On the Adaptive Security of FROST

Elizabeth Crites[1], Jonathan Katz[2], Chelsea Komlo[3], Stefano Tessaro[4], and Chenzhi Zhu[4]

[1] Web3 Foundation
[2] Google
[3] University of Waterloo & NEAR One
[4] University of Washington

Abstract. FROST and its variants are state-of-the-art protocols for threshold Schnorr signatures that are used in real-world applications. While *static* security of these protocols has been shown by several works, the security of these protocols under *adaptive* corruptions—where an adversary can choose which parties to corrupt at any time based on information it learns during protocol executions—has remained a notorious open problem that has received renewed attention due to recent standardization efforts for threshold schemes.

We show adaptive security (without erasures) of FROST and several variants under different corruption thresholds and computational assumptions. Let n be the total number of parties, $t+1$ the signing threshold, and t_c an upper bound on the number of corrupted parties.

1. We prove adaptive security when $t_c = t/2$ in the random-oracle model (ROM) based on the algebraic one-more discrete-logarithm assumption (AOMDL)—the same conditions under which FROST has been proven statically secure.
2. We introduce the *low-dimensional vector representation* (LDVR) problem, parameterized by t_c, t, and n, and prove adaptive security in the algebraic-group model (AGM) and ROM based on the AOMDL assumption and the hardness of the LDVR problem for the corresponding parameters. In some regimes (including some $t_c > t/2$), we show the LDVR problem is unconditionally hard, while in other regimes (in particular, when $t_c = t$), we show that hardness of the LDVR problem is necessary for adaptive security to hold. In fact, we show that hardness of the LDVR problem is necessary for proving adaptive security of a broad class of threshold Schnorr signatures.

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

elizabeth@web3.foundation
jkatz2@gmail.com
ckomlo@uwaterloo.ca
tessaro@cs.washington.edu
zhucz20@cs.washington.edu

Table of Contents

1 Introduction

A $(t+1, n)$-threshold signature scheme allows a signing key to be distributed among n parties such that any $t+1$ of those parties can jointly generate a signature on a message, while an adversary corrupting up to some specified number $t_c \leq t$ of those parties is unable to do so. Motivated primarily by blockchain applications, several works have considered threshold protocols for ECDSA, Schnorr, and BLS signatures in recent years.

FROST and its variants [30, 22, 12, 9, 38, 17] (hereinafter collectively referred to as FROST) represent state-of-the-art protocols for threshold Schnorr signatures; a version has been codified in an IETF RFC [19] and is currently used in real-world applications [28]. Security of FROST in a *static* corruption model—where the adversary decides at the outset which parties to corrupt—has been shown to hold in the works cited above. However, proving security of these protocols under *adaptive* corruptions—where the adversary can choose which parties to corrupt at any time based on information it learns from executions of the protocol—has remained a challenging open problem.[1] This problem has become more pressing with the recent NIST call for threshold protocols [13, 14], which highlights adaptive security (for up to $n = 1024$ parties) as a consideration. We remark that while there has been recent work designing other threshold Schnorr protocols with adaptive security, those schemes are not currently deployed, are less efficient than FROST, and have other disadvantages (see Section 1.1 for further discussion); therefore, it is still important to understand the security of FROST itself.

In this work, we show several results regarding the adaptive security of FROST. First, we prove adaptive security at half the maximal corruption threshold (i.e., when $t_c = t/2$) in the random-oracle model (ROM) based on the algebraic one-more discrete-logarithm assumption (AOMDL)—the same conditions under which FROST is known to be statically secure. Our proof methodology here is similar to the one used to analyze Sparkle+ [23]: we invoke the forking lemma and exploit the fact that the total number of corruptions in both the "original" execution and the "rewound" execution is bounded by t. While this result holds only for a sub-optimal corruption threshold, it nevertheless fills an important gap in the literature.[2]

We also prove adaptive security of FROST (under stronger conditions) for t_c beyond $t/2$. Inspired by recent work [21] proposing an adaptive strategy for attacking a broad class[3] of threshold Schnorr protocols (including FROST),

[1] Note that adaptive security implies static security. It is also trivial to show (by simply guessing in advance the parties to be corrupted) that any statically secure scheme is adaptively secure with a multiplicative security loss of $\binom{n}{t_c}$. When n, t_c are large, however, this term is super-polynomial.

[2] Note that the NIST call for threshold protocols [13, 14] explicitly allows for consideration of sub-optimal corruption thresholds, especially when there are other advantages (e.g., in terms of required assumptions or efficiency).

[3] The attack applies to any scheme where the ith party holds a Shamir share sk_i of the secret key and the "commitment" g^{sk_i} can be learned by an adversary. See Section 4.2.

we introduce a problem that we call the *low-dimensional vector representation* (LDVR) problem, parameterized by t_c, t, and n. If the LDVR problem is easy for particular values of t_c, t, n, then we show there is an explicit (adaptive) adversary corrupting t_c parties who can violate the security of FROST with threshold $t+1$; in other words, hardness of the LDVR problem is *necessary* for adaptive security of FROST to possibly hold. Complementing this result, we show that hardness of the LDVR problem for particular values of t_c, t, n is sufficient to prove security of FROST (with signing threshold $t+1$) for up to t_c adaptive corruptions, under the same conditions as above but now also in the algebraic-group model (AGM) [26]. (We remark that, prior to our work, no pairing-free, two-round threshold signature scheme with adaptive security was known, even assuming the AGM.) Roughly speaking, then, we isolate the LDVR assumption and show that its hardness is tightly connected to the adaptive security of FROST.

Because of the centrality of the LDVR problem to the adaptive security of FROST, we initiate study of the hardness of this problem. For some t_c, t, n—including some values of t_c beyond $t/2$—we show that the LDVR problem is *unconditionally* hard. In other regimes, (computational) hardness of the LDVR problem must be assumed. We justify the assumption in that case by showing that some natural approaches to solving the problem fail.

1.1 Related Work

Prior works have investigated the adaptive security of threshold protocols for Schnorr [1, 34, 23, 5, 3, 29], BLS [4, 24], ECDSA [16, 32], RSA [2], and other signature schemes [6, 7]. In this work, we focus on threshold Schnorr signatures; all the schemes discussed are in this setting as well.

Abe and Fehr [1] give an adaptively secure protocol with maximal corruption threshold (i.e., $t_c = t$) based on the decisional Diffie-Hellman (DDH) assumption (and security of Schnorr signatures). Their scheme has at least three rounds, relies on strong network assumptions (synchronous broadcast), and requires an honest majority (i.e., $t_c < n/2$). FROST tolerates a dishonest majority and works in an asynchronous network, without broadcast or even authenticated channels.

Crites et al. [23] introduce Sparkle+ and show that it is adaptively secure for $t/2$ corruptions based on the AOMDL assumption in the ROM, and adaptively secure for t corruptions based on the AOMDL assumption in the AGM+ROM.[4] Like FROST, Sparkle+ allows for pre-processing before the message or signing set is known, but requires two online signing rounds and authenticated channels.

Zero S. [34] is adaptively secure for t corruptions in the *erasure* model, where honest parties are allowed to securely erase state. It is proven to UC-realize the threshold signature functionality [15] in the global ROM, assuming the hardness of the discrete-logarithm problem. Zero S. is a three-round protocol and makes use of online-extractable zero-knowledge proofs [25], which adds overhead.

[4] The work cited previously [21] shows that full adaptive security of Sparkle+ (among other schemes) cannot be proven without an assumption that implies the hardness of some instances of a search problem P introduced in their work. We similarly show this for the LDVR problem.

HARTS [5] is a constant-round scheme achieving robustness and adaptive security in the asynchronous setting. However, their security proof only considers the super honest-majority setting (i.e., $t_c < n/3$), and the protocol has high round complexity.

Katsumata et al. [29] prove adaptive security of two schemes, Crackle and Snap, for t corruptions. These schemes do not have identifiable abort, whereas FROST does [38]. Moreover, the schemes require five and four signing rounds, respectively, as well as authenticated channels.

Crites et al. [20] prove a series of impossibility results for adaptive security of threshold signature schemes with a certain *key-uniqueness* property. In particular, they show that key-unique threshold Schnorr signature schemes cannot be proven adaptively secure beyond $t/2$ corruptions under the (A)OMDL assumption in the programmable ROM using a fully black-box reduction that rewinds the adversary and programs at least one random-oracle query. This is exactly the type of reduction we employ in our adaptive security proof against at most $t/2$ corruptions, and so aligns with their result.

Glacius [3] is a five-round protocol that is proven adaptively secure for t corruptions based on DDH in the ROM, assuming authenticated channels. Interestingly, the key shares (and commitments thereto) have a different structure than what is required for the aforementioned attack [21] and impossibility result [20].

Relationship to [21]. Our work was inspired by the recent work of Crites and Stewart [21] showing an adaptive strategy for attacking a class of threshold Schnorr protocols including FROST (cf. footnote 3). Their work can be viewed as showing an *efficient* attack given access to an oracle solving a particular computational problem P whose hardness is unclear. The main message of their work is that an efficient algorithm for solving P would give an efficient adaptive attack on FROST, in particular, and thus show an example of a natural protocol that is statically secure but not adaptively secure.

In contrast, our focus is on proving adaptive security of FROST under well-defined assumptions. (We stress that Crites and Stewart are silent about whether assumed hardness of problem P could be used to prove adaptive security of FROST.) To do so, we formalize the LDVR problem that is related to problem P, and show that the LDVR problem is sufficient (along with other assumptions) and necessary to prove adaptive security of FROST. We also analyze the hardness of the LDVR problem, and show that it is unconditionally hard in certain parameter regimes. See Section 4.1 for further discussion.

1.2 Future Work

Hardness of the LDVR problem (for parameters where our unconditional hardness result does not hold) is unclear, and requires further investigation. Note that any $o(\sqrt{p})$-time algorithm solving the LDVR problem (where p is the order of the underlying group) would be interesting, as it would imply an attack on FROST better than what is currently known using generic discrte-logarithm algorithms. Beyond the practical implications for FROST, such a result would

also be interesting insofar as it would show a natural example of a protocol where there is a gap between the known advantages of adaptive and static attackers.

2 Preliminaries

We let $\kappa \in \mathbb{N}$ denote the security parameter. We let $x \leftarrow\!\!\$\, S$ denote sampling a uniform element of a set S and assigning it to x. We use $[n]$ to represent $\{1, \ldots, n\}$ and $[i..j]$ to represent $\{i, \ldots, j\}$. We represent vectors as $\vec{a} = (a_1, \ldots)$.

PPT stands for "probabilistic polynomial time." Algorithms are randomized unless explicitly noted otherwise. We let $y := A(x; \rho)$ denote running algorithm A on input x and randomness ρ and assigning its output to y. Let $y \leftarrow\!\!\$\, A(x)$ denote $y := A(x; \rho)$ for a uniform ρ. We let GrpGen be a PPT algorithm that takes as input 1^κ and outputs a description $(\mathbb{G}, p, g)$ of a group $\mathbb{G}$ of order a prime $p > 2^\kappa$, and a generator g of $\mathbb{G}$.

Definition 1 (Schnorr signatures [39]). *The Schnorr signature scheme consists of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *defined as follows:*

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

Schnorr signatures are secure under the discrete-logarithm assumption in the random-oracle model [37] and the algebraic-group model [26].

Polynomial interpolation. A polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_t x^t$ of degree t over a field $\mathbb{F}$ can be interpolated from its value on $t+1$ points. For distinct elements $S = \{x_1, \ldots, x_{t+1}\}$ in $\mathbb{F}$, define the Lagrange polynomial as

$$L_i^{(S)}(x) = \prod_{j \in S, j \neq i} \frac{x - x_j}{x_i - x_j}. \tag{1}$$

Given $(x_i, y_i)_{i \in [t+1]}$, we can implicitly evaluate the corresponding polynomial f at any point x as

$$f(x) = \sum_{k \in [t+1]} f(x_k) \cdot L_k^{(S)}(x).$$

Definition 2 (Shamir secret sharing [40]). *The $(t+1, n)$-Shamir secret sharing scheme over a field $\mathbb{F}$ consists of efficient algorithms* $(\mathsf{Share}, \mathsf{Recover})$*, defined as follows:*

- $\mathsf{Share}(\mathsf{sk}, n, t+1) \to \{(x_1, \mathsf{sk}_1), \ldots, (x_n, \mathsf{sk}_n)\}$*: On input a secret* sk*, number of participants n, and threshold $t+1$, first, define a polynomial $f(x) = \mathsf{sk} + a_1 x + a_2 x^2 + \cdots + a_t x^t$ by sampling $a_1, \ldots, a_t \leftarrow\!\!\$\ \mathbb{Z}_p$. Choose distinct elements $x_1, \ldots, x_n$ in $\mathbb{F}$. (These could, for example, be participant indices $1, \ldots, n$, which is the convention we use.) Then, set each participant's share $\mathsf{sk}_i, i \in [n]$, to be the evaluation of $f(i)$:*

$$\mathsf{sk}_i := \mathsf{sk} + \sum_{j \in [t]} a_j i^j.$$

  Output $\{(x_i, \mathsf{sk}_i)\}_{i \in [n]}$.
- $\mathsf{Recover}(t+1, \{(x_i, \mathsf{sk}_i)\}_{i \in \mathcal{S}}) \to \perp / \mathsf{sk}$*: On input threshold $t+1$ and a set of shares $\{(x_i, \mathsf{sk}_i)\}_{i \in \mathcal{S}}$, output $\perp$ if $\mathcal{S} \not\subseteq [n]$ or if $|\mathcal{S}| < t+1$. Otherwise, recover* sk *as follows:*

$$\mathsf{sk} := \sum_{i \in \mathcal{S}} \lambda_i^{\mathcal{S}} \mathsf{sk}_i$$

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

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

Algebraic one-more discrete-logarithm (AOMDL) assumption. The AOMDL assumption was introduced formally in [35], though it was used implicitly in prior work [10, 36, 11, 27]. As in the standard OMDL experiment, the adversary is given as input $(\mathbb{G}, p, g)$ and group elements $(X_1, \ldots, X_\ell)$. The adversary can then query an oracle allowing it to learn specific linear combinations of the discrete logarithms of those group elements. The adversary succeeds if it can compute all the discrete logarithms $(x_1, \ldots, x_\ell)$ while making fewer than ℓ queries.

Assumption 1 (AOMDL) *Define*

$$\mathsf{Adv}^{\ell\text{-aomdl}}_{\mathcal{A}, \mathsf{GrpGen}}(\kappa) = \Pr[\mathsf{Expt}^{\ell\text{-aomdl}}_{\mathcal{A}, \mathsf{GrpGen}}(\kappa) = 1],$$

where $\mathsf{Expt}^{\ell\text{-aomdl}}_{\mathcal{A}, \mathsf{GrpGen}}(\kappa)$ *is defined in Figure 1. The ℓ-AOMDL assumption holds relative to* GrpGen *if* $\mathsf{Adv}^{\ell\text{-aomdl}}_{\mathcal{A}, \mathsf{GrpGen}}(\kappa)$ *is negligible for all PPT adversaries $\mathcal{A}$.*

2.1 Threshold Signatures

We begin with the definition of a threshold signature scheme, adapted to 2-round, "partially non-interactive" schemes (like FROST), where the first round can be executed before the message or signing set are determined. In this work, we

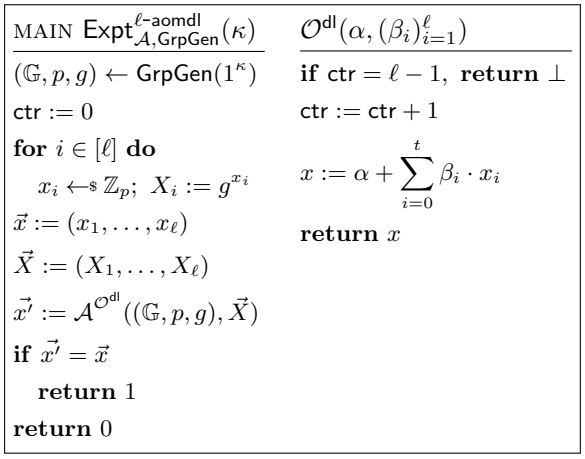

Fig. 1. Experiment for defining the AOMDL assumption.

abstract out the distribution of key material to the parties and focus on signing only; thus, we represent key generation via a centralized algorithm (that could be run by a dealer or implemented by a secure distributed key-generation protocol).

Definition 3. *A* **threshold signature scheme** *consists of algorithms* Setup, KeyGen, Sign_1, Sign_2, PreAgg, $\mathsf{Combine}$, *and* Verify, *with the following syntax:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{par}$*: Takes as input a security parameter and outputs public parameters* par *(given implicitly as input to all other algorithms).*
- $\mathsf{KeyGen}(n, t+1) \to (\mathsf{PK}, \{\mathsf{PK}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in[n]})$*: Takes as input the number of signers* n *and the threshold* $t+1 \leq n$ *and outputs a public key* PK*, a set* $\{\mathsf{PK}_i\}_{i\in[n]}$ *of parties' public keys, and a set* $\{\mathsf{sk}_i\}_{i\in[n]}$ *of secret shares.*
- $(\mathsf{Sign}_1, \mathsf{Sign}_2) \to \{\mathsf{pm}_{i,1}, \mathsf{pm}_{i,2}\}_{i\in\mathcal{S}}$*: These represent the two rounds of a signing protocol. We have*

$$(\mathsf{pm}_{i,1}, \mathsf{st}_{i,1}) := \mathsf{Sign}_1(i, \mathsf{sk}_i)$$
$$\mathsf{pm}_{i,2} := \mathsf{Sign}_2(i, \mathsf{st}_{i,1}, \mathcal{S}, m, \mathcal{PM}_1),$$

  where $\mathsf{pm}_{i,1}, \mathsf{pm}_{i,2}$ *are protocol messages,* $\mathsf{st}_{i,1}$ *is the state of party* i *after the first round,* $\mathcal{S} \subseteq [n], |\mathcal{S}| \geq t+1$ *is a signing set,* m *is a message to be signed, and* $\mathcal{PM}_1 = \{\mathsf{pm}_{i,1}\}_{i\in\mathcal{S}}$*.*
- $\mathsf{PreAgg}(\mathcal{S}, \mathcal{PM}_1) \to \mathcal{PM}_1'$*: This is an (optional) deterministic algorithm that takes as input the signing set* $\mathcal{S}$ *and a set of protocol messages* $\mathcal{PM}_1$*, and outputs a second set of protocol messages* $\mathcal{PM}_1'$*. If utilized, this algorithm is performed after* Sign_1 *and before* Sign_2*.*
- $\mathsf{Combine}(\mathcal{S}, m, \mathcal{PM}_1, \mathcal{PM}_2) \to (m, \sigma)$*: This is a deterministic algorithm that takes as input the signing set* $\mathcal{S}$*, the message* m*, and a set of protocol messages* $\mathcal{PM}_1, \mathcal{PM}_2 = \{\mathsf{pm}_{i,2}\}_{i\in\mathcal{S}}$*, and outputs a signature* σ*.*

– $\mathsf{Verify}(\mathsf{PK}, m, \sigma) \to 0/1$*: This is a deterministic algorithm that takes as input the public key* PK*, a message* m*, and a purported signature* σ *and outputs* 1 *(accept) iff the signature verifies.*

Although the $\{\mathsf{PK}_i\}_{i\in[n]}$ are not explicitly used in the above definition (nor in this work), they can optionally be used for verification purposes (i.e., identifiable abort). Moreover, many distributed key-generation algorithms reveal $\{\mathsf{PK}_i\}_{i\in[n]}$ during their execution. Finally, we remark that in FROST it is possible for an adversary to derive $\{\mathsf{PK}_i\}_{i\in[n]}$ from executions of the signing protocol.

Correctness. A threshold signature scheme is *correct* if for all κ, all par output by $\mathsf{Setup}(1^\kappa)$, all n and $t < n$, all PK, $\{(\mathsf{PK}_i, \mathsf{sk}_i)\}_{i\in[n]}$ output by $\mathsf{KeyGen}(n, t+1)$, all $\mathcal{S} \subseteq [n]$ with $|\mathcal{S}| \geq t+1$, and all m, we have:

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

where $\mathsf{TSignHon}$ is defined in Fig. 2.

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

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

for $i \in \mathcal{S}$ **do** $(\mathsf{pm}_{i,1}, \mathsf{st}_{i,1}) \leftarrow \mathsf{Sign}_1(i, \mathsf{sk}_i)$

for $i \in \mathcal{S}$ **do** $\mathsf{pm}_{i,2} \leftarrow \mathsf{Sign}_2(i, \mathsf{st}_{i,1}, \mathcal{S}, m, \mathcal{PM}_1)$

return $\mathsf{Combine}(\mathcal{S}, m, \mathcal{PM}_1, \mathcal{PM}_2)$

Fig. 2. Algorithm $\mathsf{TSignHon}$ for honestly generating a threshold signature.

Adaptive security. We define adaptive security via the experiment in Figure 3, which we now describe. Key generation is run, and a (stateful) adversary $\mathcal{A}$ is given the public parameters, public key, and parties' public keys. $\mathcal{A}$ can interact with a corruption oracle that provides the adversary with both the secret key and the entire state of any party the adversary chooses to corrupt, subject only to an upper bound t_c on the total number of parties that may be corrupted. We stress here that we do not assume any erasure of parties' states. In addition, the adversary can interact with signing oracles that model the behavior of (honest) parties executing the protocol. Although the signing oracles are defined as if targeting general two-round protocols, the definition actually captures the offline-online (or partially non-interactive) signing pattern, where the first-signing round serves as a preprocessing round that generates nonces before the message to be signed or the set of involved signers is known. Note that we do not assume private or authenticated channels; thus, $\mathcal{A}$ can see all messages honest parties send (even before any parties are corrupted) and send messages of its choice to honest parties in the second round of the protocol. Furthermore, our model allows for a *concurrent* adversary who may open multiple signing sessions simultaneously.

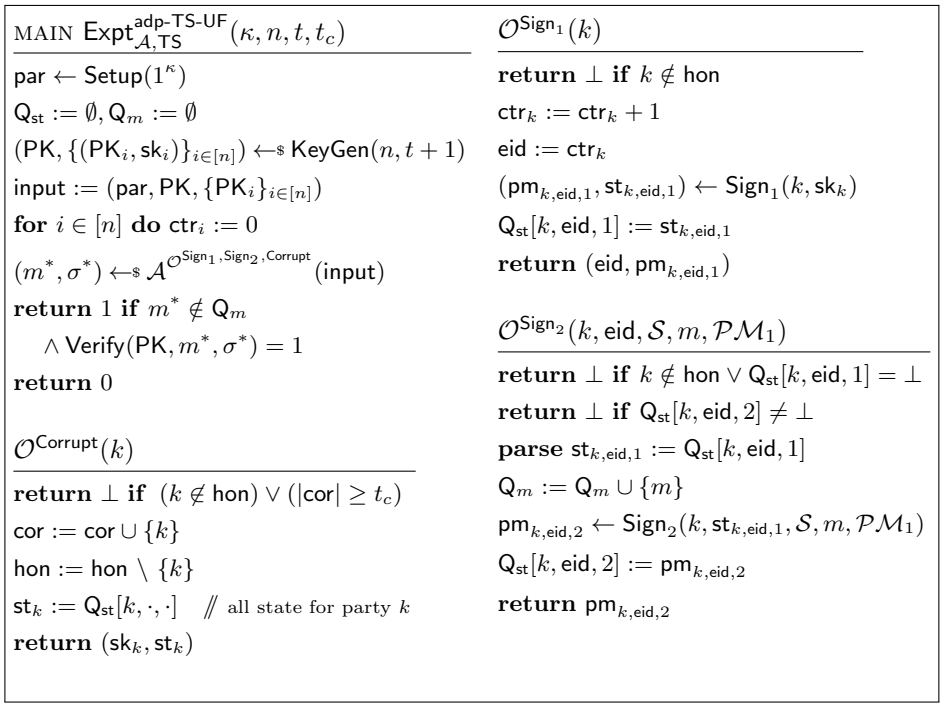

Fig. 3. Experiment for defining adaptive unforgeability. The public parameters par are implicitly given as input to all algorithms, and $\mathcal{A}$ is stateful. Here, $t+1 \leq n$ is the reconstruction threshold and $t_c \leq t$ is the maximum number of signers $\mathcal{A}$ can corrupt.

At the end of the adversary's execution, it succeeds if it outputs a forgery, i.e., a valid message/signature pair (relative to PK) for a message that was not previously queried to any party. (Our definition corresponds to TS-UF-0 from the hierarchy of definitions in [12, 9].)

Definition 4 (Adaptive Security). *For an adversary $\mathcal{A}$ and threshold signature scheme TS, define*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, n, t, t_c) = \Pr[\mathsf{Expt}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, n, t, t_c) = 1].$$

TS *is* (n, t, t_c)-adaptively secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, n, t, t_c)$ is negligible. (We stress that n, t, t_c may depend on κ.)* TS *has* full adaptive security *if it is (n, t, t)-adaptively secure for any $t < n$.*

2.2 The FROST Protocols

$\mathsf{FROST1}, \mathsf{FROST2}$, and $\mathsf{FROST3}$ are all 2-round threshold Schnorr signature schemes where the first round can be run during a pre-processing phase before the message or signing set are known. $\mathsf{FROST1}$ [30] contains the core ideas of the

scheme. FROST2 [22, 12, 9] provides a computational optimization of FROST1. (It was later observed [18] that the use of multi-scalar multiplication allows the computational efficiency of FROST1 to be comparable to FROST2. We include FROST2 for cases where such optimizations are not possible.) FROST3 [38, 17] improves the communication complexity of FROST2.

We describe the signing protocol of FROST1, and then highlight the differences in FROST2 and FROST3. (As discussed previously, we leave key generation out of scope.) An overview of all the schemes is given in Figure 4.

Parameter generation (Setup). On input 1^κ, the parameter-generation algorithm runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrpGen}(1^\kappa)$ and selects hash functions $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$. It outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}})$.

Key generation (KeyGen). On input the total number of signers n and the threshold $t+1$, the key-generation algorithm chooses a secret key $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets the public key to $\mathsf{PK} := g^{\mathsf{sk}}$. It then performs a Shamir secret sharing of sk, computing $\{(i, \mathsf{sk}_i)\}_{i\in[n]} \leftarrow\!\!\$\ \mathsf{Share}(\mathsf{sk}, n, t+1)$. The corresponding public key for each participant is $\mathsf{PK}_i := g^{\mathsf{sk}_i}$. It outputs $(\mathsf{PK}, \{(\mathsf{PK}_i, \mathsf{sk}_i)\}_{i\in[n]})$.

Signing round 1 (Sign_1)**.** On input a participant identifier k, party k samples nonces $r_k, s_k \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets $R_k := g^{r_k}$ and $S_k := g^{s_k}$. It outputs (R_k, S_k).

Signing round 2 (Sign_2)**.** On input a participant identifier k, signing set $\mathcal{S}$, message m, and set of nonce commitments $\mathcal{PM}_1 = \{(i, R_i, S_i)\}_{i\in\mathcal{S}}$, party k first checks that their nonce pair (i.e., $(k, R_k, S_k) \in \{(i, R_i, S_i)\}_{i\in\mathcal{S}}$) is what is recorded in their state from round 1. If so, it computes aggregate nonce commitment $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, $a := \mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, $c := \mathsf{H}_{\mathsf{sig}}(\tilde{R}, \mathsf{PK}, m)$, and partial signature $z_k := r_k + s_k a + c\lambda_k^{\mathcal{S}} \mathsf{sk}_k$ (where $\lambda_k^{\mathcal{S}}$ is the Lagrange coefficient for party k in $\mathcal{S}$). It outputs z_k.

Computing aggregate nonce (CompR)**.** On input $\mathsf{PK}, \mathcal{S}, m$, and $\mathcal{PM}_1$, where $\mathcal{PM}_1 = \{(i, R_i, S_i)\}_{i\in\mathcal{S}}$, the nonce aggregation algorithm computes $a_i := \mathsf{H}_{\mathsf{non}}(i, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ and aggregate nonce $\tilde{R} := \prod_{i\in\mathcal{S}} R_i \cdot {S_i}^{a_i}$.

Combining partial signatures (Combine). On input $\mathcal{S}, m, \mathcal{PM}_1$, and partial signatures $\mathcal{PM}_2 = \{z_i\}_{i\in\mathcal{S}}$, the combiner (who may be one of the signers or an external party) computes the aggregate nonce $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ and $z := \sum_{i\in\mathcal{S}} z_i$ and sets $\sigma := (\tilde{R}, z)$. It outputs σ.

Verification (Verify). On input PK, m, σ, the signature is verified exactly as in the Schnorr signature scheme.

FROST2 uses a single scalar a (instead of a scalar a_i for each party). FROST3 has an additional algorithm PreAgg, which allows for pre-aggregation of nonces, reducing the communication overhead.

We write FROST1[GrpGen], etc., when we wish to emphasize that the scheme uses a particular group-generation algorithm GrpGen.

Static security of FROST. Security of FROST1 for static corruptions was first claimed by Komlo and Goldberg [30]. However, they only gave a security proof for a less-efficient variant of FROST1, arguing heuristically that the result extends to the original protocol. Bellare et al. [9] later gave a rigorous proof of security for FROST1 and FROST2 based on the one-more discrete-logarithm assumption.

Ruffing et al. [38] prove the static security of FROST3 via a reduction to the static security of FROST2; Chu et al. [17] prove the static security of FROST3 with respect to a concrete distributed key-generation protocol.

3 Adaptive Security of FROST for $t/2$ Corruptions

We prove adaptive security of all three variants of FROST for a maximum corruption threshold of $t_c = t/2$ based on the AOMDL assumption in the random-oracle model.

Theorem 1. *Let* GrpGen *be a group generator. For $t < n$ and $t_c = \lfloor t/2 \rfloor$, any* $\mathsf{FROST} \in \{\mathsf{FROST1}[\mathsf{GrpGen}], \mathsf{FROST2}[\mathsf{GrpGen}], \mathsf{FROST3}[\mathsf{GrpGen}]\}$*, and any PPT adversary $\mathcal{A}$ making at most q_s queries to $\mathcal{O}^{\mathsf{Sign}_1}$ and at most q_h random-oracle queries, there exists a PPT adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{FROST}}(\kappa, n, t, t_c) \leq \sqrt{q \cdot \left(\mathsf{Adv}^{(2q_s+t+1)\text{-}\mathsf{aomdl}}_{\mathcal{B},\mathsf{GrpGen}}(\kappa) + \frac{3q^2}{2^\kappa}\right)} + \frac{q}{2^\kappa}\ , \quad (2)$$

where $q = n(q_s + q_h + 1)$ for FROST1 *and $q = q_s + q_h + 1$ for* FROST2 *and* FROST3*. The running time of $\mathcal{B}$ is roughly twice that of $\mathcal{A}$.*

The full proof can be found in Appendix A. The proofs for the three variants of FROST are largely the same; hence, we provide a unified proof and highlight the points where the proofs diverge.

Proof Outline. Let $\mathcal{A}$ be a PPT adversary against the $t/2$ adaptive unforgeability of FROST (Fig. 3). We construct a PPT reduction $\mathcal{B}$ against the $(2q_s+t+1)$-aomdl assumption (Fig. 1). We show that if $\mathcal{A}$ produces a forgery with non-negligible probability, then $\mathcal{B}$ breaks the $(2q_s+t+1)$-aomdl assumption with non-negligible probability.

The reduction $\mathcal{B}$ takes as input $(2q_s+t+1)$-aomdl challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t, X_1, X'_1, \ldots, X_{q_s}, X'_{q_s})$ and aims to output x_i such that $X_i = g^{x_i}$ for all i without querying its discrete-logarithm solution oracle more than $2q_s + t$ times. It runs a forking algorithm on a simulator algorithm $\mathcal{C}$ that simulates the unforgeability game to $\mathcal{A}$ as follows.

$\mathcal{C}$ is given as input the AOMDL challenges and q $(= q_s + q_h + 1)$ random values. For all $i \in [n]$, $\mathcal{C}$ sets each public key share as $\mathsf{PK}_i := X_0 \hat{X}_1^i \cdots \hat{X}_t^{i^t}$. The threshold public key is $\mathsf{PK} := X_0$. $\mathcal{C}$ then returns $(\mathsf{PK}, \{\mathsf{PK}_i\}_{i\in[n]})$ to $\mathcal{A}$. $\mathcal{C}$ simulates random-oracle queries using the q random values. $\mathcal{C}$ simulates each query to $\mathcal{O}^{\mathsf{Sign}_1}$ by returning a pair (X_i, X'_i), and simulates each query to $\mathcal{O}^{\mathsf{Sign}_2}$ by making a single query for the DL of $g^{z_k} = X_i X_i'^a \mathsf{PK}_k^{c\lambda_k^S}$ to obtain z_k for party k. $\mathcal{C}$'s DL solution queries are forwarded to $\mathcal{B}$'s DL solution oracle.

$\mathcal{C}$ is forked in order to extract $\mathsf{sk} = x_0$ 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 $\mathcal{C}$ can corrupt the remaining itself.) Then $\mathcal{C}$, and therefore $\mathcal{B}$, has made t DL solution queries on $g^{\mathsf{sk}_k} = X_0 \hat{X}_1^k \cdots \hat{X}_t^{k^t}$ to obtain

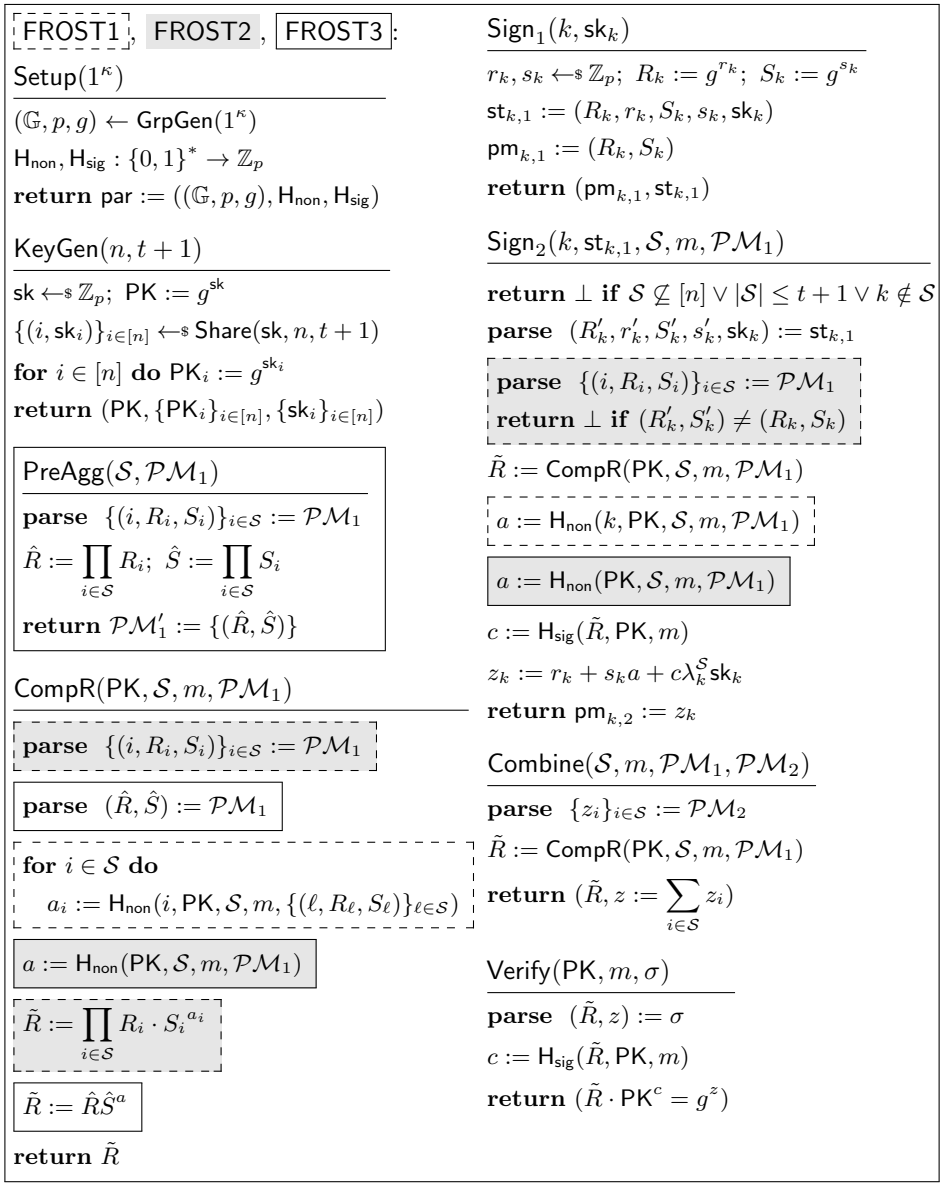

Fig. 4. The FROST1, FROST2, and FROST3 threshold signature schemes, where FROST1 contains only the dashed boxes, FROST2 contains only the highlighted boxes, and FROST3 contains only the solid boxes. The public parameters par and public key PK are implicitly given as input to all algorithms. CompR is an algorithm that computes the aggregate nonce $\tilde{R}$ from a second-round input.

sk_k. $\mathcal{B}$ now knows $t+1$ points on the polynomial $f(Z) = x_0 + \hat{x}_1 Z + \cdots + \hat{x}_t Z^t$ defining key generation. Thus, $\mathcal{B}$ can solve for $\hat{x}_1, \ldots, \hat{x}_t$. For the remaining $(x_1, x'_1, \ldots, x_{q_s}, x'_{q_s})$, $\mathcal{B}$ first considers nonce pairs (X_i, X'_i) that remain honest at the end of the game. (1) If no $\mathcal{O}^{\mathsf{Sign}_2}$ query has been made on (X_i, X'_i) in either iteration of $\mathcal{A}$, $\mathcal{B}$ queries each DL directly. (2) If one $\mathcal{O}^{\mathsf{Sign}_2}$ query has been made, $\mathcal{B}$ has already made one DL solution query, for z_k, and makes one more for X_i. $\mathcal{B}$ can then solve for the DL of X'_i itself from $z_k = x_i + x'_i a + c\lambda_k^{\mathcal{S}} \mathsf{sk}_k$, given that it knows sk_k or can compute it as $f(k)$. If two $\mathcal{O}^{\mathsf{Sign}_2}$ queries have been made, one in each iteration, there are two cases. (3) If the query occurs before the fork, then it is the same in both iterations and this is the same as case (2). (4) If the query occurs after the fork, then we have:

$$x'_i = \frac{z_{k'_j} - z_{k_j} + c_j \gamma_{k_j} \mathsf{sk}_{k_j} - c_{j'} \gamma_{k_{j'}} \mathsf{sk}_{k'_j}}{a_{j'} - a_j}$$

where $a_{j'} \neq a_j$ due to forking, and $\mathcal{B}$ then solves for x_i from z_k. $\mathcal{B}$ then considers nonce pairs (X_i, X'_i) that are corrupted by the end of the game. When a party is corrupted, $\mathcal{B}$ queries its DL solution oracle on PK_k to get sk_k. If no $\mathcal{O}^{\mathsf{Sign}_2}$ query has been made on (X_i, X'_i), $\mathcal{B}$ queries for each DL directly. This is the same as case (1); the cases (2), (3), and (4) are similar. Thus, $\mathcal{B}$ has made the same queries for these cases at the time of corruption. Overall, $\mathcal{B}$ has made exactly two DL queries for each nonce pair (X_i, X'_i). Thus, $\mathcal{B}$ has solved for $(x_0, \hat{x}_1, \ldots, \hat{x}_t, x_1, x'_1, \ldots, x_{q_s}, x'_{q_s})$ using only $2q_s + t$ DL solution queries to win the $(2q_s + t + 1)$-aomdl game.

4 Full Adaptive Security of FROST

In this section we introduce the low-dimensional vector representation (LDVR) problem and the corresponding assumption that the problem is hard. In Section 4.2 we then show that the LDVR assumption is necessary for adaptive security of a broad class of threshold Schnorr schemes (including FROST), in the sense that an efficient algorithm solving the LDVR problem implies an explicit adaptive attack on those schemes. In Section 4.3, we prove full adaptive security of FROST under the AOMDL and LDVR assumptions in the AGM and ROM. In Section 4.4 we study the hardness of the LDVR problem itself, showing that, for some parameters, the problem is unconditionally hard, and in other cases providing evidence that the problem is computationally hard.

4.1 The Low-Dimensional Vector Representation Problem

The *low-dimensional vector-representation (LDVR) problem* is parameterized by t_c, t, n with $t_c \leq t < n$. (Looking ahead to the application to threshold schemes, n will correspond to the total number of parties, $t+1$ to the signing threshold, and t_c to the corruption bound.) Roughly speaking, an adversary $\mathcal{A}$ is given

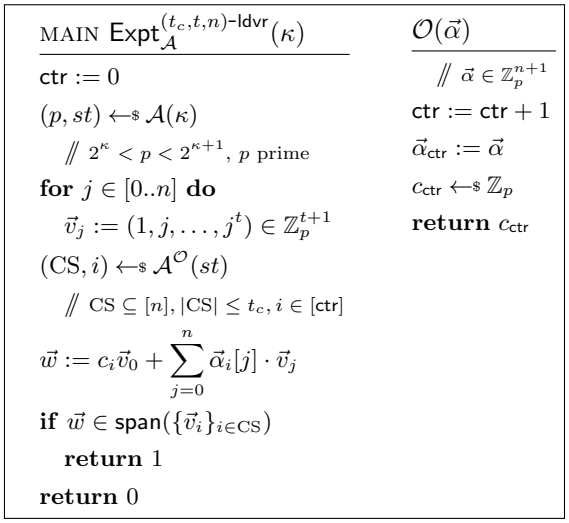

Fig. 5. The LDVR experiment with parameters $t_c \leq t < n$.

vectors $\vec{v}_0, \ldots, \vec{v}_n \in \mathbb{Z}_p^{t+1}$ and is then asked to find $\vec{\alpha} \in \mathbb{Z}_p^{n+1}$ such that the vector

$$\vec{w} := \mathcal{O}(\vec{\alpha}) \cdot \vec{v}_0 + \sum_{i=0}^{n} \alpha[i] \cdot \vec{v}_i$$

lies in a t_c-dimensional subspace of $\mathbb{Z}_p^{t+1}$, where $\mathcal{O} : \mathbb{Z}_p^{n+1} \to \mathbb{Z}_p$ can be viewed as a random oracle. More specifically, in the formal experiment defining the problem (cf. Figure 5) the $\vec{v}_0, \ldots, \vec{v}_n$ are "Vandermonde vectors" $\vec{v}_i = (1, i, \ldots, i^t)$; moreover, $\mathcal{A}$ must specify a set $\mathrm{CS} \in [n]$ of size t_c defining the t_c-dimensional subspace $\mathsf{span}(\{\vec{v}_i\}_{i\in\mathrm{CS}})$ in which $\vec{w}$ must lie. We also allow $\mathcal{O}$ to be queried multiple times with the same input; each such query results in a fresh random output.

We denote the advantage of $\mathcal{A}$ in solving the (t_c, t, n)-LDVR problem as

$$\mathsf{Adv}_{\mathcal{A}}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}(\kappa) = \Pr\left[\mathsf{Expt}_{\mathcal{A}}^{(t_c,t,n)\text{-}\mathsf{ldvr}}(\kappa) = 1\right].$$

We formulate in particular the following assumption.

Definition 5 (LDVR Assumption). *We say the (t_c, t, n)-LDVR assumption holds (where t_c, t, n may be functions of κ) if $\mathsf{Adv}_{\mathcal{A}}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}(\kappa)$ is negligible for any PPT algorithm $\mathcal{A}$.*

We analyze the hardness of this problem in Section 4.4. There, we show that the assumption holds unconditionally for several interesting values of t_c, t, n. When $t_c = t$, it seems reasonable to conjecture that solving the problem requires time $\Omega(\sqrt{p})$ (as hinted by our preliminary cryptanalysis in Section 4.5), which

roughly matches the conjectured hardness of the discrete-logarithm problem in a group of order p on a well-chosen elliptic curve.

Relationship to prior work [21]. Prior work [21] shows an attack against a large class of Schnorr threshold signature schemes (the same class we discuss in the next section) given access to an oracle $\mathcal{O}_P$ that takes an arbitrary vector $\vec{w} \in \mathbb{Z}_p^{t+1}$ as input and returns a subset $\mathrm{CS} \subseteq [n]$ of size t_c such that $\vec{w} \in \mathsf{span}(\{\vec{u}_i\}_{i \in \mathrm{CS}})$, if such a subset exists. The problem solved by the oracle $\mathcal{O}_P$ is referred to as problem P. (We refer to $\vec{w}$ for which such a subset CS exists as *good*.) Clearly, the LDVR assumption cannot hold relative to $\mathcal{O}_P$ if any of the $\vec{w}$-vectors defined by a query to $\mathcal{O}$ during the adversary's execution in the LDVR experiment are good. For this reason, the ideas from [21] give an efficient algorithm for the LDVR problem given access to $\mathcal{O}_P$, as their analysis implies that when the $\vec{\alpha}$ queried to $\mathcal{O}$ are uniform, the resulting vector $\vec{w}$ is good with high probability (depending on t_c, t, and n). Thus, with the same probability, $\mathcal{O}_P$ can be used to find a suitable set CS such that $\vec{w} \in \mathsf{span}(\{\vec{v}_i\}_{i \in \mathrm{CS}})$, and thus solve the LDVR problem. In the case $t_c = t$, they show that this happens with probability $1/2$ if (roughly) $\binom{n}{t} > p$; their argument extends to $t_c < t$, albeit with a sharp drop in probability.

Conversely, if the resulting vector $\vec{w}$ is *good* with negligible probability, the LDVR problem is unconditionally hard. In Section 4.4, we show an explicit upper bound on this probability, which shows for certain choices of (t_c, t, n), the LDVR problem is unconditionally hard. However, it is unclear whether problem P is hard or not in those scenarios.[5]

Summarizing: prior work [21] shows an oracle $\mathcal{O}_P$ relative to which there is an attack on both a certain class of Schnorr threshold schemes and the LDVR problem for certain choices of (t_c, t, n). That is, if P is easy, LDVR is easy for the same parameter regimes where the attack from [21] applies. We do not know whether our unconditional analysis is tight; thus, there is a range of parameter choices where it is possible that LDVR is hard, but P is easy. And of course, it is always possible that LDVR is easy, but P is hard in the worst case. In Section 4.2, we show that an oracle solving the LDVR problem also suffices to implement the attack on the same class of Schnorr threshold schemes.

4.2 Necessity of the LDVR Assumption for Adaptive Security

Here, we show that if the (t_c, t, n)-LDVR problem is easy, there is an explicit adaptive attack on a broad class of threshold Schnorr schemes for the same parameters. The attack does not require the adversary to request any signatures; it also does not require the adversary to learn anything about the parties' internal states other than their secret keys, so applies even if secure erasure is assumed. This shows that the LDVR assumption is necessary if there is to be any hope of proving such schemes adaptively secure.

Specifically, we consider protocols with the following properties:

[5] Here, we refer to the worst-case hardness of P.

– The secret-key shares of the parties are Shamir shares of the master secret key sk, with the ith party holding share sk_i.

– "Commitments" $\{\mathsf{PK}_i = g^{\mathsf{sk}_i}\}_{i\in[n]}$ can be computed by the adversary. Note that for the purposes of the attack it does not matter whether these commitments are required by the protocol itself, are available to the adversary because of the distributed key-generation (DKG) protocol used, or computable from other information available to the adversary.

The above conditions apply to a wide range of schemes [33, 38, 41, 23, 31] in addition to FROST and its variants. For the case of FROST in particular, note that (1) although commitments to the parties' key shares are not required to run the signing protocol, such commitments are assumed to be publicly available in the IETF RFC [19] and are necessary for identifiable abort; in any event (2) commitments to the parties' key shares can be computed from honest executions of the FROST signing protocol. Moreover, (3) we are not aware of any dlog-based DKG protocol that does not reveal these commitments.

Fix t_c, t, n for which the (t_c, t, n)-LDVR is easy, and let $\mathcal{A}'$ be an efficient algorithm solving it. Consider a Schnorr threshold scheme of the type above with n parties and signing threshold $t+1$. We show an adversary $\mathcal{A}$ (similar to the prior one [21]) who corrupts at most t_c parties and forges a signature. At the outset, $\mathcal{A}$ is given $\mathsf{PK} = g^{\mathsf{sk}}$ and $\{\mathsf{PK}_i = g^{\mathsf{sk}_i}\}_{i\in[n]}$, where $\mathsf{sk}_i = a(i)$ for a polynomial $a(X) = a_0 + a_1 X + \cdots + a_t X^t$ with $a_0 = \mathsf{sk}$. Then $\mathcal{A}$ runs $\mathcal{A}'$ (we assume here that $\mathcal{A}'$ uses the same prime p used by the scheme). When $\mathcal{A}'$ issues its ith query to $\mathcal{O}$ with input $\vec{\alpha}_i$, adversary $\mathcal{A}$ computes

$$R_i := \mathsf{PK}^{\vec{\alpha}_i[0]} \cdot \prod_{j=1}^{n} \mathsf{PK}_i^{\vec{\alpha}_i[j]}$$

followed by $c_i := \mathsf{H}_{\mathsf{sig}}(R_i, \mathsf{PK}, m_i)$ where m_i is distinct from $\{m_j\}_{j<i}$. $\mathcal{A}$ returns c_i to $\mathcal{A}'$. When $\mathcal{A}'$ outputs (CS, i^*), adversary $\mathcal{A}$ obtains $\{\mathsf{sk}_i\}_{i\in\mathrm{CS}}$ by corrupting all signers $i \in \mathrm{CS}$, and (via linear algebra) computes $(\beta_j)_{j\in\mathrm{CS}}$ such that

$$\vec{w}_{i^*} \stackrel{\text{def}}{=} c_{i^*} \cdot \vec{v}_0 + \sum_{j=0}^{n} \vec{\alpha}_{i^*}[j] \cdot \vec{v}_j = \sum_{j\in\mathrm{CS}} \beta_j \cdot \vec{v}_j.$$

Finally, $\mathcal{A}$ outputs $(m_{i^*}, \sigma = (R_{i^*}, z_{i^*}))$ where $z_{i^*} = \sum_{j\in\mathrm{CS}} \beta_j \cdot \mathsf{sk}_j$.

If $\mathsf{H}_{\mathsf{sig}}$ is modeled as a random oracle, the view of $\mathcal{A}'$ is the same as its view in $\mathsf{Expt}_{\mathcal{A}'}^{(t_c,t,n)\text{-ldvr}}(\kappa)$. Moreover, $\mathcal{A}$ succeeds when $\mathcal{A}'$ does. To see this, let $\langle\cdot,\cdot\rangle$ denote the dot product and note that if we let $\vec{a} = [a_0, a_1, \ldots, a_t]$ be the vector

of coefficients of the sharing polynomial a, then $\mathsf{sk}_i = \langle \vec{a}, \vec{v}_i \rangle$. We have

$$\begin{aligned} z_{i^*} = \sum_{j \in \mathrm{CS}} \beta_j \cdot \langle \vec{a}, \vec{v}_j \rangle &= \left\langle \vec{a}, \sum_{j \in \mathrm{CS}} \beta_j \cdot \vec{v}_j \right\rangle \\ &= \langle \vec{a}, \vec{w}_{i^*} \rangle \\ &= (c_{i^*} + \vec{\alpha}_{i^*}[0]) \cdot \mathsf{sk} + \sum_{j=1}^{n} \vec{\alpha}_{i^*}[j] \cdot \mathsf{sk}_j \ , \end{aligned}$$

which is exactly the discrete logarithm of $R_{i^*} \cdot \mathsf{PK}^{c_{i^*}}$. Thus, (R_{i^*}, z_{i^*}) is a valid signature on m_{i^*}.

4.3 Proof of Full Adaptive Security

We prove that full adaptive security of all three variants of FROST is implied by the hardness of the LDVR problem and the AOMDL assumption in the algebraic-group model and the random-oracle model.

Theorem 2. *Let* GrpGen *be a group generator. For all* $t_c \leq t < n$*, any* $\mathsf{FROST} \in \{\mathsf{FROST1}[\mathsf{GrpGen}], \mathsf{FROST2}[\mathsf{GrpGen}], \mathsf{FROST3}[\mathsf{GrpGen}]\}$*, and any* algebraic *PPT adversary* $\mathcal{A}$ *making at most* q_s *queries to* Sign_1 *and at most* q_h *random-oracle queries, there is a PPT adversary* $\mathcal{B}$ *and a PPT adversary* $\mathcal{C}$*, who makes at most* q *queries to its oracle, such that*

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{FROST}}(\kappa, n, t, t_c) \leq \mathsf{Adv}^{(2q_s+t+1)\text{-}\mathsf{aomdl}}_{\mathcal{B},\mathsf{GrpGen}}(\kappa) + \mathsf{Adv}^{(t_c,t,n)\text{-}\mathsf{ldvr}}_{\mathcal{C}}(\kappa) + \frac{2(nq)^2}{2^\kappa} \ , \tag{3}$$

where $q = q_s + q_h + 1$*. The running times of* $\mathcal{B}$ *and* $\mathcal{C}$ *are roughly the same as that of* $\mathcal{A}$*.*

The security proofs for the three FROST variants are largely the same; hence, we provide a unified proof and highlight the places where the proofs diverge.

Proof Outline. Suppose $\mathcal{A}$ outputs a valid forgery $(m^*, \sigma^* = (R^*, z^*))$ satisfying $g^{z^*} = R^* \cdot \mathsf{PK}^{c^*}$ with $c^* = \mathsf{H}_{\mathsf{sig}}(R^*, \mathsf{PK}, m^*)$. We show that given a successful adversary, we can either define an efficient reduction to AOMDL, or an efficient reduction to the LDVR problem, using the following steps. The rough idea of the AOMDL reduction can be described as follows. The reduction first sets the public key PK and public key shares $\{\mathsf{PK}_k\}_{k \in [n]}$ using $(t+1)$ AOMDL challenges $X_0, \ldots, X_t$, where $\mathsf{PK} := X_0$ and $\mathsf{PK}_k = X_0 \prod_{i \in [t]} X_i^{k^i}$. Then, the reduction runs the adversary $\mathcal{A}$ with the public key and public key shares by simulating the signing and corruption oracles on its own. For $\mathcal{O}^{\mathsf{Sign}_1}(k)$, the reduction returns (R_k, S_k), which it sets as two new AOMDL challenges. For $\mathcal{O}^{\mathsf{Sign}_2}$ and $\mathcal{O}^{\mathsf{Corrupt}}$, the reduction simulates the responses by querying its own $\mathcal{O}^{\mathsf{dl}}$ oracle. After $\mathcal{A}$ returns a valid forgery as described above, the reduction attempts to extract a *new* linear relation among the challenges from the forgery, which helps it to win

the AOMDL game. Here, "new" means that the relation does not lie in the linear span of the $\mathcal{O}^{\mathsf{dl}}$ oracle queries made by the reduction. More precisely, there are the following three cases.

1. We first show one possible strategy by which the AOMDL reduction can win. Given a forgery as described above, since $\mathcal{A}$ is algebraic, it must provide a representation of R^* with respect to g, PK, $\{\mathsf{PK}_k\}_{k\in[n]}$, and all the nonce commitments $(R_{i,k}, S_{i,k})$ output by honest signers $k \in \mathsf{hon}$ during executions of the protocol, where $(R_{i,k}, S_{i,k})$ is the i-th pair sent by the k-th signer. The AOMDL reduction can replace each term in the representation of R^* involving $(R_{i,k}, S_{i,k})$ (denoted by $R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}}$) with a term involving g and PK_k alone. This is possible if (1) there is a Sign_2 query corresponding to this nonce commitment pair for which the oracle outputs z_k satisfying $g^{z_k} = R_{i,k} S_{i,k}^{a} \cdot \mathsf{PK}_k^{c\lambda_k}$, and (2) $\gamma'_{i,k} = a \cdot \gamma_{i,k}$. If one of these conditions does not hold for some (i,k), then the equation $g^{z^*} = R^* \cdot \mathsf{PK}^{c^*}$ gives a new linear relation among $R_{i,k}, S_{i,k}$, and other group elements. From this, the AOMDL reduction can solve for a non-trivial discrete-logarithm solution without querying its own $\mathcal{O}^{\mathsf{dl}}$ oracle.
2. If both conditions (1) and (2) hold for all $(R_{i,k}, S_{i,k})$, then $R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}}$ can be replaced with $(g^{z_k} \mathsf{PK}_k^{-c\lambda_k})^{\gamma_{i,k}}$, and we show a second strategy by which an AOMDL reduction can win. The AOMDL reduction again defines R^* with the representation described in the step above, resulting in a representation of R^* with respect to $g, \mathsf{PK}, \{\mathsf{PK}_k\}_{k\in[n]}$ only, i.e., we have $R^* = g^{\delta} \mathsf{PK}^{\beta_0} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k}$ for known $\delta, \beta_0, \{\beta_k\}_{k=1}^n$. Since $g^{z^*} = R^* \cdot \mathsf{PK}^{c^*}$, we have

$$\mathsf{PK}^{\beta_0 + c^*} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} = g^{-\delta + z^*} \ . \tag{4}$$

   Denote $\vec{v}_k := (1, k, k^2, \ldots, k^t)$ for $k \in [0..n]$ and $\vec{v}^* = c^* \vec{v}_0 + \sum_{k\in[0..n]} \beta_k \vec{v}_k$. Since $\mathsf{PK}_k = \prod_{j=0}^{t} X_j^{v_{k,j}}$ and $\mathsf{PK} = \prod_{j=0}^{t} X_j v_{0,j}$ (recalled that $X_0, \ldots, X_t$ are $(t+1)$ AOMDL challenges), by Equation (4),

$$\prod_{j=0}^{t} X_j^{v_j^*} = g^{-\delta + z^*} \ . \tag{5}$$

   Then, if $\vec{v}^* \notin \mathsf{span}(\{\vec{v}_j\}_{j\in\mathsf{cor}})$, Equation (5) gives a new linear relation among $X_0, \ldots, X_t$. Therefore, the AOMDL reduction can again solve for a non-trivial discrete-logarithm solution without querying its own $\mathcal{O}^{\mathsf{dl}}$ oracle.
3. If the check in Step 2 fails, i.e., $\vec{v}^* \in \mathsf{span}(\{\vec{v}_j\}_{j\in\mathsf{cor}})$, then $\mathcal{A}$ can compute $\log_g \left(\mathsf{PK}^{\beta_0 + c^*} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} \right)$ using $\{\mathsf{sk}_j\}_{j\in\mathsf{cor}}$, and thus the AOMDL reduction cannot hope to obtain a new linear relation from the above equation. However, in this case, we can define a second reduction that solves the LDVR problem. In particular, $(\beta_0, \ldots, \beta_n)$ can be seen as an oracle query in the (t_c, t, n)-LDVR game with response c^*, and since $|\mathsf{cor}| \le t_c$ and $c^* \vec{v}_0 + \sum_{k\in[0..n]} \beta_k \vec{v}_k \in$

$\mathsf{span}(\{\vec{v}_j\}_{j\in\mathsf{cor}})$, this gives a valid solution for the LDVR problem. The formal reduction is more complicated. In particular, a key challenge is that the $\{\beta_k\}_{k\in[0..n]}$ need to be extracted *at the time* $\mathcal{A}$ makes the random-oracle query to $\mathsf{H}_{\mathsf{sig}}(R^*, \mathsf{PK}, m^*)$. We refer to the proof of Lemma 1 for details.

Proof (of Theorem 2). Let $\mathcal{A}$ be an algebraic PPT adversary in the adaptive unforgeability experiment $\mathsf{Expt}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{FROST}}(\kappa, n, t, t_c)$ that makes up to q_h queries to $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$, up to q_s queries to Sign_1, and up to t_c queries to Corrupt. We assume, without loss of generality, that $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on its forgery (R^*, PK, m^*), and that it always queries $\mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (resp., $\mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ for FROST1) before querying $\mathsf{Sign}_1(k, \mathcal{S}, m, \mathcal{PM}_1)$ for some $k \in \mathcal{S} \cap \mathsf{hon}$. We now define a PPT algorithm $\mathcal{B}$ for the AOMDL experiment.

$\mathcal{B}$ is initialized by the AOMDL challenger with the group description $(\mathbb{G}, g, p)$ and a set of $2q_s+t+1$ AOMDL challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t, X_1, X'_1, \ldots, X_{q_s}, X'_{q_s})$. $\mathcal{B}$ has access to a discrete-logarithm solution oracle $\mathcal{O}^{\mathsf{dl}}$, and its goal is to output discrete logarithms $(x_0, \hat{x}_1, \ldots, \hat{x}_t, x_1, x'_1, \ldots, x_{q_s}, x'_{q_s})$ without querying $\mathcal{O}^{\mathsf{dl}}$ more than $2q_s + t$ times. $\mathcal{B}$ begins by initializing the following values:

1. Tables $\mathsf{Q}_{\mathsf{non}} := \emptyset$, $\mathsf{Q}_{\mathsf{sig}} := \emptyset$ to store random-oracle queries to $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}}$ and corresponding simulated responses;
2. Table $\mathsf{Q}_m := \emptyset$, to store the set of messages queried by $\mathcal{A}$ to Sign_2;
3. Table $\mathsf{Q}_{\mathsf{st}} := \emptyset$ of participants' states for all signing sessions initiated by $\mathcal{A}$;
4. Counter $\mathsf{ctr} := 0$ to iterate through pairs of AOMDL challenges;
5. Counter $\mathsf{ctr}_k := 0$ for $k \in [n]$ to count the number of Sign_1 queries for party k.

To simulate key generation, $\mathcal{B}$ uses the AOMDL challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t)$ to define a polynomial $f(Z) = x_0 + \hat{x}_1 Z + \cdots + \hat{x}_t Z^t$ "in the exponent" by deriving each party's public key as:

$$\mathsf{PK}_i = X_0 \hat{X}_1^i \cdots \hat{X}_t^{i^t}, \tag{6}$$

where implicitly $\mathsf{sk}_i = f(i)$. The threshold public key is $\mathsf{PK} := X_0$ with corresponding secret key $\mathsf{sk} = x_0 = f(0)$. $\mathcal{B}$ then runs $\mathcal{A}((\mathbb{G}, g, p), \mathsf{PK}, \{\mathsf{PK}_i\}_{i\in[n]})$ with the random oracles and signing oracles simulated as follows.

Random oracles: These are simulated in the natural way, choosing a uniform output for each fresh query made by $\mathcal{A}$. Queries/responses to $\mathsf{H}_{\mathsf{non}}$ (resp., $\mathsf{H}_{\mathsf{sig}}$) are stored in $\mathsf{Q}_{\mathsf{non}}$ (resp., $\mathsf{Q}_{\mathsf{sig}}$).

$\mathcal{O}^{\mathsf{Sign}_1}$ queries: When $\mathcal{A}$ queries Sign_1 for party $k \in \mathsf{hon}$, algorithm $\mathcal{B}$ does the following:

1. Increment $\mathsf{ctr}_k := \mathsf{ctr}_k + 1$ and set $\mathsf{eid} := \mathsf{ctr}_k$.
2. Increment $\mathsf{ctr} := \mathsf{ctr} + 1$ and set $(R_k, S_k) := (X_{\mathsf{ctr}}, X'_{\mathsf{ctr}})$.
3. Set $\mathsf{st}_{k,\mathsf{eid},1} := (R_k, \bot, S_k, \bot, \bot)$. # L-Note: i.e., r_k, s_k, sk_k are not yet defined.
4. Update $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] := \mathsf{st}_{k,\mathsf{eid},1}$.

Finally, $\mathcal{B}$ returns (R_k, S_k) to $\mathcal{A}$.

$\mathcal{O}^{\mathsf{Sign}_2}$ **queries:** When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_2}(k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$ for $k \in \mathsf{hon}$, algorithm $\mathcal{B}$ first performs the checks in the $\mathcal{O}^{\mathsf{Sign}_2}$ oracle (Fig. 3) and the Sign_2 algorithm (Fig. 4). If they pass, then in particular $\mathsf{st}_{k,\mathsf{eid},1} = (R_k, \bot, S_k, \bot, \bot)$. # L-Note: i.e., again, r_k, s_k, sk_k are not yet defined. Then $\mathcal{B}$ does the following:

1. Set $\mathsf{Q}_m := \mathsf{Q}_m \cup \{m\}$.
2. Run the Sign_2 algorithm (Fig. 4) to obtain a and c. Note that $\mathcal{B}$ can internally query $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$ as needed.
3. Set $Z_k := R_k S_k^a \cdot \mathsf{PK}_k{}^{c\lambda_k}$, i.e., $Z_k = X_{i'}(X'_{i'})^a X_0^{c\lambda_k} \hat{X}_1^{c\lambda_k k} \cdots \hat{X}_t^{c\lambda_k k^t}$ for some $i' \in [q_s]$. $\mathcal{B}$ then queries $\mathcal{O}^{\mathsf{dl}}$ on Z_k with $\alpha = 0$ and
$$(\beta_i)_{i=1}^{2q_s+t+1} = (c\lambda_k, c\lambda_k k, \ldots, c\lambda_k k^t, 0, \ldots, 0, 1, a, 0, \ldots, 0)$$
(where the 1 is in position $t + 2i'$), and receives z_k.
4. Set $\mathsf{st}_{k,\mathsf{eid},2} := z_k$ and return z_k to $\mathcal{A}$.

Corruptions: When $\mathcal{A}$ corrupts party k, $\mathcal{B}$ sets $\mathsf{cor} := \mathsf{cor} \cup \{k\}$ and $\mathsf{hon} := \mathsf{hon} \backslash \{k\}$. Then $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ on PK_k with $\alpha = 0$ and (cf. Eq. (22))
$$(\beta_i)_{i=1}^{2q_s+t+1} = (1, k, \ldots, k^t, 0, \ldots, 0)$$
and receives sk_k. Then $\mathcal{B}$ does the following:

1. If no entry $\mathsf{Q}_{\mathsf{st}}[k, \star, 1]$ exists, then $\mathcal{B}$ returns sk_k to $\mathcal{A}$. (This corresponds to the case where $\mathcal{A}$ has not previously queried party k to Sign_1.)
2. Otherwise
   (a) For each entry $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] \neq \bot$ with $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] = \bot$, (This corresponds to round-1 messages that were never used in round 2.) algorithm $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ on R_k (which is equal to $X_{i'}$ for some $i' \in [q_s]$) with $\alpha = 0$ and $(\beta_i)_{i=1}^{2q_s+t+1} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + 2i'$; it receives r_k in response. $\mathcal{B}$ similarly queries $\mathcal{O}^{\mathsf{dl}}$ on S_k (with $\alpha = 0$ and $(\beta_i)_{i=1}^{2q_s+t+1} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + 1 + 2i'$), and receives s_k. Then $\mathcal{B}$ sets $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] = (R_k, r_k, S_k, s_k, \mathsf{sk}_k)$.
   (b) For each entry $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 2] \neq \bot$, (This corresponds to Round 1 queries that made it to Round 2.) algorithm $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ on S_k (which is equal to $X'_{i'+1}$ for some $i' \in [q_s]$) with $\alpha = 0$ and $(\beta_i)_{i=1}^{2q_s+t+1} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + 1 + 2i'$; it receives s_k in response. Note that $\mathcal{B}$ must have already made an $\mathcal{O}^{\mathsf{dl}}$ query to obtain z_k, so it can compute $r_k = z_k - s_k a - c\lambda_k \mathsf{sk}_k$. Then $\mathcal{B}$ sets $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] = (R_k, r_k, S_k, s_k, \mathsf{sk}_k)$.

   Finally, $\mathcal{B}$ returns sk_k and all state $\mathsf{Q}_{\mathsf{st}}[k, \cdot, \cdot]$ for party k to $\mathcal{A}$.

Output: When $\mathcal{A}$ terminates with output $(m^*, \sigma^* = (R^*, z^*))$, then $\mathcal{B}$ aborts if $\mathcal{A}$ does not succeed. $\mathcal{B}$ finds $\mathcal{A}$'s corresponding query to $\mathsf{H}_{\mathsf{sig}}$ to obtain $c^* := \mathsf{H}_{\mathsf{sig}}(R^*, \mathsf{PK}, m^*)$. When $\mathcal{A}$ made this query, it must have provided a representation
$$R^* = g^\delta \cdot \mathsf{PK}^\beta \cdot \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} \prod_{i=1}^{q_s} R_{i,1}^{\gamma_{i,1}} S_{i,1}^{\gamma'_{i,1}} \cdots R_{i,n}^{\gamma_{i,n}} S_{i,n}^{\gamma'_{i,n}}\,, \tag{7}$$

based on all group elements it had seen so far of the form, where $(R_{i,k}, S_{i,k})$ denotes the output of the ith query $\mathsf{Sign}(k)$. (Note that the above includes some group elements that $\mathcal{A}$ may not have known at the time it made the $\mathsf{H}_{\mathsf{sig}}$ query in question, in which case the corresponding $\gamma_{i,k}, \gamma'_{i,k}$ will be 0.) Let hon (resp., cor) be the set of honest (resp., corrupted) signers at the end of $\mathcal{A}$'s execution For $i \in [q_s]$ and $k \in \mathsf{hon}$, if $\mathcal{A}$ made the corresponding Sign_2 query (i.e. $\mathsf{Q}_{\mathsf{st}}[k, i, 2] \neq \bot$), then let $(k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$ be that query. Note that, because $\mathcal{B}$ simulated Sign_2, it knows $(a_{i,k}, z_{i,k}, c_{i,k}, \lambda_{i,k})$ such that

$$g^{z_{i,k}} = R_{i,k} S_{i,k}^{a_{i,k}} \mathsf{PK}_k^{c_{i,k} \cdot \lambda_{i,k}}, \tag{8}$$

where $z_{i,k} = \mathsf{Q}_{\mathsf{st}}[k, i, 2]$, $a_{i,k} = \mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, $c_{i,k} = \mathsf{H}_{\mathsf{sig}}(\tilde{R}, \mathsf{PK}, m)$ with $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, and $\lambda_{i,k} = \lambda_k^{\mathcal{S}}$. (For $\mathsf{FROST1}$, $a_{i,k} = \mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$.) Call (i, k) *bad* if $(\gamma_{i,k}, \gamma'_{i,k}) \neq (0, 0)$ and either $\mathsf{Q}_{\mathsf{st}}[k, i, 2] = \bot$ or $\gamma'_{i,k} \neq \gamma_{i,k} a_{i,k}$, and let BadRS be the event that some $(i, k) \in [q_s] \times \mathsf{hon}$ is bad. [6] (Intuitively, if (i, k) is bad then $\mathcal{B}$ cannot compute a representation of $R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}}$ with respect to g and PK_k.) We show below that when BadRS occurs $\mathcal{B}$ can succeed in the AOMDL experiment.

If BadRS does not occur, then for every (i, k) either $R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}} = 1$ or, using Equation (8),

$$R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}} = R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma_{i,k} a_{i,k}} = g^{z_{i,k}} = \left(g^{z_{i,k}} \mathsf{PK}_k^{-c_{i,k} \lambda_{i,k}} \right)^{\gamma_{i,k}}. \tag{9}$$

Since (R^*, z^*) is a valid signature,

$$g^{z^*} \mathsf{PK}^{-c^*} = g^{\delta} \mathsf{PK}^{\beta} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} \prod_{i=1}^{q_s} R_{i,1}^{\gamma_{i,1}} S_{i,1}^{\gamma'_{i,1}} \cdots R_{i,n}^{\gamma_{i,n}} S_{i,n}^{\gamma'_{i,n}}. \tag{10}$$

Thus, $\mathcal{B}$ can replace $R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma'_{i,k}}$ with $\left(g^{z_{i,k}} \mathsf{PK}_k^{-c_{i,k} \lambda_{i,k}} \right)^{\gamma_{i,k}}$ in Equation (10) for all $i \in [q_s]$ and $k \in \mathsf{hon}$ with $(\gamma_{i,k}, \gamma'_{i,k}) \neq (0, 0)$ to obtain

$$\begin{aligned}
g^{z^*} \mathsf{PK}^{-c^*} &= g^{\delta} \mathsf{PK}^{\beta} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} \prod_{i=1}^{q_s} R_{i,1}^{\gamma_{i,1}} S_{i,1}^{\gamma'_{i,1}} \cdots R_{i,n}^{\gamma_{i,n}} S_{i,n}^{\gamma'_{i,n}} \\
&= g^{\delta} \mathsf{PK}^{\beta} \prod_{k \in \mathsf{hon}} \mathsf{PK}_k^{\beta_k} \prod_{j \in \mathsf{cor}} \mathsf{PK}_j^{\beta_j} \prod_{i=1}^{q_s} \Big(\prod_{k \in \mathsf{hon}} R_{i,k}^{\gamma_{i,k}} S_{i,k}^{\gamma_{i,k} a_{i,k}} \prod_{j \in \mathsf{cor}} R_{i,j}^{\gamma_{i,j}} S_{i,j}^{\gamma'_{i,j}} \Big) \\
&= g^{\delta} \mathsf{PK}^{\beta} \prod_{k \in \mathsf{hon}} \mathsf{PK}_k^{\beta_k} \prod_{j \in \mathsf{cor}} \mathsf{PK}_j^{\beta_j} \prod_{i=1}^{q_s}
\end{aligned}$$

[6] Note that BadRS is referred to as a bad event because it represents a failure case for the reduction to the LDVR problem, as we will see later.

$$\cdot \Big(\prod_{k \in \mathsf{hon}} g^{z_{i,k}\gamma_{i,k}} \mathsf{PK}_k^{-c_{i,k}\lambda_{i,k}\gamma_{i,k}} \prod_{j \in \mathsf{cor}} R_{i,j}^{\gamma_{i,j}} S_{i,j}^{\gamma'_{i,j}} \Big)$$
$$= g^{\delta + \sum_{i=1}^{q_s} (\sum_{k \in \mathsf{hon}} z_{i,k}\gamma_{i,k} + \sum_{j \in \mathsf{cor}} (\gamma_{i,j} r_{i,j} + \gamma'_{i,j} s_{i,j}))}$$
$$\cdot \mathsf{PK}^{\beta} \prod_{j \in \mathsf{cor}} \mathsf{PK}_j^{\beta_j} \prod_{k \in \mathsf{hon}} \mathsf{PK}_k^{\beta_k - \sum_{i \in [q_s], \gamma_{i,k} \neq 0} c_{i,k}\lambda_{i,k}\gamma_{i,k}} . \tag{11}$$

(Note that it is enough to require $\gamma_{i,k} \neq 0$ since if $\gamma_{i,k} = 0$ then—because (i,k) is good—we must have $\gamma'_{i,k} = a_{i,k}\gamma_{i,k} = 0$.) Let $\vec{v}_k := (1, k, \ldots, k^t) \in \mathbb{Z}_p^{t+1}$ for $k \in [0..n]$ and define

$$\vec{v}^* := (c^* + \beta)\vec{v}_0 + \sum_{k \in \mathsf{cor}} \beta_k \cdot \vec{v}_k + \sum_{k \in \mathsf{hon}} \left(\beta_k - \sum_{i \in [q_s], \gamma_{i,k} \neq 0} c_{i,k}\lambda_{i,k}\gamma_{i,k} \right) \cdot \vec{v}_k \, . \tag{12}$$

Since we can write $\mathsf{PK} = X_0^{v_{0,0}} \cdot \prod_{j \in [t]} \hat{X}_j^{v_{0,j}}$ and $\mathsf{PK}_k = X_0^{v_{k,0}} \cdot \prod_{j \in [t]} \hat{X}_j^{v_{k,j}}$ for all $k \in [n]$, Equation (11) implies

$$g^{\gamma^*} = X_0^{v_0^*} \cdot \prod_{j \in [t]} \hat{X}_j^{v_j^*} \, , \tag{13}$$

where $\gamma^* := z^* - \delta - \sum_{i=1}^{q_s} (\sum_{k \in \mathsf{hon}} z_{i,k}\gamma_{i,k} - \sum_{j \in \mathsf{cor}} (\gamma_{i,j} r_{i,j} + \gamma'_{i,j} s_{i,j}))$ is a scalar known by $\mathcal{B}$. Let Badv be the event that BadRS does not occur and $\vec{v}^* \notin \mathsf{span}(\{\vec{v}_k\}_{k \in \mathsf{cor}})$. We show below that if Badv occurs then $\mathcal{B}$ can succeed in the AOMDL experiment.

Completing the description of $\mathcal{B}$, we show next how $\mathcal{B}$ can succeed in the AOMDL experiment when $\mathcal{A}$ succeeds and either BadRS or Badv occurs. Namely, we show that $\mathcal{B}$ can compute the discrete logarithms of every group element it was given at the outset of the experiment, while making $t + 2q_s$ queries to $\mathcal{O}^{\mathsf{dl}}$.

BadRS occurs. If BadRS occurs then there exist indices $(\hat{i}, \hat{k}) \in [q_s] \times \mathsf{hon}$ such that $(\gamma_{\hat{i},\hat{k}}, \gamma'_{\hat{i},\hat{k}}) \neq (0,0)$ and either $\mathsf{Q}_{\mathsf{st}}[\hat{k}, \hat{i}, 2] = \perp$ or $\gamma'_{\hat{i},\hat{k}} \neq \gamma_{\hat{i},\hat{k}} a_{\hat{i},\hat{k}}$. Let $i' \in [q_s]$ be such that $(R_{\hat{i},\hat{k}}, S_{\hat{i},\hat{k}}) = (X_{i'}, X'_{i'})$.

First of all, $\mathcal{B}$ can compute $(x_0, \hat{x}_1, \ldots, \hat{x}_t)$ using $\{\mathsf{sk}_k\}_{k \in \mathsf{cor}}$ and an additional $t + 1 - |\mathsf{cor}|$ queries to $\mathcal{O}^{\mathsf{dl}}$. This accounts for a total of $t + 1$ queries (since $\mathcal{B}$ already used $|\mathsf{cor}|$ queries to learn $\{\mathsf{sk}_k\}_{k \in \mathsf{cor}}$).

Then, for each (X_i, X'_i) that corresponds to nonce commitments of an corrupted signer, $\mathcal{B}$ knows (x_i, x'_i) from its simulation of the $\mathsf{Corrupt}$ oracle which involved two queries to $\mathcal{O}^{\mathsf{dl}}$. For each $(X_i, X'_i) \neq (X_{i'}, X'_{i'})$ that corresponds to nonce commitments of an honest signer, $\mathcal{B}$ can compute (x_i, x'_i) using either one $\mathcal{O}^{\mathsf{dl}}$ query if it already made a $\mathcal{O}^{\mathsf{dl}}$ query for (X_i, X'_i) as part of simulating a second-round signing query, or using two $\mathcal{O}^{\mathsf{dl}}$ queries otherwise. (This is done exactly as when $\mathcal{B}$ simulates the $\mathsf{Corrupt}$ oracle.) Therefore, $\mathcal{B}$ makes two $\mathcal{O}^{\mathsf{dl}}$ queries for each $(X_i, X'_i) \neq (X_{i'}, X'_{i'})$.

Finally, we show that $\mathcal{B}$ needs only one $\mathcal{O}^{\mathsf{dl}}$ query to compute $(x_{i'}, x'_{i'})$ corresponding to $(X_{i'}, X'_{i'})$. Assume $\gamma_{\hat{i},\hat{k}} \neq 0$. (The case of $\gamma'_{\hat{i},\hat{k}} \neq 0$ is analogous.) If $\mathcal{B}$ has not yet made any $\mathcal{O}^{\mathsf{dl}}$ query involving $(X_{i'}, X'_{i'})$, then $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ on $X'_{i'}$ to obtain $x'_{i'}$, and then derives $x_{i'}$ from Equation (10) since $\gamma_{\hat{i},\hat{k}} \neq 0$ and $\mathcal{B}$ knows the discrete logarithms of all group elements in that equation except for $R_{\hat{i},\hat{k}}$. Otherwise, $\mathcal{B}$ knows $g^{z_{\hat{i},\hat{k}} - c_{\hat{i},\hat{k}} \hat{\lambda}_{\hat{i},\hat{k}} \mathsf{sk}_{\hat{k}}} = X_{i'}(X'_{i'})^{a_{\hat{i},\hat{k}}}$ (cf. Equation (9)). Since $\gamma'_{\hat{i},\hat{k}} \neq \gamma_{\hat{i},\hat{k}} a_{\hat{i},\hat{k}}$, we see that $\mathcal{B}$ can derive $(x_{i'}, x'_{i'})$ using Equation (10), because $\mathcal{B}$ knows the discrete logarithms of all group elements in that equation except for $(R_{\hat{i},\hat{k}}, S_{\hat{i},\hat{k}})$.

Badv occurs. In this case, $\mathcal{B}$ learns a linear relation among $g, X_0, \hat{X}_1, \ldots, \hat{X}_t$ (cf. Equation (13)) that is linearly independent of all of the $\mathcal{O}^{\mathsf{dl}}$ queries it made to simulate $\mathsf{Corrupt}$ queries. $\mathcal{B}$ can thus compute $(x_0, \hat{x}_1, \ldots, \hat{x}_t)$ by making an additional $t - |\mathsf{cor}|$ queries to $\mathcal{O}^{\mathsf{dl}}$. Then, as in the previous case, $\mathcal{B}$ can compute (x_i, x'_i) corresponding to each (X_i, X'_i) using a total of two queries to $\mathcal{O}^{\mathsf{dl}}$ for each such pair.

Since $\mathcal{B}$ perfectly simulates $\mathsf{Expt}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{TS}}$, we have

$$\mathsf{Adv}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{TS}}(\kappa, n, t, t_c) \leq \mathsf{Adv}^{(2q_s + t + 1)\text{-}\mathsf{aomdl}}_{\mathcal{B}}(\kappa) + \Pr[\mathsf{Succ} \wedge \neg(\mathsf{BadRS} \vee \mathsf{Badv})],$$

where Succ is the event that $\mathcal{A}$ succeeds. Lemma 1 completes the proof. □

Lemma 1. *For $\mathcal{A}$ described as in Theorem 2, there is a PPT adversary $\mathcal{C}$ for the (t_c, t, n)-LDVR problem making at most q queries to its oracle and running in time roughly the same as $\mathcal{A}$ such that*

$$\mathsf{Adv}^{(t_c,\, t,\, n)\text{-}\mathsf{ldvr}}_{\mathcal{C}}(\kappa) \geq \Pr[\mathsf{Succ} \wedge \neg(\mathsf{BadRS} \vee \mathsf{Badv})] - \frac{2(nq)^2}{2^\kappa}\,. \tag{14}$$

Proof. We assume, without loss of generality, that $\mathcal{A}$ makes a query to $\mathsf{H}_{\mathsf{sig}}$ on its forgery $(\tilde{R}^*, \mathsf{PK}, m^*)$ and that it always queries $\mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (resp. $\mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ for $\mathsf{FROST1}$) before querying $\mathcal{O}^{\mathsf{Sign}_1}(k, \mathcal{S}, m, \mathcal{PM}_1)$ for some $k \in \mathcal{S}$. This adds $q_s + 1$ additional random-oracle queries. Then, let $q = q_h + q_s + 1$.

Construction of $\mathcal{C}$. To start with, $\mathcal{C}$ samples $(\mathbb{G}, p, g) \leftarrow\!\!\$\ \mathsf{GrpGen}(\kappa)$ and sends p back to the game (t_c, t, n)-LDVR. Then, $\mathcal{C}$ runs $\mathcal{A}$ by simulating the game $\mathsf{Expt}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{FROST}}(\kappa, n, t, t_c)$ faithfully except that $\mathcal{C}$ initializes tables $\mathsf{Flag}_{\mathsf{non}}, \mathsf{Q}_{\mathsf{non}}, \mathsf{Q}_{\mathsf{sig}}$ to empty and simulates queries to the random oracles $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$ as follows.

$\mathsf{H}_{\mathsf{non}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on T, for $\mathsf{FROST1}$, $\mathcal{C}$ parses $T = (k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, and for $\mathsf{FROST2}$ and $\mathsf{FROST3}$, $\mathcal{C}$ parses $T = (\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$. (Here w.l.o.g. we assume T can be parsed correctly and it is for PK, since otherwise the query on a malformed T would not help the adversary to win the game.) Then, $\mathcal{C}$ checks if $\mathsf{Q}_{\mathsf{non}}[T] = \bot$. If not, $\mathcal{C}$ returns $\mathsf{Q}_{\mathsf{non}}[T]$.
Otherwise, for $\mathsf{FROST1}$, $\mathcal{C}$ does the following:

1. For all $i \in \mathcal{S}$, $\mathcal{C}$ samples $a_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and stores $\mathsf{Q}_{\mathsf{non}}[(i, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)] = a_i$. If $\mathsf{Flag}_{\mathsf{non}}[a_i] \neq \perp$, $\mathcal{C}$ aborts, and we refer to this bad event as BadNon. Otherwise, $\mathcal{C}$ sets $\mathsf{Flag}_{\mathsf{non}}[a_i] := 1$.
2. Then, $\mathcal{C}$ computes $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$.
3. Finally, $\mathcal{C}$ returns a_k.

For FROST2 and FROST3, $\mathcal{C}$ does the following:
1. $\mathcal{C}$ samples $a \leftarrow\!\!\$\ \mathbb{Z}_p$ and stores $\mathsf{Q}_{\mathsf{non}}[T] := a$.
2. If $\mathsf{Flag}_{\mathsf{non}}[a] \neq \perp$, $\mathcal{C}$ aborts, and we refer to this bad event as BadNon. Otherwise, $\mathcal{C}$ sets $\mathsf{Flag}_{\mathsf{non}}[a] := 1$.
3. $\mathcal{C}$ computes $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$.
4. Finally, $\mathcal{C}$ returns a.

$\mathsf{H}_{\mathsf{sig}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{R}, \mathsf{PK}, m)$, if $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, \tilde{R})] \neq \perp$, $\mathcal{C}$ returns $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, \tilde{R})]$. (Similar to $\mathsf{H}_{\mathsf{non}}$, here we only consider queries for PK, since otherwise the query would not help the adversary to win the game.) Otherwise, $\mathcal{C}$ queries $\mathcal{O}^H$ to obtain a challenge c on a vector $\vec{\alpha}$, sets $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, \tilde{R})] := c$, and returns c, where $\vec{\alpha}$ is computed as follows.

1. $\mathcal{C}$ learns a representation of $\tilde{R}$ from $\mathcal{A}$ based on all group elements $\mathcal{A}$ had seen so far:

$$\tilde{R} = g^{\delta} \mathsf{PK}^{\beta_0} \prod_{k=1}^{n} \mathsf{PK}_k^{\beta_k} \prod_{i=1}^{q_s} R_{i,1}^{\gamma_{i,1}} S_{i,1}^{\gamma'_{i,1}} \cdots R_{i,n}^{\gamma_{i,n}} S_{i,n}^{\gamma'_{i,n}} , \tag{15}$$

   where $(R_{i,k}, S_{i,k})$ denotes the output of the i-th query to $\mathcal{O}^{\mathsf{Sign}}(k)$.
2. $\mathcal{C}$ sets $\alpha_k := \beta_k$ for $k \in [0..n]$.
3. For each $i \in [q_s]$ and $k \in \mathsf{hon}$ with $\gamma_{i,k} \neq 0$, $\mathcal{C}$ updates α_k if the following two conditions hold:
   (a) There exists a $\mathsf{H}_{\mathsf{non}}$ query T such that $\gamma'_{i,k} = \mathsf{Q}_{\mathsf{non}}[T] \cdot \gamma_{i,k}$. Since BadNon does not occur (otherwise, the game aborts), such T is unique. For FROST1, $\mathcal{C}$ parses $T = (k, \mathsf{PK}, \mathcal{S}, \hat{m}, \mathcal{PM}_1)$.

   For FROST2 and FROST3, $\mathcal{C}$ parses $T = (\mathsf{PK}, \mathcal{S}, \hat{m}, \mathcal{PM}_1)$.

   (b) $(m, \tilde{R}) \neq (\hat{m}, \hat{\tilde{R}})$, where $\hat{\tilde{R}} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, \hat{m}, \mathcal{PM}_1)$.

   If the above conditions hold, $\mathcal{C}$ sets $\alpha_k := \alpha_k - \gamma_{i,k} \lambda_{i,\mathcal{S}} \mathsf{Q}_{\mathsf{sig}}[\mathsf{PK}, \hat{m}, \hat{\tilde{R}}]$.
4. For each $i \in [q_s]$ and $k \in \mathsf{hon}$ with $\gamma_{i,k} \neq 0$, $\mathcal{C}$ sets $\mathsf{Flag}_{\mathsf{non}}[\gamma'_{i,k}/\gamma_{i,k}] := 1$.

Output. At the end of the game, after receiving $(m^*, \sigma^* = (R^*, z^*))$ from $\mathcal{A}$, $\mathcal{C}$ aborts if $\mathcal{A}$ does not win the unforgeability game or $(\mathsf{Badv} \vee \mathsf{BadRS})$ occurs. Otherwise, $\mathcal{C}$ looks up $\mathcal{A}$'s query to $\mathsf{H}_{\mathsf{sig}}$ on (R^*, PK, m^*) and its corresponding query to $\mathcal{O}^H$ on the vector $\vec{\alpha}^*$ such that $\mathsf{H}_{\mathsf{sig}}(R^*, \mathsf{PK}, m^*)$ is set to $\mathcal{O}^H(\vec{\alpha}^*)$. Suppose it is the i-th $\mathcal{O}^H$ query, $\mathcal{C}$ returns (cor, i).

Analysis of $\mathcal{C}$. We first show $\mathcal{C}$ wins the game LDVR if $\mathcal{C}$ does not abort. Since $\mathcal{A}$ corrupts at most t_c parties, we know $|\mathsf{cor}| \leq t$. Let $c^* := \mathcal{O}^H(\vec{\alpha}^*)$ and $\vec{w}^* := c^*\vec{v}_0 + \sum_{k\in[n]} \alpha_{i,k}\vec{v}_k$. Since $(\mathsf{Badv} \vee \mathsf{BadRS})$ does not occur, $\vec{v}^* \in \mathsf{span}(\{\vec{v}_k\}_{k\in\mathsf{cor}})$. We just need to show that $\vec{w}^* - \vec{v}^* \in \mathsf{span}(\{\vec{v}_k\}_{k\in\mathsf{cor}})$.

When A made the $\mathsf{H}_{\mathsf{sig}}$ query (R^*, PK, m^*), it provided a presentation as shown in Equation (7). Denote hon (resp. cor) as the set of honest (resp. corrupted) parties after $\mathcal{A}$ returned, and denote hon' (resp. cor') as the set of honest (resp. corrupted) parties when $\mathcal{A}$ made the $\mathsf{H}_{\mathsf{sig}}$ query. From the execution of $\mathcal{C}$,

$$\vec{w}^* := (c^* + \beta)\vec{v}_0 + \sum_{k\in\mathsf{hon}'} \left(\beta_k - \sum_{i\in[q_s],\gamma_{i,k}\neq 0} b_{i,k} \right) \cdot \vec{v}_k \ , \tag{16}$$

where $b_{i,k} := \gamma_{i,k}\lambda_{i,\mathcal{S}}\mathsf{Q}_{\mathsf{sig}}[\mathsf{PK}, \hat{m}, \hat{\tilde{R}}]$ denotes the decrement applied to α_k in the last line of Step 3 on behalf of (i,k) and $b_{i,k} := 0$ if one of the in Step 3 does not hold. By Equation (12),

$$\begin{aligned} \vec{w}^* - \vec{v}^* := &- \sum_{k\in\mathsf{cor}} \beta_k \cdot \vec{v}_k - \sum_{k\in\mathsf{hon}'\setminus\mathsf{hon}} \left(\sum_{i=1}^{q_s} b_{i,k} \right) \cdot \vec{v}_k \\ &+ \sum_{k\in\mathsf{hon}} \left(\sum_{i\in[q_s],\gamma_{i,k}\neq 0} (c_{i,k}\lambda_{i,k}\gamma_{i,k} - b_{i,k}) \right) \cdot \vec{v}_k \ . \end{aligned} \tag{17}$$

Since $\mathsf{hon}' \setminus \mathsf{hon} \in \mathsf{cor}$, we have $\sum_{k\in\mathsf{cor}} \beta_k \cdot \vec{v}_k + \sum_{k\in\mathsf{hon}'\setminus\mathsf{hon}} (\sum_{i=1}^{q_s} b_{i,k}) \cdot \vec{v}_k \in \mathsf{span}(\{\vec{v}_k\}_{k\in\mathsf{cor}})$, and thus it is left to show $b_{i,k} = c_{i,k}\lambda_{i,k}\gamma_{i,k}$ for all $(i,k) \in [q_s] \times \mathsf{hon}$ with $\gamma_{i,k} \neq 0$.

For each $(i,k) \in [q_s] \times \mathsf{hon}$ with $\gamma_{i,k} \neq 0$, since BadRS does not occur, there exists a query $(k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$ to $\mathcal{O}^{\mathsf{Sign}_2}$ such that $a_{i,k} = \mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (for $\mathsf{FROST1}$, $a_{i,k} = \mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$), $c_{i,k} = \mathsf{H}_{\mathsf{sig}}(\mathsf{PK}, m, \tilde{R})$ with $\tilde{R} := \mathsf{CompR}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, $\lambda_{i,k} = \lambda_k^{\mathcal{S}}$, and $\gamma'_{i,k} = a_{i,k}\gamma_{i,k}$. We now show that when A made the $\mathsf{H}_{\mathsf{sig}}$ query $(\mathsf{PK}, m^*, \tilde{R}^*)$, both conditions in Step 3 hold:

Condition (3a) holds. Let $T = (\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (resp. $(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ for $\mathsf{FROST1}$), and we have $\gamma'_{i,k} = \mathsf{H}_{\mathsf{non}}[T]\gamma_{i,k}$. Also, the query T to $\mathsf{H}_{\mathsf{non}}$ must be made before A made the $\mathsf{H}_{\mathsf{sig}}$ query $(\mathsf{PK}, m^*, \tilde{R}^*)$. Otherwise, BadNon would occur since $\mathsf{Flag}_{\mathsf{non}}[\gamma'_{i,k}/\gamma_{i,k}(= \mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1))]$ is set to 1 after the $\mathsf{H}_{\mathsf{sig}}$ query (see Step 4), which contradicts the fact that $\mathcal{C}$ does not abort. Therefore, the first condition holds.

Condition (3b) holds. Since $\mathcal{A}$ wins the unforgeability game, we have $m \neq m^*$, which implies the second condition holds.

Therefore, from the description of Step 3, $b_{i,k} = \mathsf{H}_{\mathsf{sig}}(\mathsf{PK}, m, \tilde{R})\lambda_i^{\mathcal{S}}\gamma_{i,k} = c_{i,k}\lambda_{i,k}\gamma_{i,k}$.

Since $\mathcal{C}$ simulates $\mathsf{Expt}^{\mathsf{adp\text{-}TS\text{-}UF}}_{\mathcal{A},\mathsf{FROST}}$ perfectly except for an abort if BadNon occurs,

$$\mathsf{Adv}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}_{\mathcal{C}}(\kappa) \geq \Pr[\mathsf{Succ} \wedge \neg(\mathsf{BadRS} \vee \mathsf{Badv})] - \Pr[\mathsf{BadNon}] \ .$$

For each $\mathsf{H}_{\mathsf{non}}$ query, since the number of values marked by $\mathsf{Flag}_{\mathsf{non}}$ is at most $2nq$,[7] the probability that the sampled a collides with one of the values is at most $2nq/p$. (For FROST1, the probability that one of the sampled a_i collides with one of the values is at most $2n^2q/p$). Therefore, $\Pr[\mathsf{BadNon}] \leq 2(nq)^2/p$, which concludes the proof of the lemma. □

4.4 Hardness of LDVR

In this section, we provide some basic intuition about the hardness of the LDVR problem. First note that $\{\vec{v}_i\}_{i\in \mathrm{CS}}$ can never be a basis of $\mathbb{Z}_p^{t+1}$ (since $t_c < t+1$), and therefore it is not clear whether the problem is solvable. (Indeed, we show below that for some settings of the parameters even an unbounded adversary cannot solve it except with negligible probability.) In particular, note that $\vec{v}_0 \notin \mathsf{span}(\{\vec{v}_i\}_{i\in \mathrm{CS}})$ for any $\mathrm{CS} \subset [n]$ of size t_c, and so if we fix such a CS in advance then for any query $c := \mathcal{O}(\vec{\alpha})$ made by $\mathcal{A}$, the probability that $\vec{w} = c \cdot \vec{v}_0 + \sum_{i=0}^{n} \alpha[i] \cdot \vec{v}_i$ lies in $\mathsf{span}(\{\vec{v}_i\}_{i\in \mathrm{CS}})$ is at most $1/p$. Note, however, that $\mathcal{A}$ can choose CS *after* observing c, and there are $\binom{n}{t_c}$ possibilities for CS. Nevertheless, this already shows that the (t_c, t, n)-LDVR problem is unconditionally hard (for an algorithm making polynomially many queries to its oracle) if $q\binom{n}{t_c} \cdot 2^{-\kappa}$ is negligible. We show additional results about unconditional hardness next, which extend this result to a general corruption threshold $t_c < t$.

Unconditional hardness. We prove unconditional hardness of the (t_c, t, n)-LDVR problem for various settings of t_c, t, n. Specifically, we provide an upper bound on the probability with which even an *unbounded* algorithm $\mathcal{A}$ can solve the LDVR problem, as a function of t_c, t, n and the number of oracle queries q made by $\mathcal{A}$. We also give numerical examples of our bound.

In the following, we define $(n)_k = n \cdot (n-1) \cdots (n-k+1)$ for $k \geq 1$, with $(n)_k = 1$ if $k \leq 0$.

Theorem 3 (Unconditional hardness of LDVR problem). *Let $t_c \leq t < n$ be integers. Then, for any algorithm $\mathcal{A}$ making $q = q(\kappa)$ queries to its oracle,*

$$\mathsf{Adv}_{\mathcal{A}}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}(\kappa) \leq q \cdot \frac{(n)_{2t_c-t}}{(t_c)_{2t_c-t}} \cdot \frac{1}{2^{\kappa}} \,.$$

Discussion of the bound. Before proving Theorem 3, we make some observations. First note that the inefficient attack from prior work [21] can be seen as an algorithm solving the LDVR problem (as discussed in Section 4.1), and is thus subject to our bound. For $t_c = t$, that prior work shows an algorithm that succeeds with probability $\frac{1}{2}$ if $\binom{n}{t} > 2^{\kappa}$. This implies that our upper bound is tight when $t_c = t$, since then $2t_c - t = t$ and

$$\frac{(n)_t}{(t)_t} = \frac{(n)_t}{t!} = \binom{n}{t}.$$

[7] At most n values can be marked for each $\mathsf{H}_{\mathsf{non}}$ query or $\mathsf{H}_{\mathsf{sig}}$ query.

	δ					
(t, n)	0.55	0.60	0.70	0.80	0.90	1.00
$(32, 128)$	11.77	17.61	35.09	58.85	77.85	100.22
$(64, 128)$	11.71	23.06	48.54	69.82	95.66	124.17
$(96, 128)$	13.51	25.95	45.84	64.55	79.24	100.22
$(64, 256)$	17.79	35.45	76.57	112.48	157.12	203.57
$(128, 256)$	23.51	50.07	97.52	140.36	188.59	251.67
$(192, 256)$	27.10	49.67	92.02	129.66	161.01	203.57
$(256, 1024)$	77.46	154.33	301.98	458.68	619.17	825.63
$(512, 1024)$	102.08	197.18	384.26	570.84	761.94	1018.67

Table 1. Numerical values from Theorem 3. Selected values of the function $f_{t,n}(t_c = \lfloor \delta \cdot t \rceil) = \log\left(\frac{(n)_{2t_c - t}}{(t_c)_{2t_c - t}}\right)$, where $\lfloor x \rceil$ denotes rounding x to the nearest integer.

For $t_c < t$, the algorithm from prior work succeeds with probability lower than our upper bound, and it remains open whether our bound is tight in that range. Also, compared to the trivial upper bound $q\binom{n}{t_c} \cdot 2^{-\kappa}$ mentioned above, the improved upper bound is strictly better since $\binom{n}{t_c} = \frac{(n)_{t_c}}{(t_c)_{t_c}} < \frac{(n)_{2t_c - t}}{(t_c)_{2t_c - t}}$ given $t_c < t$. Moreover, improvement is significant when t_c is far from t. For example, if $t_c = 2t/3$, the trivial upper bound gives $q\frac{(n)_{2t/3}}{(2t/3)_{2t/3}} \cdot 2^{-\kappa}$, while the improved bound gives $q\frac{(n)_{t/3}}{(2t/3)_{t/3}} \cdot 2^{-\kappa}$, which is better by a factor of $\frac{(n-t/3)_{t/3}}{(t/3)_{t/3}}$.

Observe also that for $t_c \leq t/2$, the above bound is $q/2^{\kappa}$; when plugged into Theorem 2, this implies (as implicitly confirmed by Theorem 1) that the LDVR problem is not relevant in that case.

Remark 1 (Comparison with guessing argument). It is also interesting to compare what we get by composing Theorems 2 and 3 with the folklore approach of lifting static to adaptive security via guessing the corruption set. The least favorable comparison is for the case of full corruption $t_c = t$, where our approach results in an additive term of $q\binom{n}{t}/2^{\kappa}$ with q corresponding to the number of random-oracle queries made by the attacker. In contrast, employing the guessing strategy results in a multiplicative term of $\binom{n}{t}$. Given current concrete bounds for the static security of FROST (e.g., [8]), this means that if we can prove that the advantage of a static adversary is at most ϵ, we can also show that the advantage of an adaptive one is at most $\binom{n}{t}\epsilon$. The best known ϵ is effectively the bound of Theorem 2 without the LDVR advantage, and once one factors in concrete security bounds for AOMDL (e.g., from the GGM analysis [8]), the final concrete bound would be looser. The improvement is even more important for $t_c < t$, since our bound is smaller than $\binom{n}{t_c}/2^{\kappa}$.

Table 1 gives numerical values for $f_{t,n}(t_c = \delta \cdot t) = \log\left(\frac{(n)_{2t_c-t}}{(t_c)_{2t_c-t}}\right)$ that provide a quantitative understanding of the bound from Theorem 3. For example, we see that if $n = 128$, $t = t_c = 96$, and $p > 2^{256}$, Theorem 3 gives

$$\mathsf{Adv}_{\mathcal{A}}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}(\kappa) \leq q \cdot 2^{100.22-256} = \frac{q}{2^{155.78}} \ .$$

We now prove Theorem 3:

Proof. We assume $\mathcal{A}$ makes exactly q queries, and let p be the prime output by $\mathcal{A}$. For $i \in [q]$, let Win_i be the event that the vector $\vec{w}_i = c_i \cdot \vec{v}_0 + \sum_{j=0}^{n} \vec{\alpha}_i[j] \cdot \vec{v}_j$ implicitly defined by the ith query $\vec{\alpha}_i$ returning c_i belongs to $\mathsf{Span}(\{\vec{v}_j\}_{j\in S})$ for some $S \subseteq [n]$ with $|S| = t_c$. Note that if the game outputs true then Win_i must occur for some $i \in [q]$, and so a union bound implies

$$\mathsf{Adv}_{\mathcal{A}}^{(t_c,\,t,\,n)\text{-}\mathsf{ldvr}}(\kappa) \leq \sum_{i=1}^{q} \Pr\left[\mathsf{Win}_i\right]. \tag{18}$$

Fix some $i \in [q]$, and write

$$\vec{w}_i = (c_i + \vec{\alpha}_i[0]) \cdot \vec{v}_0 + \vec{u}_i \ ,$$

where $\vec{u}_i = \sum_{j=1}^{n} \vec{\alpha}_i[j] \cdot \vec{v}_j$. We call a set $S \subseteq [n]$ of size t_c *good* if $\vec{u}_i \in \mathsf{Span}(\{\vec{v}_j\}_{j\in S} \cup \{\vec{v}_0\})$. In particular, for every good set S, there exist coefficients $(\beta_j^S)_{j\in S\cup\{0\}}$ such that $\vec{u}_i = \beta_0^S \vec{v}_0 + \sum_{j\in S} \beta_j^S \cdot \vec{v}_j$.

Claim. Win_i occurs if and only if $c_i + \vec{\alpha}_i[0] + \beta_0^S = 0$ for some good set S.

Proof. If $c_i + \vec{\alpha}_i[0] + \beta_0^S = 0$ for some good set S, then $\vec{w}_i \in \mathsf{Span}(\{\vec{v}_j\}_{j\in S})$ (as the $\vec{v}_0$ component cancels out) and thus Win_i occurs. Conversely, if Win_i occurs then there exists $S \subseteq [n]$ with $|S| = t_c$ such that $\vec{w}_i \in \mathsf{Span}(\{\vec{v}_j\}_{j\in S})$. We claim that S must be good. To see this, let $S' \supseteq S$ be smallest such that $\vec{u}_i \in \mathsf{Span}(\{\vec{v}_j\}_{j\in S'} \cup \{\vec{v}_0\})$. If $S' \neq S$, then $\vec{w}_i$ cannot be in $\mathsf{Span}(\{\vec{v}_j\}_{j\in S})$, as it would have at least one non-zero component aligned with $\vec{v}_j$ for $j \notin S \cup \{0\}$. Further, if $c_i + \vec{\alpha}_i[0] + \beta_0^S \neq 0$, it would also have a component aligned with $\vec{v}_0$, and thus not be in $\mathsf{Span}(\{\vec{v}_j\}_{j\in S})$. □

Let Good be the collection of good sets. Because Good is completely determined by $\vec{\alpha}_i$, and c_i is uniform and independent of $\vec{\alpha}_i$, we have

$$\Pr\left[\mathsf{Win}_i\right] = \Pr\left[\exists S \in \mathsf{Good} \ : \ c_i + \vec{\alpha}_i[0] + \beta_0^S = 0\right] \leq \frac{|\mathsf{Good}|}{p} \ . \tag{19}$$

It remains to upper bound $|\mathsf{Good}|$. To this end, fix distinct $S, S' \in \mathsf{Good}$. Then, we can write $\vec{w}_i$ in two different ways, namely

$$(c_i + \vec{\alpha}_i[0] + \beta_0^S)\vec{v}_0 + \sum_{j\in S} \beta_j^S \cdot \vec{v}_j \ = \ \vec{w}_i \ = \ (c_i + \vec{\alpha}_i[0] + \beta_0^{S'})\vec{v}_0 + \sum_{j\in S'} \beta_j^{S'} \cdot \vec{v}_j$$

which implies in particular that

$$(\beta_0^S - \beta_0^{S'})\vec{v}_0 + \sum_{j \in S} \beta_j^S \cdot \vec{v}_j - \sum_{j \in S'} \beta_j^{S'} \cdot \vec{v}_j = \vec{0} \,.$$

Thus $|S \cup S'| \geq t+1$, for otherwise we would have expressed $\vec{0}$ as a non-zero linear combination of the vectors $\{\vec{v}_j\}_{j \in S \cup S'} \cup \{\vec{v}_0\}$, contradicting their linear independence. Therefore, if we define Good' to be the set of subsets $S \subseteq [n]$ of size t_c such that $|S \cup S'| \geq t+1$ for any distinct $S, S' \in \mathsf{Good}'$, we have $|\mathsf{Good}| \leq |\mathsf{Good}'|$. It is thus enough to upper bound the latter.

If $t_c \leq t/2$, then Good' includes at most one set, which proves the claimed upper bound in that case. To obtain a bound for $t_c > t/2$, we assign every set $S \in \mathsf{Good}'$ to its characteristic vector $\vec{c}_S \in \{0,1\}^n$, which has weight exactly t_c, to build a code $\mathcal{C} = \{\vec{c}_S \ : \ S \in \mathsf{Good}'\} \subseteq \{0,1\}^n$. We observe that for any two $S, S' \in \mathsf{Good}'$, the Hamming distance of $\vec{c}_S$ and $\vec{c}_{S'}$ is

$$\begin{aligned} \Delta(\vec{c}_S, \vec{c}_{S'}) &= |S \oplus S'| = |S| + |S'| - 2|S \cap S'| \\ &= |S| + |S'| - 2(|S| + |S'| - |S \cup S'|) \\ &= 2|S \cup S'| - |S| - |S'| \geq 2(t + 1 - t_c) \,, \end{aligned}$$

where $S \oplus S'$ denotes symmetric difference of the two sets. By invoking the Johnson bound (Lemma 2) with $e = t + 1 - t_c$, we obtain the desired bound for $|\mathsf{Good}'| = |\mathcal{C}|$, since $w - e + 1 = t_c - t - 1 + t_c + 1 = 2t_c - t$. Combining Equations (18) and (19), and using the fact that $p \geq 2^\kappa$, concludes the proof. □

For completeness, we provide a proof of the following in Appendix B.

Lemma 2 (Johnson Bound). *For any integers $e \leq w \leq n$, let $\mathcal{C} \subseteq \{0,1\}^n$ be a binary code such that every codeword $\vec{c} \in \mathcal{C}$ has weight w, and any two distinct $\vec{c}, \vec{c}' \in \mathcal{C}$ have Hamming distance $\Delta(\vec{c}, \vec{c}') \geq 2e$. Then,*

$$|\mathcal{C}| \leq \prod_{i=0}^{w-e} \frac{n-i}{w-i} = \frac{(n)_{w-e+1}}{(w)_{w-e+1}} \,.$$

Computational hardness. We conjecture that the LDVR problem is computationally hard for any $t_c \leq t < n$. In Section 4.5 we show that one natural algorithmic strategy for solving the problem results in a modular subset-sum problem which is hard for current algorithms. Of course, further cryptanalysis of the LDVR problem is needed.

4.5 Cryptanalysis of the LDVR Problem

In order to support a plausible conjecture that LDVR is hard for any choice of $t_c \leq t < n$, we describe here a cryptanalytic attack strategy for the LDVR problem which targets the maximum corruption case $t_c = t$, under the assumption that $n \geq t + 2$. Clearly, $t_c = t$ is the easiest case to attack, since an attack for a

lower t_c woud also apply. One of the proposed attacks below runs in time $\tilde{O}(2^{\kappa/2})$ under reasonable conjectures, but it appears hard to improve upon this approach.

The attack template. Fix a prime $2^\kappa < p < 2^{\kappa+1}$. The basic version of the attack queries a single vector $\alpha = (0, 1, 0, \ldots, 0)$ to $\mathcal{O}$, which is then associated with a single uniform challenge $c \in \mathbb{Z}_p$. We let $\vec{w} = c \cdot \vec{v}_0 + \vec{v}_1 \in \mathbb{Z}_p^{t+1}$, and we aim to find a set $\mathrm{CS} \subseteq [2..n]$ of size $|\mathrm{CS}| = t_c$ such that $\vec{w} \in \mathsf{Span}(\{\vec{v}_i\}_{i \in \mathrm{CS}})$.

We can now proceed as follows to find such a set CS. First off, we notice that $\{\vec{v}_i\}_{i \in \mathrm{CS} \cup \{0\}}$ is a basis of $\mathbb{Z}_p^{t+1}$ for any CS with $|\mathrm{CS}| = t_c$, and thus we can find coefficients $(\alpha_i^{\mathrm{CS}})_{i \in \mathrm{CS} \cup \{0\}}$ such that

$$\vec{v}_1 = \sum_{i \in \mathrm{CS} \cup \{0\}} \alpha_i^{\mathrm{CS}} \vec{v}_i \ .$$

We note that the adversary wins if it can find CS such that $\alpha_0^{\mathrm{CS}} = -c = p - c$. To this end, the following claim gives an explicit formula for α_0^{CS}.

Claim. $\alpha_0^{\mathrm{CS}} = \prod_{j \in \mathrm{CS}} (1 - j^{-1})$

Proof. Let $2 \leq j_1 < j_2 < \cdots < j_{t_c} \leq n$ be the elements of CS. Then, we want to solve the system of equations

$$(\alpha_0^{\mathrm{CS}}, \alpha_{j_1}^{\mathrm{CS}}, \ldots, \alpha_{j_{t_c}}^{\mathrm{CS}}) \underbrace{\begin{bmatrix} \vec{v}_0 \\ \vec{v}_{j_1} \\ \vdots \\ \vec{v}_{j_{t_c}} \end{bmatrix}}_{=: \vec{V}_{\mathrm{CS}}} = \vec{v}_1 = (1, 1, \ldots, 1) \ .$$

Now, let $\vec{a} \in \mathbb{Z}_p^{t+1}$ be a (row) vector such that its transpose $\vec{a}^\top$ is the first column of the inverse $\vec{V}_{\mathrm{CS}}^{-1}$ of $\vec{V}_{\mathrm{CS}}$. It can be thought of as defining the coefficients of a polynomial $a(X)$ of degree t such that $a(0) = \vec{v}_0 \vec{a}^\top = 1$, and $a(j) = \vec{v}_j \vec{a}^\top = 0$ for all $j \in \mathrm{CS}$. By Lagrange interpolation, this polynomial is uniquely defined as

$$a(X) = 1 \cdot \prod_{j \in S} \frac{X - j}{0 - j} = \prod_{j \in S} (1 - j^{-1} X) \ .$$

The claim follows by observing that $\alpha_0^{\mathrm{CS}} = \vec{v}_1 \vec{a}^\top = a(1)$. □

Attacks. We only give some high-level ideas about attack strategies that follow this template.

A first attack, given a random $-c$, attempts to find a subset $\mathrm{CS} \subseteq [2..n]$ with size $|\mathrm{CS}| = t$ such that $\alpha_0^{\mathrm{CS}} = \prod_{j \in \mathrm{CS}} (1 - j^{-1}) = -c$. This can be done by using dynamic programming in time $\tilde{\Theta}(n \cdot t \cdot p)$, nothing that in general $n \cdot t$ is much smaller than p.

It is not clear how to prove a formal lower bound on the success probability of this attack. However, under the assumption that $\binom{n-1}{t} > p$, it is plausible to conjecture that the set of potential values α_0^{CS} covers a constant fraction of $\mathbb{Z}_p$, and thus the success probability is constant.

A better attack strategy exploits multiple targets. In particular, assume the adversary queries $\alpha = (0, 1, 0, \ldots, 0)$ a total of $q = \sqrt{p}$ times to $\mathcal{O}$ (recall that repeat queries *are* allowed); each query is then associated with a uniform challenge $c_i \in \mathbb{Z}_p$ defining a vector $\vec{w}_i = c_i \cdot \vec{v}_0 + \vec{v}_1 \in \mathbb{Z}_p^{t+1}$. Our goal now is to find CS such that $\alpha_0^{\mathrm{CS}} = \prod_{j \in \mathrm{CS}}(1 - j^{-1}) \in \{c_1, \ldots, c_q\}$.

We can then use the following simple strategy of picking random sets $\mathrm{CS}_1, \mathrm{CS}_2, \ldots, \mathrm{CS}_q$, and if $\alpha_0^{\mathrm{CS}_j} \in \{c_1, \ldots, c_q\}$ for some $j \in [1..q]$, we have succeeeded. This strategy can be implemented in time $\tilde{O}(\sqrt{p})$, where we are hiding small polylogarithmic terms in p. We note that the probability that each CS_i satisfies $\alpha_0^{\mathrm{CS}_j} \in \{c_1, \ldots, c_q\}$ is approximately q/p, and the expected number of j's for which this is true is $q^2/p = 1$.

Of course, to formally validate this attack, we would have to prove that the $\alpha_0^{\mathrm{CS}_j}$'s are sufficiently independent, but we take this simple attack as strong validation that an attack taking time $2^{\kappa/2}$ exists.

Acknowledgments

Research of the last two authors was partially supported by NSF grants CNS-2026774, CNS-2154174, and CNS-2426905, a gift from Microsoft, and a Stellar Development Foundation Academic Research Award.

References

1. Abe, M., Fehr, S.: Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In: Franklin, M.K. (ed.) Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings. LNCS, vol. 3152, pp. 317–334. Springer (2004). https://doi.org/10.1007/978-3-540-28628-8_20, https://doi.org/10.1007/978-3-540-28628-8_20
2. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006. LNCS, vol. 4004, pp. 593–611. Springer (2006)
3. Bacho, R., Das, S., Loss, J., Ren, L.: Glacius: Threshold Schnorr signatures from DDH with full adaptive security (2024), available at https://eprint.iacr.org/2024/1628
4. Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. IACR Cryptol. ePrint Arch. p. 534 (2022), https://eprint.iacr.org/2022/534
5. Bacho, R., Loss, J., Stern, G., Wagner, B.: HARTS: High-threshold, adaptively secure, and robust threshold Schnorr signatures (2024), available at https://eprint.iacr.org/2024/280

6. Bacho, R., Loss, J., Tessaro, S., Wagner, B., Zhu, C.: Twinkle: Threshold signatures from DDH with full adaptive security. IACR Cryptol. ePrint Arch. p. 1482 (2023), https://eprint.iacr.org/2023/1482
7. Battagliola, M., Borin, G., Crescenzo, G.D., Meneghetti, A., Persichetti, E.: Enhancing threshold group action signature schemes: Adaptive security and scalability improvements. In: Niederhagen, R., Saarinen, M.O. (eds.) Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Taipei, Taiwan, April 8-10, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15577, pp. 129–161. Springer (2025). https://doi.org/10.1007/978-3-031-86599-2_5, https://doi.org/10.1007/978-3-031-86599-2_5
8. Bauer, B., Fuchsbauer, G., Plouviez, A.: The one-more discrete logarithm assumption in the generic group model. In: Tibouchi, M., Wang, H. (eds.) Asiacrypt 2021. LNCS, vol. 13093, pp. 587–617. Springer (2021)
9. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Crypto 2022. LNCS (2022)
10. Bellare, M., Palacio, A.: GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) Crypto 2002. LNCS, vol. 2442, pp. 162–177. Springer (2002)
11. Bellare, M., Shoup, S.: Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 201–216. Springer (2007)
12. Bellare, M., Tessaro, S., Zhu, C.: Stronger security for non-interactive threshold signatures (2022), available at https://eprint.iacr.org/2022/833
13. Brandão, L., Davidson, M.: Notes on threshold EdDSA/Schnorr signatures. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8214B.ipd.pdf (2022)
14. Brandão, L., Peralta, R.: NIST first call for multi-party threshold schemes. https://nvlpubs.nist.gov/nistpubs/ir/2025/NIST.IR.8214C.2pd.pdf (2025)
15. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1769–1787. ACM (2020). https://doi.org/10.1145/3372297.3423367, https://doi.org/10.1145/3372297.3423367
16. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. Theor. Comput. Sci. **939**, 78–104 (2023). https://doi.org/10.1016/J.TCS.2022.10.016, https://doi.org/10.1016/j.tcs.2022.10.016
17. Chu, H., Gerhart, P., Ruffing, T., Schröder, D.: Practical Schnorr threshold signatures without the algebraic group model. In: Handschuh, H., Lysyanskaya, A. (eds.) Crypto 2023. LNCS, vol. 14081, pp. 743–773. Springer (2023)
18. Connolly, D., Gouvea, C.: Speeding up FROST with multi-scalar multiplication (2023), available at https://zfnd.org/speeding-up-frost-with-multi-scalar-multiplication
19. Connolly, D., Komlo, C., Goldberg, I., Wood, C.: Two-round threshold Schnorr signatures with FROST (2022), available at https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost
20. Crites, E., Komlo, C., Maller, M.: On the adaptive security of key-unique threshold signatures. Cryptology ePrint Archive, Paper 2025/943 (2025), https://eprint.iacr.org/2025/943

21. Crites, E., Stewart, A.: A plausible attack on the adaptive security of threshold Schnorr signatures, *To appear in Crypto 2025*
22. Crites, E.C., Komlo, C., Maller, M.: How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures (2021), available at https://eprint.iacr.org/2021/1375
23. Crites, E.C., Komlo, C., Maller, M.: Fully adaptive Schnorr threshold signatures. In: Handschuh, H., Lysyanskaya, A. (eds.) Crypto 2023. LNCS, vol. 14081, pp. 678–709. Springer (2023), https://doi.org/10.1007/978-3-031-38557-5_22
24. Das, S., Ren, L.: Adaptively secure BLS threshold signatures from DDH and co-cdh. IACR Cryptol. ePrint Arch. p. 1553 (2023), https://eprint.iacr.org/2023/1553
25. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005, Santa Barbara, California, USA, August 14-18, 2005. LNCS, vol. 3621, pp. 152–168. Springer (2005)
26. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Crypto 2018. LNCS, vol. 10992, pp. 33–62. Springer (2018)
27. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) Eurocrypt 2020. LNCS, vol. 12106, pp. 63–95. Springer (2020)
28. Gouvea, C., Komlo, C., Connolly, D.: FROST for spend authorization multisignatures (2022), available at https://zips.z.cash/zip-0312
29. Katsumata, S., Reichle, M., Takemure, K.: Adaptively secure 5-round threshold signatures from MLWE/MSIS and DL with rewinding. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology—Crypto 2024, Part VII. LNCS, vol. 14926, pp. 459–491. Springer (2024), https://doi.org/10.1007/978-3-031-68394-7_15
30. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 34–65. Springer (2020)
31. Komlo, C., Goldberg, I.: Arctic: Lightweight and stateless threshold Schnorr signatures. Cryptology ePrint Archive, Paper 2024/466 (2024), https://eprint.iacr.org/2024/466
32. Koziel, B., Gordon, S.D., Gentry, C.: Fast two-party threshold ECDSA with proactive security. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 1567–1580. ACM (2024). https://doi.org/10.1145/3658644.3670387, https://doi.org/10.1145/3658644.3670387
33. Lindell, Y.: Simple three-round multiparty Schnorr signing with full simulatability. IACR Communications in Cryptology **1**(1) (2024)
34. Makriyannis, N.: On the classic protocol for MPC Schnorr signatures. Cryptology ePrint Archive (2022), available at https://eprint.iacr.org/2022/1332
35. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: Simple two-round Schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) Crypto 2021. LNCS, vol. 12825, pp. 189–221. Springer (2021)
36. Nicolosi, A., Krohn, M.N., Dodis, Y., Mazières, D.: Proactive two-party signatures for user authentication. In: NDSS 2003. The Internet Society (2003), https://www.ndss-symposium.org/ndss2003/proactive-two-party-signatures-user-authentication/
37. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000)

38. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: ROAST: Robust asynchronous schnorr threshold signatures. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) CCS 2022. pp. 2551–2564. ACM (2022)
39. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
40. Shamir, A.: How to share a secret. Comm. ACM **22**(11), 612–613 (1979)
41. Tessaro, S., Zhu, C.: Threshold and multi-signature schemes from linear hash functions. In: Hazay, C., Stam, M. (eds.) Eurocrypt 2023. LNCS, vol. 14008, pp. 628–658. Springer (2023), https://doi.org/10.1007/978-3-031-30589-4_22

A Proof of Adaptive Security for $t/2$ Corruptions

We now prove Theorem 1. Our reduction makes use of rewinding. We give the general forking algorithm by Bellare and Neven [**?**] in Figure 6, and general forking lemma next.

Lemma 3 (General Forking Lemma [?]). *Let $q \in \mathbb{N}$ and H be a finite, non-empty set. Let IG be a randomized algorithm that generates an input X. Let $\mathcal{C}$ be a randomized algorithm that takes as input X and $h_1, \ldots, h_q \in H$ and random coins ρ and outputs an index $I \in [0..q]$ and side output out. Let $\mathsf{accept}(\mathcal{C})$ be the probability that $\mathcal{C}$ outputs an index $I \neq 0$ in the following experiment:*

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

Let the forking algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$ associated to $\mathcal{C}$ be the randomized algorithm that takes as input X and proceeds as in Figure 6. Let

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

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

$$\mathsf{accept}(\mathsf{Fork}^{\mathcal{C}}) \geq \mathsf{accept}(\mathcal{C}) \cdot \left(\frac{\mathsf{accept}(\mathcal{C})}{q} - \frac{1}{|H|} \right). \tag{20}$$

or, alternatively,

$$\mathsf{accept}(\mathcal{C}) \leq \frac{q}{|H|} + \sqrt{q \cdot \mathsf{accept}(\mathsf{Fork}^{\mathcal{C}})} \tag{21}$$

We now prove Theorem 1.

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the adaptive unforgeability of FROST (Fig. 3) that makes up to q_h queries to $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$, and q_s queries to $\mathcal{O}^{\mathsf{Sign}_1}$. $\mathcal{A}$ is allowed to make $t/2$ adaptive corruptions.

We assume, without loss of generality, that $\mathcal{A}$ makes a query to $\mathsf{H}_{\mathsf{sig}}$ on its forgery $(\mathsf{PK}, m^*, \tilde{R}^*)$ and that it always queries $\mathsf{H}_{\mathsf{non}}(\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (resp. $\mathsf{H}_{\mathsf{non}}(k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ for $\mathsf{FROST1}$) before querying $\mathcal{O}^{\mathsf{Sign}_1}(k, \mathcal{S}, m, \mathcal{PM}_1)$ for

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

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

Fig. 6. The forking algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$ associated to an algorithm $\mathcal{C}$ and input X.

some $k \in \mathcal{S}$. This adds $q_s + 1$ additional random-oracle queries. Then, let $q = q_h + q_s + 1$.

We define an algorithm $\mathcal{C}$ that simulates the adaptive unforgeability game. $\mathcal{C}$ is given an oracle $\mathcal{O}^{\mathsf{dl}}$, which it can query on a group element, and receive its discrete logarithm. $\mathcal{C}$ can query this oracle up to $2q_s + t$ times.

Setup. $\mathcal{C}$ accepts as input an instance X, which is a tuple consisting of the group description $(\mathbb{G}, p, g)$ and a set of $2q_s+t+1$ AOMDL challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t, X_1, X'_1, \ldots, X_{q_s}, X'_{q_s})$. In addition, $\mathcal{C}$ accepts as input a set of $q = q_h + q_s + 1$ values $\{h_1, \ldots, h_q\}$, as in Figure 6, which it will use to program its random-oracle responses.

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

Otherwise, $\mathcal{C}$ initializes the following values:

1. tables $\mathsf{Q}_{\mathsf{non}} := \emptyset$, $\mathsf{Q}_{\mathsf{sig}} := \emptyset$, to store random-oracle queries to $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}}$ and corresponding simulated responses;
2. table $\mathsf{Q}_m := \emptyset$, to store the set of messages queried by $\mathcal{A}$ to $\mathcal{O}^{\mathsf{Sign}_2}$;
3. table $\mathsf{Q}_{\mathsf{st}} := \emptyset$ of participants' states for all signing sessions initiated by $\mathcal{A}$;
4. table $\mathsf{Q}_{\mathsf{used}} := \emptyset$ to track information associated with each AOMDL pair (X_i, X'_i) used in an $\mathcal{O}^{\mathsf{Sign}_2}$ query;
5. counter $\mathsf{ctr} := 0$, to iterate through pairs in the AOMDL challenges;
6. counter $\mathsf{ctr}_{\mathsf{RO}} := 0$, to iterate through random-oracle queries;
7. counter $\mathsf{ctr}_k := 0$, to count the number of first-round signing queries for each party k.

$\mathcal{C}$ then picks random coins ρ and runs $\mathcal{A}(\mathsf{par}; \rho)$ once on the public parameters $\mathsf{par} = (\mathbb{G}, p, g)$ and randomness ρ.

Simulating KeyGen. To simulate key generation where $\mathcal{C}$ is the trusted dealer, $\mathcal{C}$ uses the AOMDL challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t)$ to define a polynomial $f(Z) =$

$x_0 + \hat{x}_1 Z + \cdots + \hat{x}_t Z^t$ "in the exponent" (see Section 2) by deriving each party's public key as:

$$\mathsf{PK}_i = X_0 \hat{X}_1^i \cdots \hat{X}_t^{i^t} \tag{22}$$

for all $i \in [n]$, where implicitly $\mathsf{sk}_i = f(i)$. The threshold public key is $\mathsf{PK} := X_0$ with corresponding secret key $\mathsf{sk} = x_0 = f(0)$. $\mathcal{C}$ then runs $\mathcal{A}((\mathbb{G}, g, p), \mathsf{PK}, \{\mathsf{PK}_i\}_{i\in[n]})$ with access to the signing and random oracles simulated as follows.

Simulating Random Oracles. When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ or $\mathsf{H}_{\mathsf{sig}}$, $\mathcal{C}$ simulates the response by lazy sampling, as follows.

$\underline{\mathsf{H}_{\mathsf{non}}:}$ When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on T, for FROST1, $\mathcal{C}$ parses $T = (k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$, and for FROST2 and FROST3, $\mathcal{C}$ parses $T = (\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$. (Here w.l.o.g. we assume T can be parsed correctly and it is for PK, since otherwise the query on a malformed T would not help the adversary to win the game.) Then, $\mathcal{C}$ checks if $\mathsf{Q}_{\mathsf{non}}[T] = \bot$. If not, $\mathcal{C}$ returns $\mathsf{Q}_{\mathsf{non}}[T]$. Otherwise, $\mathcal{C}$ increments $\mathsf{ctr}_{\mathsf{RO}} := \mathsf{ctr}_{\mathsf{RO}} + 1$, and, for FROST1, $\mathcal{C}$ does the following:

1. For all $i \in \mathcal{S}$, $\mathcal{C}$ samples $a_i := h_{(n+1)\mathsf{ctr}_{\mathsf{RO}}-i}$ and stores $\mathsf{Q}_{\mathsf{non}}[(\mathsf{PK}, \mathcal{S}, m, \{(\ell, R_\ell, S_\ell)\}_{\ell\in\mathcal{S}}, i)] = a_i$.
2. Then, $\mathcal{C}$ derives the aggregate $\tilde{R}$ from T (i.e., $\tilde{R} := \prod_{i\in\mathcal{S}} R_i S_i^{a_i}$) and queries $\mathsf{H}_{\mathsf{sig}}$ on $(\mathsf{PK}, m, \tilde{R})$.
3. Finally, $\mathcal{C}$ returns a_k.

For FROST2 and FROST3, $\mathcal{C}$ does the following:

1. $\mathcal{C}$ sets $a := h_{2\mathsf{ctr}_{\mathsf{RO}}-1}$ and stores $\mathsf{Q}_{\mathsf{non}}[T] = a$.
2. $\mathcal{C}$ derives the aggregate $\tilde{R}$ from T. (For FROST2, it means that $\mathcal{C}$ parses $\mathcal{PM}_1 = \{(i, R_i, S_i)\}_{i\in\mathcal{S}}$ and computes $\tilde{R} := \prod_{\ell\in\mathcal{S}} R_\ell S_\ell^a$. For FROST3, it means that $\mathcal{C}$ parses $\mathcal{PM}_1 = (\hat{R}, \hat{S})$ and computes $\tilde{R} := \hat{R}\hat{S}^a$.)
3. If $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, \tilde{R})] = \bot$, $\mathcal{C}$ sets $c := h_{2\mathsf{ctr}_{\mathsf{RO}}}$ and stores $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, \tilde{R})] = c$.
4. Finally, $\mathcal{C}$ returns a.

$\underline{\mathsf{H}_{\mathsf{sig}}:}$ When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (PK, m, R), $\mathcal{C}$ checks if $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, R)] = \bot$. (Similar to $\mathsf{H}_{\mathsf{non}}$, here we only consider queries for PK, since otherwise the query would not help the adversary to win the game.) If not, $\mathcal{C}$ samples $c \leftarrow\!\!\$\ \mathbb{Z}_p$, stores $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, R)] = c$, and returns c. Otherwise, $\mathcal{C}$ increments $\mathsf{ctr}_{\mathsf{RO}} := \mathsf{ctr}_{\mathsf{RO}} + 1$, sets $c := h_{2\mathsf{ctr}_{\mathsf{RO}}}$, (resp. $h_{(n+1)\mathsf{ctr}_{\mathsf{RO}}}$ for FROST1) stores $\mathsf{Q}_{\mathsf{sig}}[(\mathsf{PK}, m, R)] = c$, and returns c.

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

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

1. Sets $\mathsf{eid} := \mathsf{ctr}_k$ and increments $\mathsf{ctr}_k := \mathsf{ctr}_k + 1$.

2. Sets $(R_k, S_k) := (X_{\mathsf{ctr}}, X'_{\mathsf{ctr}})$ and increments $\mathsf{ctr} := \mathsf{ctr} + 1$. $\mathcal{C}$ checks if there exists $T = (\mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ (resp. $T = (k, \mathsf{PK}, \mathcal{S}, m, \mathcal{PM}_1)$ for FROST1) in $\mathsf{Q}_{\mathsf{non}}[T]$ such that $(X_{\mathsf{ctr}}, X'_{\mathsf{ctr}}) = (R_k, S_k)$. If so, $\mathcal{C}$ aborts. We refer to this bad event as NonceCollision.
3. Sets $\mathsf{st}_{k,\mathsf{eid},1} := (R_k, \bot, S_k, \bot, \bot)$. # L-Note: i.e., r_k, s_k, sk_k are not yet defined.
4. Updates $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1] := \mathsf{st}_{k,\mathsf{eid},1}$.
5. Finally, $\mathcal{C}$ returns (R_k, S_k) to $\mathcal{A}$.

Round 2 ($\mathcal{O}^{\mathsf{Sign}_2}(k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$): In the second round of signing, each party in the signing set $\mathcal{S}$ takes as input the first-round nonce commitments $\mathcal{PM}_1$ and outputs is partial signature share z_i.

When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}_2}$ on $(k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$ for $k \in \mathsf{hon}$, $\mathcal{C}$ first performs the checks in $\mathcal{O}^{\mathsf{Sign}_2}$ (Fig. 3) and Sign_2 (Fig. 4). If they pass, then, in particular $\mathsf{st}_{k,\mathsf{eid},1} = (R_k, \bot, S_k, \bot, \bot)$. # L-Note: i.e., again, r_k, s_k, sk_k are not yet defined.
$\mathcal{C}$ then does the following:

1. $\mathcal{C}$ updates $\mathsf{Q}_m := \mathsf{Q}_m \cup \{m\}$.
2. $\mathcal{C}$ follows the Sign_2 algorithm (Fig. 4) to obtain a and c. (For FROST1, a here refers to a_k.) Note that $\mathcal{C}$ can internally query $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$ as needed.
3. $\mathcal{C}$ then sets $Z_k := R_k S_k^a \cdot {\mathsf{PK}_k}^{c\lambda_k}$, i.e., $Z_k = X_{i'}(X'_{i'})^a X_0^{c\lambda_k} \hat{X}_1^{c\lambda_k k} \cdots \hat{X}_t^{c\lambda_k k^t}$ for some $i' \in [q_s]$. $\mathcal{C}$ queries $\mathcal{O}^{\mathsf{dl}}$ on Z_k with representation $\alpha = 0$ and (as in Fig. 1):

$$(\beta_i)_{i=1}^{2q_s+t+1} = (c\lambda_k, c\lambda_k k, \ldots, c\lambda_k k^t, 0, \ldots, 0, 1, a, 0, \ldots, 0),$$

   where 1 is in position $t + 2i'$, and receives z_k.
4. $\mathcal{C}$ sets $\mathsf{st}_{k,\mathsf{eid},2} := z_k$.
5. $\mathcal{C}$ adds $((R_k, S_k), (i', z_k, \eta, a, c\lambda_k), \mathsf{query})$ to $\mathsf{Q}_{\mathsf{used}}$, where $\mathsf{query} = (k, \mathsf{eid}, \mathcal{S}, m, \mathcal{PM}_1)$ and η is the index such that $a = h_{2\eta-1}$.
6. Finally, $\mathcal{C}$ returns z_k to $\mathcal{A}$.

Simulating Corruption Queries $\mathcal{O}^{\mathsf{Corrupt}}(k)$: $\mathcal{A}$ may at any time corrupt an honest party k by querying $\mathcal{O}^{\mathsf{Corrupt}}(k)$. Upon receiving a corruption query, $\mathcal{C}$ first checks that $k \in \mathsf{hon}$ and that $|\mathsf{cor}| < t$, returning $\bot$ if not. Otherwise, $\mathcal{C}$ sets $\mathsf{cor} := \mathsf{cor} \cup \{k\}$ and $\mathsf{hon} := \mathsf{hon} \setminus \{k\}$.

$\mathcal{C}$ then queries $\mathcal{O}^{\mathsf{dl}}$ on PK_k with representation $\alpha = 0$ and (recalling that $\mathsf{PK}_k = X_0 \hat{X}_1^k \cdots \hat{X}_t^{k^t}$ (Eq. (22)) :

$$(\beta_i)_{i=0}^{2q_s+t} = (1, k, \ldots, k^t, 0, \ldots, 0)$$

and receives sk_k. $\mathcal{C}$ then does the following:

1. If no entry $\mathsf{Q}_{\mathsf{st}}[k, \mathsf{eid}, 1]$ exists for any eid that has occurred, then $\mathcal{C}$ returns sk_k to $\mathcal{A}$. # L-Note: This corresponds to the case where $\mathcal{A}$ has not queried party k to $\mathcal{O}^{\mathsf{Sign}_1}$ in any signing session.
2. Otherwise, $\mathcal{C}$ does the following.

(a) For each entry $\mathsf{Q_{st}}[k, \mathsf{eid}, 1] \neq \bot$ where $\mathsf{Q_{st}}[k, \mathsf{eid}, 2] = \bot$, # L-Note: This corresponds to Round 1 queries that never made it to Round 2. $\mathcal{C}$ queries $\mathcal{O}^{\mathsf{dl}}$ on R_k, which is equal to $X_{i'}$ for some $i' \in [q_s]$, with representation $\alpha = 0$ and $(\beta_i)_{i=0}^{2q_s+t} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + 2i'$, and receives r_k. $\mathcal{C}$ similarly queries $\mathcal{O}^{\mathsf{dl}}$ on S_k (with representation $\alpha = 0$ and $(\beta_i)_{i=0}^{2q_s+t} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + 2i' + 1$), and receives s_k. $\mathcal{C}$ then sets $\mathsf{Q_{st}}[k, \mathsf{eid}, 1] = (R_k, r_k, S_k, s_k, \mathsf{sk}_k)$.

(b) For each entry where $\mathsf{Q_{st}}[k, \mathsf{eid}, 2] \neq \bot$, # L-Note: This corresponds to Round 1 queries that made it to Round 2. $\mathcal{C}$ queries $\mathcal{O}^{\mathsf{dl}}$ on S_k, which is equal to $X'_{i'+1}$ for some $i' \in [q_s]$, with representation $\alpha = 0$ and $(\beta_i)_{i=0}^{2q_s+t} = (0, \ldots, 0, 1, 0, \ldots, 0)$, where 1 is in position $t + i' + 2$, and receives s_k. $\mathcal{C}$ already made an $\mathcal{O}^{\mathsf{dl}}$ query to obtain z_k, so it can compute $r_k = z_k - s_k a - c\lambda_k \mathsf{sk}_k$. (For FROST1, a here refers to a_k.) $\mathcal{C}$ then sets $\mathsf{Q_{st}}[k, \mathsf{eid}, 1] = (R_k, r_k, S_k, s_k, \mathsf{sk}_k)$.

Finally, $\mathcal{C}$ returns sk_k and all state $\mathsf{Q_{st}}[k, \cdot, \cdot]$ for party k to $\mathcal{A}$.

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

Otherwise, $\mathcal{C}$ looks up the index I for $(\mathsf{PK}, m^*, R^*, c^*) \in \mathsf{Q_{sig}}$ corresponding to $\mathcal{A}$'s forgery. It sets $\mathsf{out} := (\sigma^*, \mathsf{aux}^*)$, where aux^* are all variables received or generated by $\mathcal{C}$, and outputs (I, out).

$\mathcal{C}$'s simulation is perfect when the values in the instance X are sampled uniformly at random. $\mathcal{C}$ aborts with negligible probability in the following cases:

1. ROCollision: $\mathcal{C}$ aborts due to $\mathsf{ROCollision}$ with probability less than $\frac{q^2}{p}$.
2. NonceCollision: $\mathcal{C}$ aborts due to $\mathsf{NonceCollision}$ with probability less than $\frac{q^2}{2p}$.

Reduction $\mathcal{B}$ Against AOMDL. Finally, we construct our PPT reduction $\mathcal{B}$ against the AOMDL assumption (Fig. 1). Specifically, we show how $\mathcal{B}$ uses $\mathsf{Fork}^{\mathcal{C}}$ as a subroutine such that:

$$\mathsf{Adv}_{\mathcal{B}}^{t+1\text{-}\mathsf{aomdl}}(\kappa) \geq \mathsf{accept}(\mathsf{Fork}^{\mathcal{C}}(X)) - 2\Pr[\mathsf{ROCollision}] - 2\Pr[\mathsf{NonceCollision}] \tag{23}$$

where X is the instance given by the AOMDL challenger.

Initialization. $\mathcal{B}$ is initialized by the AOMDL challenger with input the group description $(\mathbb{G}, p, g)$ and $2q_s + t + 1$ AOMDL challenges $(X_0, \hat{X}_1, \ldots, \hat{X}_t, X_1, X'_1, \ldots, X_{q_s}, X'_{q_s})$. $\mathcal{B}$ has access to a discrete-logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to $2q_s + t$ times. $\mathcal{B}$ aims to output $(x_0, \hat{x}_1, \ldots, \hat{x}_t, x_1, x'_1, \ldots, x_{q_s} x'_{q_s})$ such that $X_i = g^{x_i}$ for all i, with only $2q_s + t$ queries to its DL oracle.

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

Execution. $\mathcal{B}$ then runs the general forking algorithm $\mathsf{Fork}^{\mathcal{C}}(X)$ as described in Figure 6, on the simulator algorithm $\mathcal{C}$ and its instance X.

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

With non-negligible probability lower-bounded by Equation 20, $\mathsf{Fork}^{\mathcal{C}}(X)$ will output the accepting answer $(I, \mathsf{out}, \mathsf{out}')$, such that:

$$h_I = c^*,\ h'_I = c^{**},$$
$$(\sigma^*, \mathsf{aux}^*) := \mathsf{out},\ (\sigma^{**}, \mathsf{aux}^{**}) := \mathsf{out}'$$
$$(R^*, z^*) := \sigma^*,\ (R^*, z^{**}) := \sigma^{**}.$$

Extracting the Discrete Logarithm of X_0. We show that $\mathcal{B}$ can extract the discrete logarithm of X_0 from $\mathcal{A}$'s two valid forgeries. We assume without loss of generality that $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (PK, R^*, m^*) on its forgery in execution.

With overwhelming probability, $c^* \neq c^{**}$, and $\mathcal{B}$ can solve for $x_0 = \frac{z^* - z^{**}}{c^* - c^{**}}$. If $\mathcal{B}$ extracts x_0, then we use this to extract a full AOMDL solution as follows.

Extracting an AOMDL Solution. $\mathcal{B}$ must now extract the remaining $(\hat{x}_1, \ldots, \hat{x}_t, x_1, x'_1, \ldots, x_{q_s}, x'_{q_s})$. Assume without loss of generality that $\mathcal{A}$ makes $t/2$ corruptions in each iteration. Recall that $\mathsf{PK}_i = X_0 \hat{X}_1^i \cdots \hat{X}_t^{i^t}$. Since t $\mathcal{O}^{\mathsf{dl}}$ queries have been made on PK_i to obtain sk_i, together with x_0, this gives $t+1$ points on the polynomial $f(Z) = x_0 + \hat{x}_1 Z + \cdots + \hat{x}_t Z^t$ defining key generation. Thus, $\mathcal{B}$ can solve for $\hat{x}_1, \ldots, \hat{x}_t$.

Now, for $x_1, x'_1, \ldots, x_{q_s}, x'_{q_s}$ corresponding to each X_i, the method for extracting x_i will be one of the following cases, depending on how $\mathcal{A}$ queried $\mathcal{O}^{\mathsf{Sign}_1}$ and $\mathcal{O}^{\mathsf{Sign}_2}$ over the two iterations.

Case A. (X_i, X'_i) belongs to an honest party at the end of the two iterations.

Case 1 ((X_i, X'_i) has not appeared in an $\mathcal{O}^{\mathsf{Sign}_2}$ query over the two iterations). In this case, X_i and X'_i have not yet been queried by $\mathcal{B}$. Thus, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ directly to obtain x_i and x'_i. Two queries are made for each i in this case.

Case 2 ((X_i, X'_i) has appeared in an $\mathcal{O}^{\mathsf{Sign}_2}$ query in a single iteration). A single query has been made by $\mathcal{B}$ to $\mathcal{O}^{\mathsf{dl}}$ containing (X_i, X'_i). If it occurred in the first iteration, then $((R_{k_j}, S_{k_j}), (i, z_{k_j}, \eta_j, a_j, c_j \lambda_{k_j}), \mathsf{query}_j) \in \bar{\mathsf{Q}}_{\mathsf{used}}$, where $\bar{\mathsf{Q}}_{\mathsf{used}}$ denotes $\mathsf{Q}_{\mathsf{used}}$ for the first iteration. (For FROST1, a_j here refers to $a_{k,j}$.) is such that $z_{k_j} = r_{k_j} + a_j s_{k_j} + c_j \lambda_{k_j} \mathsf{sk}_{k_j}$. (Note that $\mathcal{B}$ can compute $f(k_j) = \mathsf{sk}_{k_j}$.) To obtain a second value, $\mathcal{B}$ queries X'_i to $\mathcal{O}^{\mathsf{dl}}$ and thus learns x'_i. Then $\mathcal{B}$ sets $x_i = z_{k_j} - a_j x'_i - c_j \lambda_{k_j} \mathsf{sk}_{k_j}$. The case where the query occurred in the second iteration is similar. In total, two queries are made for each i in this case.

Now consider when (X_i, X'_i) appears in an $\mathcal{O}^{\mathsf{Sign}_2}$ query in both iterations. Let query_j be that query from the first iteration and $\mathsf{query}_{j'}$ from the second. Chenzhi: If the above does not use afterward, I suggest to just remove it. It would also make it easier to include FROST1 and FROST3. such that $(X_i, X'_i) =$

$(R_{k_j}, S_{k_j}) = (R_{k_{j'}}, S_{k_{j'}})$. Let

$$((R_{k_j}, S_{k_j}), (i, z_{k_j}, \eta_j, a_j, c_j\lambda_{k_j}), \mathsf{query}_j) \in \bar{\mathsf{Q}}_{\mathsf{used}}$$
$$((R_{k_{j'}}, S_{k_{j'}}), (i, z_{k'_j}, \eta_{j'}, a_{j'}, c_{j'}\lambda_{k_{j'}}), \mathsf{query}_{j'}) \in \mathsf{Q}_{\mathsf{used}}$$

be the associated table entries for query_j and $\mathsf{query}_{j'}$. Recall that I is the index such that $c^* = h_I$.

Case 3 ($\eta_j = \eta_{j'} < I$). In this case, a single query has been made, to obtain z_{k_j} in $((R_{k_j}, S_{k_j}), (i, z_{k_j}, \eta_j, a_j, c_j\lambda_{k_j}), \mathsf{query}_j) \in \bar{\mathsf{Q}}_{\mathsf{used}}$, such that $z_{k_j} = r_{k_j} + a_j s_{k_j} + c_j\lambda_{k_j}\mathsf{sk}_{k_j}$. To obtain a second value, $\mathcal{B}$ queries X'_i to $\mathcal{O}^{\mathsf{dl}}$ and thus learns x'_i. Then $\mathcal{B}$ sets $x_i = z_{k_j} - a_j x'_i - c_j\lambda_{k_j}\mathsf{sk}_{k_j}$. In total, two queries are made for each i in this case.

Case 4 ($\eta_j \neq \eta_{j'}$ or $\eta_j = \eta_{j'} > I$). In this case, $a_j \neq a_{j'}$ and $\mathcal{B}$ has made two queries to obtain z_{k_j} and $z_{k_{j'}}$. $\mathcal{B}$ computes:

$$x'_i = \frac{z_{k'_j} - z_{k_j} + c_j\gamma_{k_j}\mathsf{sk}_{k_j} - c_{j'}\gamma_{k_{j'}}\mathsf{sk}_{k'_j}}{a_{j'} - a_j} \qquad x_i = z_{k_j} - a_j x'_i - c_j\lambda_{k_j}\mathsf{sk}_{k_j} \quad (24)$$

Two queries are made for each i in this case.

Case B. (X_i, X'_i) belongs to a corrupt party at the end of the two iterations.
In this case, $\mathcal{B}$ has already made the queries in each of the cases above.

Thus, $\mathcal{B}$ has extracted x_i for all X_i using exactly $2q_s + t$ queries and returns $(x_0, \hat{x}_1, \ldots, \hat{x}_t, x_1, x'_1, \ldots, x_{q_s}, x'_{q_s})$ and win the $2q_s + t + 1$-aomdl game. Combining Equations 20 and 23 gives Equation 2.

B Proof of Lemma 2

Let $A_2(n, d, w)$ denote the maximum size of a code $\mathcal{C} \subseteq \{0,1\}^n$ where all codewords have weight w, and each pair has distance at least d. Clearly $A_2(n, d, w) = 1$ if $d > 2w$, since two codewords of weight w have distance at most $2w$.

Assume $d \leq 2w$ and pick $\mathcal{C}$ such that $|\mathcal{C}| = A_2(n, d, w)$. For $i \in [n]$, we let $\mathcal{C}_i \in \{0,1\}^{n-1}$ be the code obtained by taking all $\mathbf{c} \in \mathcal{C}$ such that $\mathbf{c}[i] = 1$, and then removing the ith coordinate. Note that the codewords in $\mathcal{C}_i$ all have weight $w - 1$, and the distance remains at least d. Let $M_i = |\mathcal{C}_i|$. Note that

$$\sum_{i=1}^{n} M_i = w \cdot |\mathcal{C}| = w \cdot A_2(n, d, w) \ ,$$

since every codewords in $\mathcal{C}$ is added to exactly w sets $\mathcal{C}_i$. On the other hand, $M_i \leq A_2(n-1, d, w-1)$, and therefore

$$A_2(n, d, w) \leq \frac{n}{w} A_2(n-1, d, w-1) \ .$$

For the specific case where $d = 2e$, and $w \geq e$, we can iterate this argument exactly $w - e + 1$ times to obtain

$$A_2(n, 2e, w) \leq \left(\prod_{i=0}^{w-e} \frac{n-i}{w-i} \right) A_2(n - (w-e) - 1, 2e, e-1) = \left(\prod_{i=0}^{w-e} \frac{n-i}{w-i} \right) ,$$

since $A_2(n - (w-e) - 1, 2e, e-1) = 1$.