$\mathcal{B}_2$ also aborts in Round 2 if $\mathcal{A}$ had previously queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$. In that case, $\mathcal{B}$ had returned a random $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_k before $\mathcal{B}$ revealed it, which occurs with negligible probability $1/p$.

It remains to show that $\mathcal{B}$ can extract the discrete logarithm of $\dot{X}$ from $\mathcal{A}$'s two valid forgeries. $\mathcal{A}$'s first forgery $(m^*, (\tilde{R}^*, z^*))$ satisfies $\tilde{R}^*\tilde{X}^{c^*} = g^{z^*}$, where $c^* = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$. Here, z^* does not suffice for $\mathcal{B}$ to extract the discrete logarithm of $\tilde{X}$ because it does not necessarily know the discrete logarithm of $\tilde{R}^*$. Thus, $\mathcal{B}$ chooses $c' \leftarrow\!\!\$\, \mathbb{Z}_p$ and programs $\mathsf{H}_{\mathsf{sig}}$ to output c' on input $(\tilde{X}, m^*, \tilde{R}^*)$. $\mathcal{B}$ resets Q to the empty set, but the sets Q_{cm}, $\mathsf{Q}_{\mathsf{sig}}$ are kept for the second iteration of the adversary. $\mathcal{B}$ then runs $\mathcal{A}$ again on the same random coins.

After the second iteration, suppose $\mathcal{A}$ terminates with $(m', (\tilde{R}', z'))$. If $(m', \tilde{R}') = (m^*, \tilde{R}^*)$ and $\mathcal{A}$'s forgeries both verify, then $\mathcal{B}$ returns $\dot{x} = \frac{z^*-z'}{c^*-c'}$ such that $\tilde{X} = g^{\dot{x}}$ and wins $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}}(\kappa)$. If $(m', \tilde{R}') \neq (m^*, \tilde{R}^*)$ or $\mathcal{A}$'s forgery does not verify, then $\mathcal{B}$ must abort. By the local forking lemma [6], this happens with probability less than $\frac{1}{q}(\mathsf{Adv}^{st\text{-}sec}_{\mathcal{A}}(\kappa, 1))^2$. If $\mathcal{A}$ succeeds having not queried H_{cm} on $(m, \mathcal{S}, R_j)$, then $\mathcal{B}$ aborts. This occurs with probability less than $\frac{q_H}{p}$. Thus,

$$\frac{1}{q}(\mathsf{Adv}^{st\text{-}sec}_{\mathcal{A}}(\kappa, 1))^2 \leq \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\kappa) + \mathsf{negl}(\kappa)$$

9.2 Proof of Adaptive Security for up to $t/2$ Corruptions

Proof. (of Theorem 2.) Let $\mathcal{A}$ be a PPT adversary attempting to break the adaptive unforgeability of $\mathsf{Sparkle}$ (Fig. 4) that makes up to q_H queries to H_{cm} and $\mathsf{H}_{\mathsf{sig}}$, and q_S queries to its signing oracles. Without loss of generality, we assume $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on its forgery $(\tilde{X}, m^*, \tilde{R}^*)$. Then, let $q = q_H + q_S + 1$. We construct a PPT reduction $\mathcal{B}$ against the $t+1$-aomdl assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}^{adp\text{-}sec}_{\mathcal{A}}(\kappa, 1/2) \leq \sqrt{q\mathsf{Adv}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}}(\kappa) + \mathsf{negl}(\kappa)}$$

Here, $\tau = 1/2$ to restrict $\mathcal{A}$ to corrupt $\tau \cdot t = t/2$ parties. The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times. Over the two iterations, $\mathcal{B}$ makes no more than t queries to its discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$ and aims to output $t+1$ discrete logarithms that constitute a valid solution to the AOMDL challenge. If $\mathcal{B}$ makes fewer than t queries while responding to $\mathcal{A}$'s oracle queries, then it makes the additional queries necessary to extract an AOMDL solution. On the second iteration of $\mathcal{A}$, $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{sig}}$ to output a different random value on a single point so that it can extract one of the $t+1$ discrete logarithm solutions from $\mathcal{A}$'s two forgeries. $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{adp\text{-}UF}}_{\mathcal{A}}(\kappa, 1/2)$. However, $\mathcal{B}$ can only extract a solution if $\mathcal{A}$'s forgery $(m^*, (\tilde{R}^*, z^*))$ at the end of each iteration verifies and includes the same nonce $\tilde{R}^*$. By the local forking lemma [6], this occurs with probability less than $\frac{1}{q}(\mathsf{Adv}^{adp\text{-}sec}_{\mathcal{A}}(\kappa, 1/2))^2$.

The Reduction $\mathcal{B}$: We define the reduction $\mathcal{B}$ playing game $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}}(\kappa)$ as follows. $\mathcal{B}$ is responsible for simulating key generation and oracle responses for queries to $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}, \mathcal{O}^{\mathsf{Corrupt}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$. Let $\mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}$ be the set of $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries and their responses, respectively. $\mathcal{B}$ initializes them to the empty set and maintains them across both iterations of the adversary. $\mathcal{B}$ may program the random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$. Let Q be the set of messages that have been queried in $\mathcal{O}^{\mathsf{Sign}}$ as in $\mathsf{Game}^{\mathsf{adp\text{-}UF}}_{\mathcal{A}}(\kappa, 1/2)$. $\mathcal{B}$ initializes Q to the empty set. At the beginning of the second iteration of $\mathcal{A}$, $\mathcal{B}$ resets Q to the empty set.

AOMDL Input. $\mathcal{B}$ 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 $\mathsf{Game}^{t+1\text{-}\mathsf{aomdl}}_{\mathcal{B}}(\kappa)$, $\mathcal{B}$ has access to a discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to t times. $\mathcal{B}$ aims to output $(y_0, \ldots, y_t)$ such that $Y_i = g^{y_i}$ for all $0 \le i \le t$.

Initial Corruption. $\mathcal{B}$ runs $\mathcal{A}()$. $\mathcal{A}$ chooses the total number of potential signers n, the threshold $t+1 \le n$, and the initial set of corrupt parties $\mathsf{cor} \leftarrow \{j\}, |\mathsf{cor}| \le t/2$. $\mathcal{B}$ sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which $\mathcal{B}$ does in the next step.

Simulating KeyGen. $\mathcal{B}$ simulates the key generation algorithm (Fig. 5) using its AOMDL challenge $(Y_0, \ldots, Y_t)$ as follows. For all $1 \le i \le n$, $\mathcal{B}$ sets the public key share as

$$\tilde{X}_i = Y_0 Y_1^i \cdots Y_t^{i^t}$$

which is implicitly equal to $g^{f(i)}$. The joint public key is $\tilde{X} = g^{f(0)} = Y_0$ with corresponding secret key $x = y_0$. $\mathcal{B}$ obtains the initial corrupt secret key shares by querying $x_j = f(j) \leftarrow \mathcal{O}^{\mathsf{dl}}(\tilde{X}_j)$ (with representation $(1, j, \ldots, j^t)$) for all $j \in \mathsf{cor}$. $\mathcal{B}$ runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}'',\mathsf{Corrupt}}}(\tilde{X}, \{\tilde{X}_i\}_{i \in [n]}, \{x_j\}_{j \in \mathsf{cor}})$.

Simulating Random Oracle Queries. $\mathcal{B}$ handles $\mathcal{A}$'s random oracle queries throughout the protocol by lazy sampling, as follows.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on $(m, \mathcal{S}, R)$, $\mathcal{B}$ checks whether $(m, \mathcal{S}, R, \mathsf{cm}) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm. Else, $\mathcal{B}$ samples $\mathsf{cm} \leftarrow\!\!\$\ \mathbb{Z}_p$, appends $(m, \mathcal{S}, R, \mathsf{cm})$ to Q_{cm}, and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\!\$\, \mathbb{Z}_p$, appends (X, m, R, c) to $\mathsf{Q}_{\mathsf{sig}}$, and returns c.

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

Round 1 ($\mathcal{O}^{\mathsf{Sign}}$): In the first round of signing for session ssid, each party i in the signing set $\mathcal{S}$ sends a commitment cm_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}}$ on $(k, \mathsf{ssid}, m, \mathcal{S})$ for honest $k \in \mathsf{hon}$, $\mathcal{B}$ samples $\mathsf{cm}_k \leftarrow\!\!\$\, \mathbb{Z}_p$, appends $(m, \mathcal{S}, \cdot, \mathsf{cm}_k)$ to Q_{cm}, sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, \cdot, \cdot, m, \mathcal{S})$, and returns cm_k to $\mathcal{A}$.

Round 2 ($\mathcal{O}^{\mathsf{Sign}'}$): In the second round of signing for session ssid, each party i in the signing set $\mathcal{S}$ takes as input the set of commitments $\{\mathsf{cm}_i\}_{i \in \mathcal{S}}$ and reveals its nonce R_i such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}'}$ on $(k, \mathsf{ssid}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$ for $k \in \mathsf{hon}$, then for all $j \in \mathcal{S} \cap \mathsf{cor}$, $\mathcal{B}$ looks up cm_j for a record $(m, \mathcal{S}, R_j, \mathsf{cm}_j) \in \mathsf{Q}_{\mathsf{cm}}$.

If there exists $j' \in \mathcal{S}$ such that $(m', \mathcal{S}', \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ and $(m', \mathcal{S}') \neq (m, \mathcal{S})$, or if there exists some $j' \in \mathcal{S}$ for which no record $(m, \mathcal{S}, \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ exists, then $\mathcal{B}$ chooses $R_k \leftarrow g^{r_k}$ randomly, updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$, programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ and returns R_k.

Else if this is the first query in the signing session, $\mathcal{B}$ samples $c \leftarrow\!\!\$\, \mathbb{Z}_p$. For all $k \in \mathsf{hon}$, $\mathcal{B}$ samples $z_k \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $R_k \leftarrow g^{z_k} \tilde{X}_k^{-c\lambda_k}$ (λ_k is the Lagrange coefficient for party k in the set $\mathcal{S}$), and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ (updating $(m, \mathcal{S}, R_k, \mathsf{cm}_k) \in \mathsf{Q}_{\mathsf{cm}}$). Then $\mathcal{B}$ computes $\tilde{R} = \prod_{i \in \mathcal{S}} R_i$ and programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$. (However, if $\mathcal{A}$ has already queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$, then $\mathcal{B}$ aborts.)

Finally, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_k\}_{k \in \mathcal{S}})$, and returns R_k to $\mathcal{A}$. If this is not the first query, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_k\}_{k \in \mathcal{S}})$ and returns R_k.

Round 3 ($\mathcal{O}^{\mathsf{Sign}''}$): In the third round of signing for session ssid, each party i in the signing set $\mathcal{S}$ produces a partial signature on the message m. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}''}$ on $(k, \mathsf{ssid}, \{R_i\}_{i \in \mathcal{S}})$ for $k \in \mathsf{hon}$, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$, checks whether $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ for all $i \in \mathcal{S}$ and returns $\perp$ if not. If $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ but $\mathcal{A}$ never queried H_{cm} on input $(m, \mathcal{S}, R_i)$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow ()$ and returns z_k.

Simulating Corruption Queries ($\mathcal{O}^{\mathsf{Corrupt}}$): $\mathcal{A}$ may at any time corrupt an honest party k by querying $\mathcal{O}^{\mathsf{Corrupt}}(k)$. Upon receiving a corruption query, $\mathcal{B}$ first checks that $k \in \mathsf{hon}$, returning $\perp$ if not. Otherwise, $\mathcal{B}$ queries its DL oracle $\mathcal{O}^{\mathsf{dl}}$ on $\tilde{X}_k = g^{f(k)}$ (with representation $(1, k, \ldots, k^t)$) to obtain the secret key $x_k = f(k)$. Then, for each $\mathsf{st}_{k,\mathsf{ssid}}$, $\mathcal{B}$ does the following:

– If $\mathsf{st}_{k,\mathsf{ssid}} = (\mathsf{cm}_k, \cdot, \cdot, m, \mathcal{S})$, then $\mathcal{B}$ chooses $r_k \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $R_k \leftarrow g^{r_k}$, and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$. It then updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$.
– If $\mathsf{st}_{k,\mathsf{ssid}} = (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$, then $\mathcal{B}$ computes $r_k = z_k - c\lambda_k x_k$ (now that it knows x_k) and updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$.

– If $\mathsf{st}_{k,\mathsf{ssid}} = ()$, then return $()$. This case occurs if signing session ssid has already been completed (i.e., a valid signature was issued).

Finally, $\mathcal{B}$ sets $\mathsf{hon} \leftarrow \mathsf{hon} \setminus \{k\}$ and $\mathsf{cor} \leftarrow \mathsf{cor} \cup \{k\}$ and returns $x_k, \{\mathsf{st}_{k,\ell}\}_{\ell \in [\mathsf{ssid}]}$ to $\mathcal{A}$.

Output. At the end of the game, $\mathcal{A}$ produces a forgery $(m^*, \sigma^*) = (m^*, (\tilde{R}^*, z^*))$ and wins if $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1, m^* \notin Q$, and $|\mathsf{cor}| \leq t/2$.

Extracting the Discrete Logarithm of Y_0. $\mathcal{B}$'s simulation of key generation and signing is perfect, and $\mathcal{B}$ aborts with negligible probability. Indeed, $\mathcal{B}$ aborts in Round 3 if $\mathcal{A}$ reveals R_j such that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_j)$ but $\mathcal{A}$ never queried H_{cm} on $(m, \mathcal{S}, R_j)$. This requires $\mathcal{A}$ to have guessed cm_j ahead of time, which occurs with negligible probability $1/p$.

$\mathcal{B}_2$ also aborts in Round 2 if $\mathcal{A}$ had previously queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$. In that case, $\mathcal{B}$ had returned a random $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_k before $\mathcal{B}$ revealed it, which occurs with negligible probability $1/p$.

Next, we show that $\mathcal{B}$ can extract the discrete logarithm of Y_0 from $\mathcal{A}$'s two valid forgeries. $\mathcal{A}$'s first forgery $(m^*, (\tilde{R}^*, z^*))$ satisfies $\tilde{R}^* \tilde{X}^{c^*} = g^{z^*}$, where $c^* = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$. Here, z^* does not suffice for $\mathcal{B}$ to extract the discrete logarithm of $\tilde{X} = Y_0$ because it does not necessarily know the discrete logarithm of $\tilde{R}^*$. Thus, $\mathcal{B}$ chooses $c' \leftarrow\!\!\$\, \mathbb{Z}_p$ and programs $\mathsf{H}_{\mathsf{sig}}$ to output c' on input $(\tilde{X}, m^*, \tilde{R}^*)$. $\mathcal{B}$ resets Q to the empty set, but the sets Q_{cm}, $\mathsf{Q}_{\mathsf{sig}}$ are kept for the second iteration of the adversary. $\mathcal{B}$ then runs $\mathcal{A}$ again on the same random coins.

After the second iteration, suppose $\mathcal{A}$ terminates with $(m', (\tilde{R}', z'))$. If $(m', \tilde{R}') = (m^*, \tilde{R}^*)$ and $\mathcal{A}$'s forgeries both verify, then $\mathcal{B}$ returns $y_0 = x = \frac{z^* - z'}{c^* - c'}$ such that $Y_0 = \tilde{X} = g^x$. If $(m', \tilde{R}') \neq (m^*, \tilde{R}^*)$ or $\mathcal{A}$'s forgery does not verify, then $\mathcal{B}$ must abort. By the local forking lemma [6], this happens with probability less than $\frac{1}{q}(\mathsf{Adv}^{adp\text{-}sec}_{\mathcal{A}}(\kappa, 2))^2$. If $\mathcal{A}$ succeeds having not queried H_{cm} on $(m, \mathcal{S}, R_j)$, then $\mathcal{B}$ aborts. This occurs with probability less than $\frac{q_H}{p}$. Thus,

$$\frac{1}{q}(\mathsf{Adv}^{adp\text{-}sec}_{\mathcal{A}}(\kappa, 2))^2 \leq \Pr[\mathcal{B} \text{ extracts } y_0] + \mathsf{negl}(\kappa)$$

If $\mathcal{B}$ extracts y_0, then we use this to extract a full AOMDL solution as follows.

Extracting an AOMDL Solution. The reduction $\mathcal{B}$ 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, $\mathcal{B}$ can corrupt the remaining number at the end, by querying its DL oracle until it reaches t secret keys.) Recall that $\mathcal{B}$ set $\tilde{X}_i = Y_0 Y_1^i \cdots Y_t^{i^t}$

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 $\tilde{X} = Y_0$. 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,

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

$\mathcal{B}$ 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, $\mathcal{B}$ can solve for $(y_0, y_1, \ldots, y_t)$ and win the $t+1$-aomdl game.

9.3 Proof of Adaptive Security for up to t Corruptions

Proof. (of Theorem 3.) Let $\mathcal{A}$ be an algebraic adversary attempting to break the adaptive unforgeability of $\mathsf{Sparkle}$ (Fig. 4). We construct a PPT reduction $\mathcal{B}$ against the $t+1$-aomdl assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}_{\mathcal{A}}^{adp\text{-}sec}(\kappa, 1) \leq \mathsf{Adv}_{\mathcal{B}}^{t\,+\,1\text{-}\mathsf{aomdl}}(\kappa) + \mathsf{negl}(\kappa)$$

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

The Reduction $\mathcal{B}$: We define the reduction $\mathcal{B}$ playing game $\mathsf{Game}_{\mathcal{B}}^{t+1\text{-}\mathsf{aomdl}}(\kappa)$ as follows. $\mathcal{B}$ is responsible for simulating key generation and oracle responses for queries to $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}, \mathcal{O}^{\mathsf{Corrupt}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$. Let $\mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}$ be the set of $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries and their responses, respectively. $\mathcal{B}$ may program the random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$. Let Q be the set of messages that have been queried to $\mathcal{O}^{\mathsf{Sign}}$ as in game $\mathsf{Game}_{\mathcal{A}}^{\mathsf{adp\text{-}UF}}(\kappa, 1)$. $\mathcal{B}$ initializes $\mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}, Q$ to the empty set.

AOMDL Input. $\mathcal{B}$ 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 $\mathsf{Game}_{\mathcal{B}}^{t+1\text{-}\mathsf{aomdl}}(\kappa)$, $\mathcal{B}$ has access to a discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to t times. $\mathcal{B}$ aims to output $(y_0, \ldots, y_t)$ such that $Y_i = g^{y_i}$ for all $i \in [0..t]$.

Initial Corruption. $\mathcal{B}$ runs $\mathcal{A}()$. $\mathcal{A}$ chooses the total number of potential signers n, the threshold $t+1 \leq n$, and the initial set of corrupt parties $\mathsf{cor} \leftarrow \{j\}, |\mathsf{cor}| \leq t$. $\mathcal{B}$ sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which $\mathcal{B}$ does in the next step.

Simulating KeyGen. $\mathcal{B}$ simulates the key generation algorithm (Fig. 5) using its AOMDL challenge $(Y_0, \ldots, Y_t)$ as follows. For all $i \in [n]$, $\mathcal{B}$ sets the public key share as

$$\tilde{X}_i = Y_0 Y_1^i \cdots Y_t^{i^t}$$

which is implicitly equal to $g^{f(i)}$. The joint public key is $\tilde{X} = g^{f(0)} = Y_0$ with corresponding secret key $x = y_0$. $\mathcal{B}$ obtains the initial corrupt secret key shares by querying $x_j = f(j) \leftarrow \mathcal{O}^{\mathsf{dl}}(\tilde{X}_j)$ (with representation $(1, j, \ldots, j^t)$) for all $j \in \mathsf{cor}$. $\mathcal{B}$ runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'', \mathsf{Corrupt}}}(\tilde{X}, \{\tilde{X}_i\}_{i \in [n]}, \{x_j\}_{j \in \mathsf{cor}})$.

Simulating Random Oracle Queries. $\mathcal{B}$ handles $\mathcal{A}$'s random oracle queries throughout the protocol by lazy sampling, as follows.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on $(m, \mathcal{S}, R)$, $\mathcal{B}$ checks whether $(m, \mathcal{S}, R, \mathsf{cm}) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm. Else, $\mathcal{B}$ samples $\mathsf{cm} \leftarrow\!\!\$\ \mathbb{Z}_p$, appends $(m, \mathcal{S}, R, \mathsf{cm})$ to Q_{cm}, and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (X, m, R, c) to $\mathsf{Q}_{\mathsf{sig}}$, and returns c.

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

Round 1 ($\mathcal{O}^{\mathsf{Sign}}$): In the first round of signing for session ssid, each party i in the signing set $\mathcal{S}$ sends a commitment cm_i. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}}$ on $(k, \mathsf{ssid}, m, \mathcal{S})$ for honest $k \in \mathsf{hon}$, $\mathcal{B}$ samples $\mathsf{cm}_k \leftarrow\!\!\$\ \mathbb{Z}_p$, appends $(m, \mathcal{S}, \cdot, \mathsf{cm}_k)$ to Q_{cm}, sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, \cdot, \cdot, m, \mathcal{S})$, and returns cm_k.

Round 2 ($\mathcal{O}^{\mathsf{Sign}'}$): In the second round of signing for session ssid, each party i in the signing set $\mathcal{S}$ takes as input the set of commitments $\{\mathsf{cm}_i\}_{i \in \mathcal{S}}$ and reveals its nonce R_i such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}'}$ on $(k, \mathsf{ssid}, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$ for $k \in \mathsf{hon}$, then for all $j \in \mathcal{S} \cap \mathsf{cor}$, $\mathcal{B}$ looks up cm_j for a record $(m, \mathcal{S}, R_j, \mathsf{cm}_j) \in \mathsf{Q}_{\mathsf{cm}}$.

If there exists $j' \in \mathcal{S}$ such that $(m', \mathcal{S}', \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ and $(m', \mathcal{S}') \neq (m, \mathcal{S})$, or if there exists some $j' \in \mathcal{S}$ for which no record $(m, \mathcal{S}, \cdot, \mathsf{cm}_{j'}) \in \mathsf{Q}_{\mathsf{cm}}$ exists, then

$\mathcal{B}$ chooses $R_k \leftarrow\!\!\$\; g^{r_k}$ randomly, updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$, programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$, and returns R_k.

Else if this is the first query in the signing session, $\mathcal{B}$ samples $c \leftarrow\!\!\$\; \mathbb{Z}_p$. For all $k \in \mathsf{hon}$, $\mathcal{B}$ samples $z_k \leftarrow\!\!\$\; \mathbb{Z}_p$, computes $R_k \leftarrow g^{z_k} \tilde{X}_k^{-c\lambda_k}$ (λ_k is the Lagrange coefficient for party k in the set $\mathcal{S}$), and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$ (updating $(m, \mathcal{S}, R_k, \mathsf{cm}_k) \in \mathsf{Q}_{\mathsf{cm}}$). Then $\mathcal{B}$ computes $\tilde{R} = \prod_{i \in \mathcal{S}} R_i$ and programs $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$. (However, if $\mathcal{A}$ has already queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$, then $\mathcal{B}$ aborts.)

Finally, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$, and returns R_k to $\mathcal{A}$. If this is not the first query, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$ and returns R_k.

Round 3 ($\mathcal{O}^{\mathsf{Sign}''}$): In the third round of signing for session ssid, each party i in the signing set $\mathcal{S}$ produces a partial signature on the message m. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}''}$ on $(k, \mathsf{ssid}, \{R_i\}_{i \in \mathcal{S}})$ for $k \in \mathsf{hon}$, $\mathcal{B}$ looks up $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$, checks whether $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ for all $i \in \mathcal{S}$ and returns $\perp$ if not. If $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_i)$ but $\mathcal{A}$ never queried H_{cm} on input $(m, \mathcal{S}, R_i)$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ sets $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow ()$, and returns z_k.

Simulating Corruption Queries ($\mathcal{O}^{\mathsf{Corrupt}}$): $\mathcal{A}$ may at any time corrupt an honest party k by querying $\mathcal{O}^{\mathsf{Corrupt}}(k)$. Upon receiving a corruption query, $\mathcal{B}$ first checks that $k \in \mathsf{hon}$, returning $\perp$ if not. Otherwise, $\mathcal{B}$ queries its DL oracle $\mathcal{O}^{\mathsf{dl}}$ on $\tilde{X}_k = g^{f(k)}$ (with representation $(1, k, \ldots, k^t)$) to obtain the secret key $x_k = f(k)$. Then, for each $\mathsf{st}_{k,\mathsf{ssid}}$, $\mathcal{B}$ does the following:

- If $\mathsf{st}_{k,\mathsf{ssid}} = (\mathsf{cm}_k, \cdot, \cdot, m, \mathcal{S})$, then $\mathcal{B}$ chooses $r_k \leftarrow\!\!\$\; \mathbb{Z}_p$, sets $R_k \leftarrow g^{r_k}$, and programs $\mathsf{cm}_k \leftarrow \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_k)$. It then updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S})$.
- If $\mathsf{st}_{k,\mathsf{ssid}} = (\mathsf{cm}_k, R_k, z_k, c\lambda_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$, then $\mathcal{B}$ computes $r_k = z_k - c\lambda_k x_k$ (now that it knows x_k) and updates $\mathsf{st}_{k,\mathsf{ssid}} \leftarrow (\mathsf{cm}_k, R_k, r_k, m, \mathcal{S}, \{\mathsf{cm}_i\}_{i \in \mathcal{S}})$.
- If $\mathsf{st}_{k,\mathsf{ssid}} = ()$, then return $()$. This case occurs if signing session ssid has already been completed (i.e., a valid signature was issued).

Finally, $\mathcal{B}$ sets $\mathsf{hon} \leftarrow \mathsf{hon} \setminus \{k\}$ and $\mathsf{cor} \leftarrow \mathsf{cor} \cup \{k\}$ and returns $x_k, \{\mathsf{st}_{k,\ell}\}_{\ell \in [\mathsf{ssid}]}$ to $\mathcal{A}$.

Output. At the end of the game, $\mathcal{A}$ produces a forgery $(m^*, \sigma^*) = (m^*, (\tilde{R}^*, z^*))$ and wins if $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, $m^* \notin Q$, and $|\mathsf{cor}| \leq t$.

Extracting an AOMDL Solution. $\mathcal{B}$'s simulation of key generation and signing is perfect, and $\mathcal{B}$ aborts with negligible probability. Indeed, $\mathcal{B}$ aborts in Round 3 if $\mathcal{A}$ reveals R_j such that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(m, \mathcal{S}, R_j)$ but $\mathcal{A}$ never queried H_{cm} on $(m, \mathcal{S}, R_j)$. This requires $\mathcal{A}$ to have guessed cm_j ahead of time, which occurs with negligible probability $1/p$.

$\mathcal{B}_2$ also aborts in Round 2 if $\mathcal{A}$ had previously queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$. In that case, $\mathcal{B}$ had returned a random $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_k before $\mathcal{B}$ revealed it, which occurs with negligible probability $1/p$.

It remains to show that $\mathcal{B}$ can extract an AOMDL solution from $\mathcal{A}$'s output. Assume without loss of generality that $\mathcal{A}$ makes t corruptions over the course of the protocol. (If not, $\mathcal{B}$ can corrupt the remaining number at the end, by querying its DL oracle until it reaches t secret keys.) Recall that $\mathcal{B}$ set $\tilde{X}_i = Y_0 Y_1^i \cdots Y_t^{i^t}$ and made t DL queries $g^{x_{i_1}}, \dots, g^{x_{i_t}}$:

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

Recall also that $\tilde{X} = Y_0$. 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,

$$
\begin{pmatrix} x \\ x_{i_1} \\ \vdots \\ x_{i_t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & i_1 & \cdots & {i_1}^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_t & \cdots & {i_t}^t \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{pmatrix} \tag{2}
$$

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

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

$$
\tilde{R}^* = g^{z^*} \tilde{X}^{-c^*} \tag{3}
$$

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

$$
\tilde{R}^* = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} R_{i,1}^{\rho_{i,1}^*} \cdots R_{i,n}^{\rho_{i,n}^*}
$$

Each $R_{i,k}$ verifies as $R_{i,k} = g^{z_{i,k}} \tilde{X}_k^{-c_i \lambda_{i,k}}$, where $c_i = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m_i, \tilde{R}_i)$. Thus,

$$\tilde{R}^* = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (g^{z_{i,1}} \tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (g^{z_{i,n}} \tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*}$$

Equating this with Eq. (3), we have:

$$g^{z^*} \tilde{X}^{-c^*} = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (g^{z_{i,1}} \tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (g^{z_{i,n}} \tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*}$$

Rearranging, we have:

$$\begin{aligned} &g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1} \rho_{i,1}^*} \cdots g^{-z_{i,n} \rho_{i,n}^*} \\ &\qquad = \tilde{X}^{c^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (\tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (\tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*} \end{aligned} \tag{4}$$

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. (4) can be rewritten as:

$$g^{\eta^*} = \tilde{X}^{c^* + \xi^*} \tilde{X}_1^{\zeta_1^*} \cdots \tilde{X}_n^{\zeta_n^*} \tag{5}$$

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

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

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 $\tilde{X}_k$ can be rewritten as:

$$\tilde{X}_k = \tilde{X}^{\mu_k^*} g^{\nu_k^*}$$

and Eq. (5) can be rewritten as:

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

Rearranging, we have:

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

and

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

$\mathcal{A}$ fixed $\tilde{R}^*$ and thus $\eta^*, \{\nu_i^*\}_{i\in[n]}, \xi^*, \{\mu_i^*\}_{i\in[n]}$ as it queried $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$ to receive random c^*. Thus, the denominator is nonzero with overwhelming probability and $\mathcal{B}$ 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 2 is a Vandermonde matrix and is therefore invertible. Thus, $\mathcal{B}$ can solve for $(y_0, y_1, \ldots, y_t)$ and win the $t+1$-aomdl game.

Acknowledgements. Elizabeth Crites was supported by Input Output through their funding of the Blockchain Technology Lab at the University of Edinburgh.

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] 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`.

[4] 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.

[5] M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. "Better than Advertised Security for Non-interactive Threshold Signatures". In: *CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022*. Ed. by Y. Dodis and T. Shrimpton. Vol. 13510. LNCS. Springer, 2022, pp. 517–550.

[6] 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.

[7] 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.

[8] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. "On the (in)security of ROS". In: *EUROCRYPT 2021, Zagreb, Croatia, October 17-21, 2021*. Ed. by A. Canteaut and F. Standaert. Vol. 12696. LNCS. Springer, 2021, pp. 33–53.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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`.

[16] 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.

[17] 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/`.

[18] E. Crites, C. Komlo, and M. Maller. *Fully Adaptive Schnorr Threshold Signatures.* Cryptology ePrint Archive, Paper 2023/445. 2023. URL: `https://eprint.iacr.org/2023/445`.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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`.

[36] 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.

[37] 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`.

[38] 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.

[39] 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.

[40] 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/`.

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

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

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

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

A Proof of Static Security in the AGM

We additionally provide a proof of static security for $\mathsf{Sparkle}$ under the discrete logarithm (DL) assumption in the algebraic group model (AGM) and random oracle model (ROM) without any tightness loss. This security reduction serves as a warm-up to the proof of full adaptive security in Section 8.

Theorem 4. $\mathsf{Sparkle}$ *is statically secure under DL in the AGM+ROM.*

Proof. Let $\mathcal{A}$ be an algebraic adversary attempting to break the static unforgeability of $\mathsf{Sparkle}$ (Fig. 4) We construct a PPT reduction $\mathcal{B}$ against the DL assumption (Fig. 3) that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}_{\mathcal{A}}^{st\text{-}sec}(\kappa, 1) \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{dl}}(\kappa) + \mathsf{negl}(\kappa)$$

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

The Reduction $\mathcal{B}$: We define the reduction $\mathcal{B}$ playing game $\mathsf{Game}_{\mathcal{B}}^{\mathsf{dl}}(\kappa)$ as follows. $\mathcal{B}$ is responsible for simulating key generation and oracle responses for queries to $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$. Let $\mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}$ be the set of $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries and their responses, respectively. $\mathcal{B}$ may program the random oracles $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$. Let Q be the set of messages that have been queried in $\mathcal{O}^{\mathsf{Sign}}$ as in $\mathsf{Game}_{\mathcal{A}}^{\mathsf{UF}}(\kappa, 1)$. $\mathcal{B}$ initializes $\mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}, Q$ to the empty set.

DL Input. $\mathcal{B}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and a DL challenge $\dot{X}$. $\mathcal{B}$ aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$.

Static Corruption. $\mathcal{B}$ runs $\mathcal{A}()$. $\mathcal{A}$ 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. $\mathcal{B}$ sets $\mathsf{hon} \leftarrow [n] \setminus \mathsf{cor}$ and must reveal the secret keys of the corrupt parties to $\mathcal{A}$, which $\mathcal{B}$ does in the next step.

Simulating KeyGen. $\mathcal{B}$ simulates the key generation algorithm (Fig. 5) using its DL challenge $\dot{X}$ as follows.

1. $\mathcal{B}$ sets the joint public key $\tilde{X} \leftarrow \dot{X}$.
2. $\mathcal{B}$ simulates a Shamir secret sharing of the discrete logarithm of $\tilde{X}$ by performing the following steps. (See Section 3 for notation.) Assume without loss of generality that $|\mathsf{cor}| = t$.
   (a) $\mathcal{B}$ 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 $j \in \mathsf{cor}$. $\mathcal{B}$ 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 $1 \leq i \leq t$, $\mathcal{B}$ computes
$$Y_i = \tilde{X}^{L'_{0,i}} \prod_{j \in \mathsf{cor}} g^{x_j L'_{j,i}} \tag{6}$$
   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 $1 \leq i \leq n$, $\mathcal{B}$ computes
$$\tilde{X}_i = \tilde{X} Y_1^i Y_2^{i^2} \cdots Y_t^{i^t} \tag{7}$$
   which is implicitly equal to $g^{f(i)}$.

The joint public key is $\tilde{X} = g^{f(0)} = \dot{X}$ with corresponding secret key $\dot{x}$. $\mathcal{B}$ runs $\mathcal{A}^{\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}''}}(\tilde{X}, \{\tilde{X}_i\}_{i \in [n]}, \{x_j\}_{j \in \mathsf{cor}})$.

Simulating Random Oracle and Signing Queries. $\mathcal{B}$ simulates random oracle and signing queries as in the static proof under DL+ROM (Section 9.1).

Output. At the end of the game, $\mathcal{A}$ produces a forgery $(m^*, \sigma^*) = (m^*, (\tilde{R}^*, z^*))$ and wins if $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$ and $m^* \notin Q$.

Extracting the Discrete Logarithm of $\dot{X}$. $\mathcal{A}$'s forgery verifies as:
$$\tilde{R}^* = g^{z^*} \tilde{X}^{-c^*} \tag{8}$$
where $c^* = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$. On the other hand, when $\mathcal{A}$ made its query $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$, it provided a representation of $\tilde{R}^*$ in terms of all of the group elements it had seen so far, namely $(g, \tilde{X}, \tilde{X}_1, \ldots, \tilde{X}_n, \{R_{i,1}, \ldots, R_{i,n}\}_{i \in [q_S]})$, where $\{R_{i,k}\}_{i \in [q_S]}$ are the honest nonces returned by the Sign' oracle over the q_S signing queries that $\mathcal{A}$ makes. We assume without loss of generality that $\mathcal{A}$ completes every signing session. (Otherwise, $\mathcal{B}$ can perform any unmet Sign' and Sign'' 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:
$$\tilde{R}^* = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} R_{i,1}^{\rho_{i,1}^*} \cdots R_{i,n}^{\rho_{i,n}^*}$$

Each $R_{i,k}$ verifies as $R_{i,k} = g^{z_{i,k}} \tilde{X}_k^{-c_i \lambda_{i,k}}$, where $c_i = \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m_i, \tilde{R}_i)$. Thus,

$$\tilde{R}^* = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (g^{z_{i,1}} \tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (g^{z_{i,n}} \tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*}$$

Equating this with Eq. (8), we have:

$$g^{z^*} \tilde{X}^{-c^*} = g^{\gamma^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (g^{z_{i,1}} \tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (g^{z_{i,n}} \tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*}$$

Rearranging, we have:

$$\begin{aligned} g^{z^*} g^{-\gamma^*} \prod_{i=1}^{q_S} g^{-z_{i,1}\rho_{i,1}^*} \cdots g^{-z_{i,n}\rho_{i,n}^*} \\ = \tilde{X}^{c^*} \tilde{X}^{\xi^*} \tilde{X}_1^{\xi_1^*} \cdots \tilde{X}_n^{\xi_n^*} \prod_{i=1}^{q_S} (\tilde{X}_1^{-c_i \lambda_{i,1}})^{\rho_{i,1}^*} \cdots (\tilde{X}_n^{-c_i \lambda_{i,n}})^{\rho_{i,n}^*} \end{aligned} \quad (9)$$

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. (9) can be rewritten as:

$$g^{\eta^*} = \tilde{X}^{c^* + \xi^*} \tilde{X}_1^{\zeta_1^*} \cdots \tilde{X}_n^{\zeta_n^*} \quad (10)$$

Recall that $\tilde{X}_i = \tilde{X} Y_1^i Y_2^{i^2} \cdots Y_t^{i^t}$ for all $i \in [n]$ (Eq. (7)) and that $Y_i = \tilde{X}^{L'_{0,i}} \prod_{j \in \mathsf{cor}} g^{x_j L'_{j,i}}$ for all $i \in [t]$ (Eq. (6)). Thus,

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

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 $\tilde{X}_k$ can be rewritten as:

$$\tilde{X}_k = \tilde{X}^{\mu_k^*} g^{\nu_k^*}$$

and Eq. (10) can be rewritten as:

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

Rearranging, we have:

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

and

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

$\mathcal{A}$ fixed $\tilde{R}^*$ and thus $\eta^*, \{\nu_k^*\}_{k\in[n]}, \xi^*, \{\mu_k^*\}_{k\in[n]}$ as it queried $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)$ to receive random c^*. Thus, the denominator is nonzero with overwhelming probability and $\mathcal{B}$ can solve for $\dot{x}$ to win the DL game.