HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures

Renas Bacho [1,3] Julian Loss [1] Gilad Stern [2]

Benedikt Wagner [1,3]

February 19, 2024

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{renas.bacho,loss,benedikt.wagner}@cispa.de
[2] Tel Aviv University, Tel Aviv-Yafo, Israel
gilad.stern@mail.huji.ac.il
[3] Saarland University, Saarbrücken, Germany

Abstract

Threshold variants of the Schnorr signature scheme have recently been at the center of attention due to their applications to Bitcoin, Ethereum, and other cryptocurrencies. However, existing constructions for threshold Schnorr signatures among a set of n parties with corruption threshold t_c suffer from at least one of the following drawbacks: (i) security only against static (i.e., non-adaptive) adversaries, (ii) cubic or higher communication cost to generate a single signature, (iii) strong synchrony assumptions on the network, or (iv) $t_c + 1$ are sufficient to generate a signature, i.e., the corruption threshold of the scheme equals its reconstruction threshold. Especially (iv) turns out to be a severe limitation for many asynchronous real-world applications where $t_c < n/3$ is necessary to maintain liveness, but a higher signing threshold of $n - t_c$ is needed. A recent scheme, ROAST, proposed by Ruffing et al. (ACM CCS '22) addresses (iii) and (iv), but still falls short of obtaining subcubic complexity and adaptive security.

In this work, we present HARTS, the *first* threshold Schnorr signature scheme to incorporate all these desiderata. More concretely:

- HARTS is adaptively secure and remains fully secure and operational even under asynchronous network conditions in the presence of up to $t_c < n/3$ malicious parties. This is optimal.
- HARTS outputs a Schnorr signature of size λ with a near-optimal amortized communication cost of $O(\lambda n^2 \log n)$ bits and $O(1)$ rounds per signature.
- HARTS is a *high-threshold* scheme: no fewer than $t_r + 1$ signature shares can be combined to yield a full signature, where $t_r \geq 2n/3 > 2t_c$. This is optimal.

We prove our result in a modular fashion in the algebraic group model. At the core of our construction, we design a new simple, and adaptively secure high-threshold AVSS scheme which may be of independent interest.

Keywords: Threshold Signatures, Schnorr Signatures, Adaptive Security, Robustness, High-Threshold, Asynchronous Network

1 Introduction

A *threshold signature* [Des88, DF90] scheme is a special type of digital signature scheme that allows any set of $t_r + 1$ signers in a system of n parties to jointly generate a compact signature σ on a message m. On the other hand, this should be infeasible for t_r or less signers. Over the last two decades, many threshold versions of the Schnorr signature scheme [Sch91] have been proposed [BP23, CGRS23, CKM23a, GJKR07, KG20, Lin22, RRJ+22, SS01, TZ23, WNR20]. Collectively, these schemes offer a great variety

Table 1: Comparison table of some relevant threshold Schnorr signatures.

Scheme	Network	Robust	Corrupt	Reconst	Adapt	Commun	Rounds
FROST [KG20]	*sync*	✗	$t_c < n$	$t_r = t_c$	✗	$O(\lambda n^2)$	2
Sparkle [CKM23a]	*sync*	✗	$t_c < n$	$t_r = t_c$	✓	$O(\lambda n^2)$	3
GJKR [GJKR07]	*sync*	(✓)$^\diamond$	$t_c < n/2$	$t_r = t_c$	✗	$O(\lambda n^3)$	3 BC
ROAST [RRJ+22]	*async*	✓	$t_c < n^\dagger$	$t_r < n - t_c$	✗	$O(\lambda n^3 + n^4)$	$O(n)$
SPRINT [BHK+23]	*async*	✓	$t_c < n/3$	$t_r = t_c$	✗	$O(\lambda n^2)$	3 BC
GS23 [GS23]	*async*	✓	$t_c < n/3$	$t_r = t_c$	✗	$O(\lambda n)$	$O(1)$
Our work	*async*	✓	$t_c < n/3$	$t_r < n - t_c$	✓	$O(\lambda n^2)$	$O(1)$

Network denotes network model: *sync* for synchronous, *async* for asynchronous. **Robust** denotes robustness. $^\diamond$ The work [GJKR07] assumes a synchronous network and thus is not fully robust in our sense. **Corrupt** denotes corruption threshold. **Reconst** denotes reconstruction threshold. **Adapt** denotes adaptive security. **Commun** denotes amortized communication cost per signature in bits. **Rounds** denotes number of rounds per signature. The works [BHK+23, GJKR07] assume an (atomic) broadcast channel BC that parties can access. † ROAST has a weaker notion of robustness and does not guarantee signature generation for $t_c \geq n/3$.

of trade-offs between efficiency and robustness to adverse signer behaviour and network conditions. A recent work, ROAST [RRJ+22], combines several of these desirable features into a single scheme which supports high reconstruction threshold (see below) and maintains liveness even under full asynchrony. Unfortunately, the scheme is rather inefficient: creating a single signature costs $O(\lambda n^3 + n^4)$ bits and $O(n)$ rounds of communication, where λ denotes the size of a signature.

Our Contribution. In this work, we propose HARTS, a novel threshold Schnorr signature scheme that improves significantly over prior works. Concretely, HARTS has the following properties:

- **Efficiency:** HARTS produces signatures with a (near-optimal) amortized communication cost of $O(\lambda n^2 \log n)$ bits and round complexity $O(1)$.
- **Asynchrony:** HARTS remains fully secure and operational against up to (optimal) $t_c < n/3$ corrupted parties in a *fully asynchronous* network where message delivery between honest parties can take longer than expected [LSP82].
- **Adaptive Security:** HARTS is secure against *adaptive corruptions*.
- **High-threshold**: HARTS is a *high-threshold* signature scheme satisfying the following features: (i) a signing session results in a valid signature even in the presence of up to t_c malicious parties that try to prevent the other parties from generating a signature, (ii) a signature cannot be created given less than $t_r + 1 = n - t_c$ signature shares, where $t_r \geq 2n/3 > 2t_c$. This notion of enhanced security has found many applications and real-world significance in recent years, especially in the context of consensus and blockchain systems [GKKS+22, YMR+19].

We refer to Table 1 for a complete overview and comparison of our scheme's properties with existing schemes from the literature.

A Modular Approach. We build HARTS by following a modular approach, which is summarized in Figure 1 and outlined in more detail below.

- **Threshold Schnorr Signatures from ADKG.** Building upon the technique of Gennaro et al. [GJKR99], we construct a generic high-threshold and robust threshold Schnorr signature scheme. As a building block, we use a (packed) high-threshold asynchronous distributed key generation (ADKG) protocol. We prove unforgeability of this scheme against an adaptive adversary in the algebraic group model (AGM) [FKL18] based on the security of the (packed) ADKG protocol and the one-more discrete logarithm assumption.
- **Packed ADKG from AVSS.** We give a generic construction (cf. Figure 2) for an efficient packed high-threshold ADKG from a high-threshold asynchronous verifiable secret sharing (AVSS) scheme using the technique of superinvertible matrices [HN06]. We prove the adaptive security of this construction by reduction to the security of the AVSS scheme and the underlying consensus primitives.

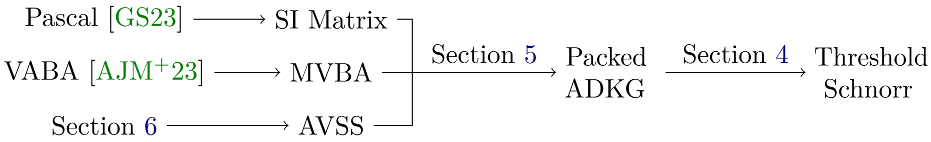

Figure 1: Overview of our framework to construct high-threshold, adaptively secure, and robust threshold Schnorr signatures.

- **New High-Threshold AVSS.** We design a simple high-threshold AVSS scheme and give an adaptive security proof. This gives the first pairing-free, adaptively secure AVSS scheme with quadratic communication cost (cf. Table 2 for a comparison with existing schemes). With our new AVSS scheme and building blocks from the literature, we instantiate our framework, yielding a threshold Schnorr signature scheme with (amortized) communication cost of $O(\lambda n^2 \log n)$ bits per signature.

1.1 Technical Overview

In the following, we provide a technical overview of our work.

Starting Point: Robust Threshold Schnorr Signatures. Our starting point is the construction for robust threshold Schnorr signatures by Gennaro et al. [GJKR99]. First, we recall the (single-party) Schnorr signature scheme. For this, let $\mathbb{G} = \langle g \rangle$ be a group of prime order p with generator g. The secret key is a random element $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$ and the public key is $\mathsf{pk} := g^{\mathsf{sk}}$. To sign a message m, the party samples a random element $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes the signature on m as $\sigma := (R, s)$ where $s = \mathsf{H}(\mathsf{pk}, R, m) \cdot \mathsf{sk} + r \in \mathbb{Z}_p$ and $R = g^r$. Here, $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ is a hash function (modeled as a random oracle). Verification of the signature (R, s) is done by checking $g^s = R \cdot \mathsf{pk}^c$ where $c := \mathsf{H}(\mathsf{pk}, R, m)$. Now let us switch to the multi-party setting in which n parties $P_1, \ldots, P_n$ want to jointly create a signature σ on m. For convenience we assume that parties have already established a (t_r, n)-threshold key setup, e.g., by running a distributed key generation (DKG) protocol. Concretely, this means that each party P_i has a share sk_i of the secret key sk such that any set of $t_r + 1$ shares uniquely determine the secret key and the public key shares $\mathsf{pk}_i := g^{\mathsf{sk}_i}$ are known to all parties. In order to transform the Schnorr signature scheme into a (t_r, n)-threshold signature scheme, Gennaro et al.'s insight was to run a DKG protocol to generate shares r_i of a secret nonce r for parties along with associated public shares $R_i = g^{r_i}$ and $R = g^r$. To sign a message m, each party P_i computes its share of the signature on m as $\sigma_i := (R_i, s_i)$ where $s_i = \mathsf{H}(\mathsf{pk}, R, m) \cdot \mathsf{sk}_i + r_i$. Verification of a signature share σ_i is done with respect to the public key share pk_i and the public nonce share R_i. It can be seen that any $t_r + 1$ valid signature shares recover the full signature $\sigma = (R, s)$. However, a major drawback of this approach is its efficiency: parties need to run a DKG protocol each time they want to sign a new message. Using a state-of-the-art asynchronous DKG protocol in terms of efficiency [AJM+23, DXKKR23], this yields a communication cost of $O(\lambda n^3)$ bits per signature. On the other hand, assuming nonce shares have already been generated, each party can locally compute its signature share and send it to all other parties, which cost only a total of $O(\lambda n^2)$ bits communication.

Regaining Efficiency: Superinvertible Matrices. Introduced by Hirt and Nielsen [HN06] in the context of multi-party computation (MPC) protocols, we use the technique of superinvertible (SI) matrices to construct an (ℓ, t_c, t_r, n)-*packed ADKG protocol* for more efficient multi-nonce generation. Informally, such a protocol executed by n parties has the following property in the presence of up to t_c malicious parties: it outputs ℓ independent keys distributed among the parties such that each of them can be reconstructed independently from the others with reconstruction threshold t_r. Our construction has the following parameters: it generates $\ell = t_c + 1$ keys with (arbitrary) reconstruction threshold $t_r < n - t_c$ in the presence of $t_c < n/3$ malicious parties. Our construction (cf. Figure 2) follows the usual flow of an ADKG protocol with some tweaks in the parameters in order to apply an SI matrix at the end. We briefly elaborate on this. Each party P_i samples a random element $s_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and shares it via an (t_c, t_r, n)-*threshold asynchronous verifiable secret sharing (AVSS) scheme* where s_i lies on some polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t_r. Then, parties agree on a set $I \subset [n]$ of $n - t_c$ dealers whose AVSS sharings completed successfully using a consensus tools. Say $I = \{1, \ldots, n - t_c\}$ so that after this phase, each

Table 2: Comparison table of some relevant AVSS schemes.

Scheme	**Adaptive**	**High-Thresh**	**Pairing-Free**	**No Trust**	**No PKI**	**Commun**
Backes et al. [BDK13]	✗	✗	✗	✗	✓	$O(\lambda n^2)$
hbACSS [YLF+22]	✗	✗	✓	✓	✓	$O(\lambda n^2)$
GoAVSS [SS23]	✗	✗	✓	✓	✓	$O(\lambda n^2)$
Cachin et al. [CKLS02]	✗	(✓)†	✓	✓	✓	$O(\lambda n^3)$
Das et al. [DYX+22]	✗	✓	✓	✓	✗	$O(\lambda n^2)$
HAVEN [AVZ21]	✗	✓	✓	✓	✓	$O(\lambda n^2 \log n)$
Bingo [AJM+23]	✓	✓	✗	✗	✓	$O(\lambda n^2)$
HAVSS (our work)	✓	✓	✓	✓	✓	$O(\lambda n^2 \log n)$

Adaptive denotes adaptive security. **High-Thresh** denotes high reconstruction threshold. † The work [CKLS02] only provides sub-optimal corruption threshold $t_c < n/4$ (in asynchrony). **Pairing-Free** denotes suitability to pairing-free groups. **No Trust** denotes no trusted setup. **No PKI** denotes no public key infrastructure. **Commun** denotes communication cost per sharing in bits.

party P_i has shares $f_1(i), \ldots, f_{n-t_c}(i)$. Instead of just summing up these shares, as is done usually in an ADKG, parties take different linear combinations given by the rows of an $(\ell, n - t_c)$-dimensional SI matrix SI to obtain ℓ new shares $r_1(i), \ldots, r_\ell(i)$. The describing property of an SI matrix now tells us that if at least ℓ input secrets are independent and uniformly random, then the ℓ output secrets are also guaranteed to be independent and uniformly random. Since there are at most t_c malicious parties, we know that $|I| - t_c \geq t_c + 1$ of the dealers specified by the set I are honest. Thus, we can set $\ell := t_c + 1$. Similar constructions were recently introduced in [BHK+23, GS23, Sho23] for the same purpose of efficient multi-nonce generation. These constructions, however, employ low-threshold AVSS schemes with $t_r = t_c < n/3$. To the best of our knowledge, the only existing high-threshold AVSS schemes are [AJM+23, AVZ21, KMS20], each of them with its own limitations. Kokoris-Kogias et al.'s AVSS [KMS20] has cubic communication cost, resulting in prohibitive $\Omega(\lambda n^4)$ communication to share $t_c + 1$ nonces. Alhaddad et al. [AVZ21] provide a generic construction for AVSS with quadratic communication cost, but lacking a proof of adaptive security. Abraham et al.'s AVSS [AJM+23] relies on the KZG polynomial commitment scheme [KZG10] that requires pairings (and trusted setup) which is not suitable for Schnorr signatures.

HAVSS: New AVSS Scheme to the Rescue. We take insights from both protocols, Bingo [AJM+23] and HAVEN [AVZ21], and combine certain aspects to obtain a simple high-threshold AVSS scheme called HAVSS. On a high level, our AVSS scheme works as follows. The designated dealer P_d holds a secret $s \in \mathbb{Z}_p$ as input that it wants to share among all parties. For this, it samples a bivariate polynomial $S \in \mathbb{Z}_p[X, Y]$ of degree t_r in X and t_c in Y such that $S(0, 0) = s$. The goal is to let each party P_i receive the column polynomial $C_i(Y) := S(i, Y)$ assigned to it so that it can recover its share $s_i := S(i, 0) \in \mathbb{Z}_p$ of the secret s. Note that the shares s_i lie on a polynomial $S(X, 0) \in \mathbb{Z}_p[X]$ of degree t_r. We follow a simple two-step approach which results in an $(n \times n)$-dimensional matrix whose entry at coordinates $a, b \in [n]$ is $S(a, b)$. First, the dealer reliably broadcasts Pedersen commitments $\{\mathsf{com}_1, \ldots, \mathsf{com}_{t_r+1}\}$ on the column polynomials $C_1(Y), \ldots, C_{t_r+1}(Y)$, from which parties can locally (by interpolation) derive the commitments $\{\mathsf{com}_1, \ldots, \mathsf{com}_n\}$ to all n column polynomials $C_1(Y), \ldots, C_n(Y)$. Following this, P_d sends each party P_i shares $\{C_1(i), C_2(i), \ldots, C_n(i)\}$ on each other party's assigned column polynomial, along with proofs that the openings are correct. This can be thought of as sending to P_i the evaluations along the row $R_i(X) := S(X, i)$. Whenever a party P_i receives a row with correct opening proofs, it sends every other party P_j the share $C_j(i)$ (along with the proof sent by the dealer) on its column polynomial $C_j(Y)$. In this way, it is guaranteed that each party P_i obtains at least $t_c + 1$ shares on its column polynomial $C_i(Y)$ and can recover its share $s_i = C_i(0) = S(i, 0)$ of the dealer's initial secret s. To guarantee unanimous termination, we employ a Bracha-style termination gadget [Bra84] in which parties send their approval to all parties upon receiving a correct row, and echo other parties' approvals upon seeing a total of $n - t_c$ approvals. HAVSS has a near-optimal communication cost of $O(\lambda n^2 \log n)$ per sharing (cf. Table 2). In combination with the aforementioned technique of superinvertible matrices, we are able to construct a packed ADKG protocol that outputs $\ell = t_c + 1 \in O(n)$ nonces with $O(\lambda n^3 \log n)$ bits and $O(1)$ rounds of communication. As a result, we achieve an amortized communication cost of $O(\lambda n^2 \log n)$ per generated Schnorr signature.

Handling Adaptive Corruptions. To prove adaptive security, our starting point is the recent work of Bacho and Loss [BL22] who introduced a new security notion for DKG protocols called *oracle-aided* *simulatability*. Loosely speaking, this notion states the existence of an efficient simulator Sim that on input k group elements $\xi_1, \dots, \xi_k \in \mathbb{G}$ can simulate an execution of the DKG protocol under adaptive corruptions while having $(k-1)$-time access to a discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$. With this notion of security for DKG, they show a reduction from the one-more discrete logarithm (OMDL) assumption [BNPS03] of degree k to the unforgeability of the threshold BLS signature scheme against an adaptive adversary. Their reduction internally runs Sim (on input the OMDL challenge $\xi_1, \dots, \xi_k \in \mathbb{G}$) in order to simulate an execution of the DKG protocol as part of the broader simulation of the unforgeability experiment. To emulate the oracle $\mathsf{DL}_{\mathbb{G},g}$ for the simulator Sim, the reduction simply forwards any query Sim makes to its own oracle. We want to employ a similar strategy to simulate the executions for multi-nonce and key generation. However, we encounter several challenges when trying to adopt this strategy naively. For the remainder of this overview, we assume for simplicity that parties employ a regular single-output ADKG protocol for nonce generation instead of a packed one.

Challenges in Our Context. Very recently, Crites et al. [CKM23a] gave an adaptive security proof for their threshold Schnorr signature scheme under the algebraic OMDL assumption[1]. In their proof, corruption queries are simulated using the oracle $\mathsf{DL}_{\mathbb{G},g}$ and signing queries are simulated using honest-verifier zero-knowledge and by programming the random oracle suitably. Omitting details, their simulator essentially samples random signature shares $\sigma_i \leftarrow\$ \mathbb{Z}_p$ for honest parties and retroactively defines the public nonce shares R_i by suitably programming the random oracle. To make this strategy work in our context, the (packed) ADKG protocol NDKG for nonce generation would have to be *fully secret* in the sense of Gennaro et al. [GJKR99], i.e., there exists an efficient simulator that on input a group element $R \in \mathbb{G}$ can simulate an execution of NDKG that terminates with R as public nonce. Unfortunately, without resorting to tools such as secure erasures or non-committing encryption, this notion of security seems to be hopeless to achieve in the context of adaptive corruptions [BL22, GJKR07, JL00]. Therefore, to deal with signing queries under adaptive corruptions, a new approach is required.

Combining Different Proof Strategies. In their work, Bellare et al. [BTZ22] provide a security reduction from the OMDL assumption to the security of the FROST1 and FROST2 schemes under a static adversary. Their reduction uses the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ to answer certain signing queries. Essentially, the values for the nonce R_j, the challenge c_j, and the public key shares $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ are fixed and determine the signature shares $\sigma_{j,i}$ by the relation $g^{\sigma_{j,i}} = R_j \cdot \mathsf{pk}_i^{c_j}$. We observe that a similar strategy could be useful for our scheme, in particular in combination with previously explained oracle-aided simulatability. And indeed, combining these two proof strategies [BL22, BTZ22] (almost!) succeeds: using oracle-aided simulators to simulate executions of IDKG and NDKG along with corruption queries[2], and at the same time using the oracle $\mathsf{DL}_{\mathbb{G},g}$ separately to answer signing queries. However, trying to employ this approach as it currently stands, we exceed the number of allowed queries to the oracle $\mathsf{DL}_{\mathbb{G},g}$ prescribed by the OMDL challenge: assume the reduction simulating the unforgeability game uses $\mathsf{DL}_{\mathbb{G},g}$ on input $g^{\sigma_{j,i}}$ to answer a signing query for party P_i and nonce R_j. If P_i gets corrupted later on, the simulators for IDKG and the j-th execution of NDKG that generated R_j might make discrete logarithm queries such that they can internally compute the secret key share sk_i and the secret nonce share $r_{j,i}$, respectively. Three discrete logarithm oracle queries have been made to return the values $\sigma_{j,i}, \mathsf{sk}_i, r_{j,i}$, although by the identity $\sigma_{j,i} = c_j \cdot \mathsf{sk}_i + r_{j,i}$ two queries would suffice. To resolve this issue we have to (i) adapt the original definition of oracle-aided simulatability delicately and (ii) cleverly design the reduction to limit the number of its queries to $\mathsf{DL}_{\mathbb{G},g}$.

1.2 More on Related Work

We discuss related work on threshold signatures and DKG. In Supplementary Material Section A, we discuss related work on AVSS and threshold Schnorr signatures with a focus on robustness, high-threshold, and efficiency.

Threshold Signatures. Most of the threshold signature schemes [BCK+22, CKM+23b, KY02, LP01] focus on threshold DSA/ECDSA and threshold Schnorr [CGG+20, CKM21, CKM23a, DOK+20, GG18, KG20], mainly due to their significance in blockchain systems and cryptocurrency wallets. Among the threshold Schnorr signatures, only the work [CKM23a] provides adaptive security. Further, several protocols for threshold RSA signatures were proposed [ADN06, KY02, Sho00] from which only [ADN06]

[1] Their scheme is the first and to date only proof achieving adaptive security for any threshold Schnorr signature.
[2] Here, IDKG denotes the initial ADKG protocol employed for key generation.

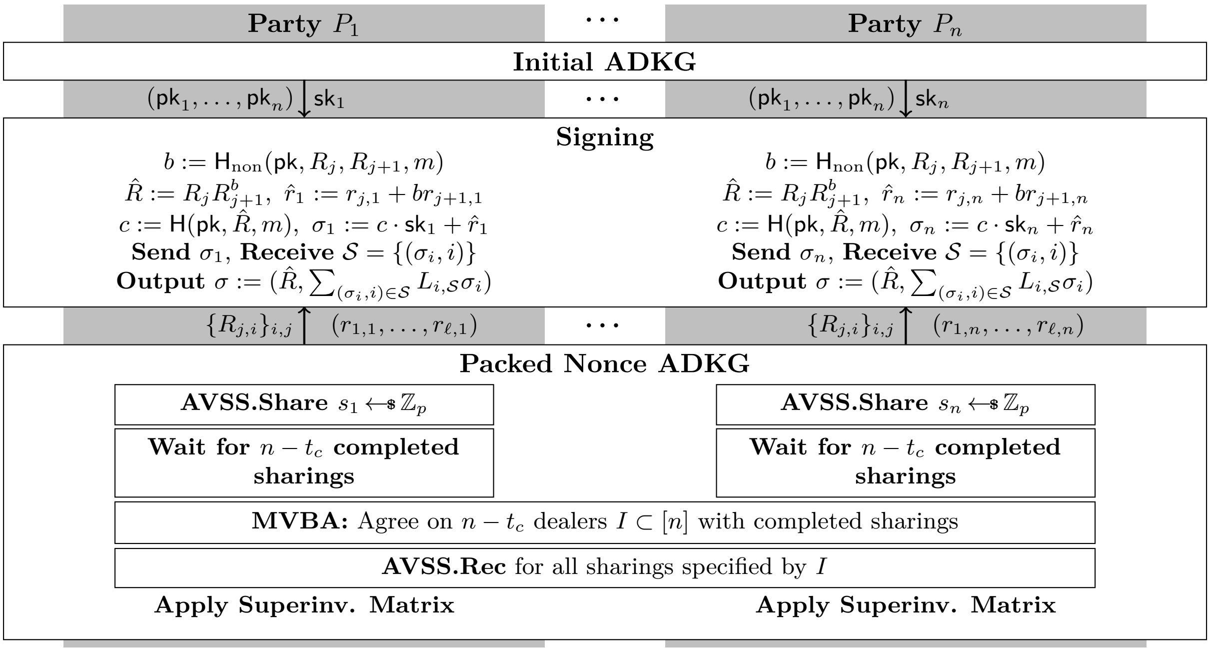

Figure 2: Overview of our protocol to generate threshold Schnorr signatures.

provides adaptive security. As in [CGJ+99, GG18, JL00], the methods to provide adaptive security include the use of reliable erasures and non-committing encryption. In the domain of pairing-based threshold signatures, there are several constructions [ACR21, Bol03, CKP+23, LJY14] from which [Bol03, LJY14] are proven adaptively secure. A recent line of works [BLT+23, DR23] study adaptively secure threshold signatures from standard assumptions.

Distributed Key Generation. Most of the DKG protocols assume an underlying synchronous network [BLL+23, CGJ+99, GJKR07, GJM+21, JL00, KG20, SBKN21]. Among these protocols, only the ones in [BLL+23, CGJ+99, JL00] provide adaptive security. However, the works [CGJ+99, JL00] rely on tools such as non-committing encryption or secure erasure of secret states. On the other hand, DKG in the asynchronous setting has only recently attracted attention [AJM+23, AJM+21, DXKKR23, DYX+22, KMS20]. Among these, only the works of Abraham et al. [AJM+23] and Kokoris-Kogias et al. [KMS20] provide adaptive security. The protocols in [AJM+23, DXKKR23, KMS20] provide high-threshold reconstruction of the key with optimal resilience threshold, but we note that [KMS20] is substantially less efficient than the other two protocols [AJM+23, DXKKR23]. All these asynchronous DKG protocols have cubic communication cost except the one in [KMS20] which has quartic communication cost.

1.3 Outline of the Paper

The paper is organized as follows. In Section 2, we define relevant preliminaries. In Section 3, we define the model of syntax and security of a robust threshold signature scheme relevant for this work. In Section 4, we give a generic construction for a high-threshold, robust, and efficient threshold Schnorr signature scheme and prove it adaptively secure in the AGM. In Section 5, we give a generic construction for an efficient packed ADKG protocol and prove it adaptively secure from its building blocks. In Section 6, we present our new high-threshold AVSS scheme and prove it adaptively secure. In Section 7, we instantiate our framework to obtain HARTS, and evaluate the communication and round complexity of it. In Supplementary Material Section A, we discuss related work on AVSS schemes and threshold Schnorr signatures. In Supplementary Material Section B, we cover additional preliminaries relevant for the paper. Due to space constraints, we defer security proofs to Supplementary Material Section C.

2 Preliminaries and Model

In this section, we fix notation and preliminaries for our paper.

General Notation. Let λ denote the security parameter. Throughout the paper, we assume that global parameters $par := (\mathbb{G}, p, g)$ implicitly parameterized by λ are fixed and known to all parties. Here, $\mathbb{G}$ is a cyclic group of prime order p generated by g. For two integers $a \leq b$, we define the set $[a, b] := \{a, \ldots, b\}$; if $a = 1$, we denote this set by $[b]$, and if $a = 0$, we denote it by $[\![b]\!]$. For an element x in a finite set S, we write $x \leftarrow\!\!\$\, S$ to denote that x was sampled from S uniformly at random. All our algorithms may be randomized, unless stated otherwise. We use the acronym PPT to mean probabilistic polynomial-time. By $x \leftarrow \mathsf{A}(x_1, \ldots, x_n)$ we denote running algorithm A on inputs $(x_1, \ldots, x_n)$ and uniformly random coins and then assigning its output to x.

Adversarial and Network Model. We consider a complete network of n parties $P_1, \ldots, P_n$ (modeled as PPT machines) connected by bilateral private and authenticated channels[3]. We consider an *asynchronous* network model, i.e., any message can be delayed arbitrarily under the constraint that messages sent between correct parties must eventually be delivered. We consider an adversary who can corrupt up to $t_c < n/3$ parties maliciously and may cause them to deviate from the protocol arbitrarily. We refer to t_c as the corruption threshold and to $t_r \in [t_c, n - t_c)$ as the reconstruction threshold. Further, the adversary is *strongly adaptive* and can choose its corruptions at any time during the protocol execution. When it corrupts a party, it can delete or substitute any undelivered messages that this party sent while being correct. We refer to the correct parties as *honest* and to the malicious parties as *corrupt*.

Public Key Infrastructure. We assume that parties have established a bulletin board public key infrastructure (PKI) before the protocol execution. Concretely, this means that every party P_i has a verification-signing key pair $(\mathsf{vk}_i, \mathsf{sik}_i)$ for a digital signature scheme, where vk_i is known to all parties but sik_i is known only to P_i. For this, we assume that each party generates its keys locally (where corrupt parties may choose their keys arbitrarily) and then makes its verification key known to everybody using a public bulletin board. These keys are used to provide authentication. In particular, we assume that parties sign each message before they send it to other parties.

Algebraic Group Model. In the algebraic group model (AGM) [FKL18], all algorithms are treated as algebraic: whenever an algorithm outputs a group element, it must also provide a representation of that element with respect to all of the inputs the algorithm has received so far. Formally, an algorithm A is called *algebraic* (over a group $\mathbb{G}$) if for all group elements $h \in \mathbb{G}$ that A outputs, it additionally outputs a vector $\mathbf{z}_\zeta = (z_1, \ldots, z_m)$ of integers such that $h = \prod_{i \in [m]} g_i^{z_i}$, where $\zeta = (g_1, \ldots, g_m) \in \mathbb{G}^m$ is the list of group elements A has received so far.

Computational Assumptions. We rely on the *one-more discrete logarithm (OMDL) assumption* [BNPS03] for our security proofs. Throughout the paper, we denote by $\mathsf{DL}_{\mathbb{G},g}$ an oracle that on input $\xi := g^z \in \mathbb{G}$ returns the discrete logarithm $z \in \mathbb{Z}_p$ of ξ to base g.

Definition 2.1 (OMDL Assumption). Let $\mathbb{G}$ be a cyclic group of prime order p generated by g and $\mathsf{DL}_{\mathbb{G},g}$ as defined above. For an algorithm A and $k \in \mathbb{N}$, we consider the following experiment:

- *Offline Phase.* Sample $(z_1, \ldots, z_k) \leftarrow\!\!\$\, \mathbb{Z}_p^k$ and set $\xi_i := g^{z_i} \in \mathbb{G}$ for all $i \in [k]$.
- *Online Phase.* Run A on input $(\mathbb{G}, p, g)$ and $(\xi_1, \ldots, \xi_k)$. Here, A gets access to the oracle $\mathsf{DL}_{\mathbb{G},g}$.
- *Winning Condition.* Let $(z'_1, \ldots, z'_k)$ denote the output of A. Return 1 if (i) $z'_i = z_i$ for all $i \in [k]$, and (ii) $\mathsf{DL}_{\mathbb{G},g}$ was queried at most $k - 1$ times during the online phase. Otherwise, return 0.

We say that the one-more discrete logarithm assumption of degree k holds relative to $(\mathbb{G}, p, g)$ if for any PPT algorithm A, the probability that the above experiment outputs 1 is negligible in λ.

Further, the discrete logarithm assumption (DLOG) is the one-more discrete logarithm assumption of degree $k = 1$.

2.1 Cryptographic and Consensus Primitives

In this section, we formally define syntax and security notions of the cryptographic and consensus primitives used in the paper.

[3]When implementing those channels, one has to make sure that they are secure in the presence of adaptive corruptions. For an example implementation, we refer to the early works [BH93, JL00].

Multivalued Validated Byzantine Agreement. A *multivalued validated Byzantine agreement (MVBA) protocol* [CKPS01] allows a set of parties, each holding an input $v_i \in V$ from a value set V with $|V| \geq 2$, to agree on a common output value $v \in V$ satisfying a predefined external validity function $\mathsf{Val}\colon V \to \{0,1\}$. A value $v \in V$ is said to be *externally valid* if $\mathsf{Val}(v) = 1$. We formally define an MVBA protocol as follows.

Definition 2.2 (MVBA Protocol). Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i holds $v_i \in V$ as input, and let $\mathsf{Val} : V \to \{0,1\}$ be an external validity function. We say that Π is a (t_c, n)-secure MVBA protocol if whenever at most t_c parties are corrupted the following properties hold:

- *External Validity.* If every honest party's input is externally valid, then every honest party P_i outputs an externally valid value v_i.
- *Consistency.* If every honest party's input is externally valid, then all honest parties output the same value v.
- *Termination.* If every honest party's input is externally valid, then every honest party P_i terminates with an output value v_i.

Hereafter, we write MVBA to denote a generic (t_c, n)-secure multivalued validated Byzantine agreement protocol.

Reliable Broadcast. A *reliable broadcast (RBC) protocol* [Bra84] allows a designated party P_s (called sender) to consistently distribute a message among all parties. In contrast to synchronous broadcast, reliable broadcast does not require full termination. We formally define a reliable broadcast protocol as follows.

Definition 2.3 (RBC Protocol). Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where a designated sender P_s holds $v \in V$ as input. We say that Π is a (t_c, n)-secure RBC protocol if whenever at most t_c parties are corrupted the following properties hold:

- *Validity.* If the sender P_s is honest and holds v as input, then every honest party P_i outputs $v_i = v$.
- *Consistency.* All honest parties that output a value output the same value v'.
- *Totality.* If an honest party outputs a value, then every honest party eventually outputs a value.

Hereafter, we write RBC to denote a generic (t_c, n)-secure RBC protocol.

Superinvertible Matrices. A *superinvertible (SI) matrix of dimension (ℓ, k) with $k \geq \ell$* [HN06] is a matrix $A \in \mathbb{Z}_p^{\ell \times k}$ over some field $\mathbb{Z}_p$ with the property that each of its $(\ell \times \ell)$-dimensional square submatrix A_I is invertible. Large classes of superinvertible matrices are given in [BHK+23, GS23]. Looking ahead, each party P_i applies the SI matrix A of dimension $(\ell, k) := (n - 2t_c, n - t_c)$ to its k secret shares $f_1(i), \ldots, f_k(i)$ that it received from different completed AVSS sharings. The result is ℓ new secret shares $r_1(i), \ldots, r_\ell(i)$ with the property that if at least ℓ input secrets are independent and uniformly random, then the ℓ output secrets are also guaranteed to be independent and uniformly random.

Asynchronous Verifiable Secret Sharing. An *asynchronous verifiable secret sharing (AVSS) scheme* [BCG93, CR93] consists of two protocols Share and Rec which allow a designated dealer to share a secret s over some field $\mathbb{Z}_p$ among all parties using Shamir secret sharing. Here, the threshold $t_r \in [t_c, n - t_c)$ specifies the degree of the shared polynomial f. In our definition of an AVSS scheme, we require a reconstruction protocol in which parties reconstruct exponentiated evaluations of the polynomial f at the points $\{0, 1, \ldots, n\}$. We formally define an AVSS scheme over the group $(\mathbb{G}, p, g)$ as follows.

Definition 2.4 ((t_c, t_r, n)-Threshold AVSS Scheme). Let $\Pi = (\mathsf{Share}, \mathsf{Rec})$ be a pair of protocols executed by n parties $P_1, \ldots, P_n$, where a designated dealer P_d holds a secret $s \in \mathbb{Z}_p$ as input. Upon completion of Share parties only maintain a state and do not output anything. Parties can then call Rec with their state and output a tuple of $n+1$ elements in $\mathbb{G}$ and an element in $\mathbb{Z}_p$. We say that Π is a complete (t_c, t_r, n)-threshold AVSS scheme if whenever at most t_c parties are corrupted the following properties hold:

- *Correctness.* Once the first honest party completes Share, there exists a unique polynomial $f \in \mathbb{Z}_p[X]$ of degree t_r such that every honest party P_i upon completing Rec outputs a tuple $(S, S_1, \ldots, S_n)$ of elements in $\mathbb{G}$ and an element $f(i) \in \mathbb{Z}_p$ such that $S = g^{f(0)}$ and $S_j = g^{f(j)}$ for all $j \in [n]$. If P_d is honest, then it holds that $f(0) = s$.
- *Termination.* The following properties hold.
  - If P_d is honest and all honest parties call Share, then all honest parties complete Share.
  - If all honest parties call Share and an honest party completes Share, then all honest parties complete Share.
  - If all honest parties call Rec, then all honest parties complete Rec.

Hereafter, we write $\mathsf{AVSS} := (\mathsf{Share}, \mathsf{Rec})$ to denote a generic complete (t_c, t_r, n)-threshold asynchronous verifiable secret sharing scheme. If AVSS allows for an arbitrary threshold $t_r \in [t_c, n - t_c)$, we call it a high-threshold AVSS scheme.

Remark 2.5 In this definition, we leave out a notion of secrecy and postpone it to Section 6 instead. We do this for the following reasons. (i) This allows us to provide the reader with a clearer picture of our work and not overload him with several new technical definitions right at the beginning. (ii) Our secrecy definition for an AVSS scheme is strongly motivated by the one we introduce for an ADKG protocol in the next section. As we organize this work according to a top-down structure, it makes more sense to introduce the secrecy notion after that.

Non-Interactive Zero-Knowledge Proofs. In our AVSS construction, we use non-interactive zero-knowledge (NIZK) proofs [BFM88]. Informally, a non-interactive proof system for an **NP** relation $\mathcal{R}$ with respect to a random oracle H is a pair of PPT algorithms $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$, where $\mathsf{PProve}^{\mathsf{H}}$ takes a statement x and a witness w with $(x, w) \in \mathcal{R}$ as input and outputs a proof π, and $\mathsf{PVer}^{\mathsf{H}}$ takes the statement x and the proof π as input and decides to accept or reject. Completeness requires that honestly computed proofs for $(x, w) \in \mathcal{R}$ are accepted, whereas soundness requires that no malicious prover can find an accepting proof for a false statement x, i.e., a statement such that $(x, w) \notin \mathcal{R}$ for all w. Further, zero-knowledge requires that there is a simulator that can simulate proofs without knowing w by programming the random oracle H. Finally, the system is a proof of knowledge, if there is an extractor that can extract the witness from any proof provided by the adversary. To do so, the extractor is allowed to observe the random oracle queries made by the adversary. Our definitions hence model online-extraction, which is reasonable in the algebraic group model. We postpone formal definitions to Supplementary Material Section B.

3 Packed Asynchronous DKG and Threshold Signatures

In this section, we introduce the notion of a *packed asynchronous distributed key generation (ADKG) protocol* and define our model of syntax and security of a *threshold signature scheme.*

3.1 Packed Asynchronous DKG

In a regular distributed key generation (DKG) protocol, a set of mutually distrusting parties securely establish a public-secret key pair without relying on a trusted dealer. At the end of the protocol, the public key is output in the clear, whereas the secret key is kept as a virtual secret distributed among all parties. This shared secret key can then be used for threshold cryptosystems, such as threshold signatures or threshold encryption, without ever being explicitly reconstructed. When the underlying network is asynchronous, we call it an *asynchronous DKG (ADKG).* In the following, we introduce and define the notion of an (ℓ, t_c, t_r, n)*-packed ADKG protocol* which allows n parties out of which at most t_c are corrupted to generate $\ell \geq 1$ independent shared keys each with reconstruction threshold $t_r \in [t_c, n - t_c)$ in a way that is potentially more efficient than just executing ℓ instances of an ADKG protocol in parallel. The basic idea is to realize the same functionality as if ℓ *independent* instances of an ADKG protocol were run in parallel. For the definition, we use the group $\mathbb{G}$ specified by $par = (\mathbb{G}, p, g)$. Hereafter, fix the parameter $\delta_a := t_r + 1 - t_c$. The subscript stands for *asynchrony*, since the two thresholds t_r and t_c coincide in synchrony.

Definition 3.1 ((ℓ, t_c, t_r, n)-Packed ADKG Protocol). Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where for each $j \in [\ell]$, P_i outputs a secret key share $r_{j,i}$, a vector of public key shares $(R_{j,1}, \ldots, R_{j,n})$, and a public key R_j. We say that Π is an *oracle-aided secure* (ℓ, t_c, t_r, n)*-packed ADKG protocol* if whenever at most t_c parties are corrupted the following properties hold:

- *Consistency.* For each $j \in [\ell]$, all honest parties output the same public key $R_j = g^{x_j}$ and the same vector of public key shares $(R_{j,1}, \ldots, R_{j,n})$.
- *Correctness.* For each $j \in [\ell]$, there exists a unique polynomial $f_j \in \mathbb{Z}_p[X]$ of degree t_r such that for all $i \in [n]$, $r_{j,i} = f_j(i)$ and $R_{j,i} = g^{r_{j,i}}$. Moreover, $R_j = g^{f_j(0)}$.
- *Termination.* If all honest parties participate in the protocol execution, then all honest parties terminate with an output.
- *Oracle-aided Simulatability.* There exists $k \in \mathsf{poly}(\lambda)$ with $k \geq \ell(t_r + 1)$ such that for any PPT algorithm A, there exists an algebraic PPT simulator Sim that on input $\xi := (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$ makes $k' := k - \ell\delta_a$ queries to the oracle $\mathsf{DL}_{\mathbb{G},g}$ and such that:
  - *Syntax.* Sim simulates the role of the honest parties in an execution of Π. At the end of the simulation, Sim outputs the public keys $R_1, \ldots, R_\ell$ and public key shares $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$.
  - *Queries upon Corruption.* Denote by $\mathcal{C} \subset [n]$ the dynamic set of corrupted parties. Once the first honest party outputs $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$, the following holds. Upon corruption query $i \in [n] \setminus \mathcal{C}$, Sim invokes $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,i} = g^{r_{j,i}}$ for all $j \in [\ell]$ among (possibly) other input elements. Conversely, it does not query $R_{j,i}$ for any $j \in [\ell]$ before that corruption.
  - *Query Independence.* Let $\mathcal{C}$ be as before and $\mathcal{H} := [n] \setminus \mathcal{C}$. Assume that $|\mathcal{C}| = t_c$ after a simulation of Π. For $i \in [k - \ell\delta_a]$, denote by $g_i \in \mathbb{G}$ the i-th query to $\mathsf{DL}_{\mathbb{G},g}$ and let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ be the corresponding algebraic vector, i.e., $g_i = g^{\hat{a}_i} \cdot \xi_1^{a_{i,1}} \cdot \ldots \cdot \xi_k^{a_{i,k}}$. Further, denote by $(\hat{b}_{\mathbf{i}}, b_{\mathbf{i},1}, \ldots, b_{\mathbf{i},k})$ the algebraic vector of the public key share $R_{\mathbf{i}}$ for all $\mathbf{i} = (j,i) \in [\ell] \times \mathcal{H}$. Then for all $\mathcal{I} := I^\ell \subset \mathcal{H}^\ell$ with $|I| = \delta_a$, the following matrix is invertible over $\mathbb{Z}_p$

$$L(\mathcal{I}, \mathcal{C}) := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-\ell\delta_a,1} & a_{k-\ell\delta_a,2} & \cdots & a_{k-\ell\delta_a,k} \\ b_{\mathbf{i}_1,1} & b_{\mathbf{i}_1,2} & \cdots & b_{\mathbf{i}_1,k} \\ \vdots & \vdots & & \vdots \\ b_{\mathbf{i}_{\ell\delta_a},1} & b_{\mathbf{i}_{\ell\delta_a},2} & \cdots & b_{\mathbf{i}_{\ell\delta_a},k} \end{pmatrix} \in \mathbb{Z}_p^{k \times k},$$

    where the indices $\mathbf{i}_{(\cdot)}$ range over the set $\bigcup_{j \in [\ell]} (\{j\} \times I)$. Whenever Sim completes a simulation of an execution of Π, we call $L(\mathcal{I}, \mathcal{C})$ the *simulatability matrix* of Sim (for this particular simulation and the set $\mathcal{I}$). Further, we call k a *simulatability factor* of Π.
  - *Bad Event.* There is an event Bad, such that for any PPT algorithm A, the probability of Bad in an execution of Π with adversary A is negligible.
  - *Indistinguishability.* Denote by $\mathsf{view}_{\mathsf{A},\Pi}$ the view of A in an execution of Π. Denote by $\mathsf{view}_{\mathsf{A},\xi,\mathsf{Sim}}$ the view of A when interacting with Sim on input ξ. Then, the distributions $(\xi, \mathsf{view}_{\mathsf{A},\Pi})$ and $(\xi, \mathsf{view}_{\mathsf{A},\xi,\mathsf{Sim}})$ where $\xi \leftarrow\$\ \mathbb{G}^k$ and both distributions are conditioned on $\neg\mathsf{Bad}$ are statistically close.

For $\ell = 1$ (when the packing is trivial), we simply call Π a (t_c, t_r, n)*-threshold ADKG protocol (over* $(\mathbb{G}, p, g)$*).* Further, we call $\ell \geq 1$ the *packing parameter.*

Remark 3.2 In our above definition of oracle-aided security, we do not require the simulator Sim to terminate once it outputs the public keys, but only after it has made the required $k - \ell\delta_a$ calls to the oracle $\mathsf{DL}_{\mathbb{G},g}$ (conditioned on the simulation of Π being completed). The reason for this being that adaptive corruptions can happen even after termination of the DKG protocol, e.g., when the DKG is part of a more complex protocol such as a threshold signature scheme.

Discussion. For simplicity, we consider only the case $\ell = 1$ in this discussion. First, note that consistency, correctness, and termination notions are in line with established definitions from the literature for DKG protocols. In addition, our definition is built upon the oracle-aided algebraic security (OAAS) notion from [BL22] which is defined for DKG protocols with a single threshold t. We adjust their definition in several ways. First, we extend it to the (t_c, t_r)-dual-threshold setting which is often relevant in asynchronous networks[4]. Second, we state a more precise requirement on the behavior of the oracle-aided simulator Sim, which is explained below. This allows us to make the DKG definition amenable for a more general framework of adaptively secure threshold signatures, such as threshold Schnorr and threshold BLS.

We begin with our new property "Queries upon Corruption" that specifies Sim's behavior when a corruption $i \in \mathcal{H}$ happens after the public key shares are defined from the protocol. Specifically, we require Sim to call $\mathsf{DL}_{\mathbb{G},g}$ on input $R_i = g^{f(i)}$ only upon that event and not before. The intuition for this being that any reasonable simulator should not know the secret key share $f(i)$ of that party P_i before the corruption happens; not surprisingly, all the OAAS simulators constructed in [BL22] have this property. We proceed with the property "Query Independence" that upon [BL22] takes a dual-threshold (t_c, t_r) into consideration. For this, we introduce the set $\mathcal{I}$. To understand this, we observe that the idea behind the invertability of the matrix $L(\mathcal{C})$ as given in their paper is that the joint secret key $f(0)$ should not be known to the simulator even after t_c corruptions happened. In the dual-threshold setting, we want this property to hold even if $t_r - t_c = \delta_a - 1$ additional secret key shares are leaked. That is why we require the algebraic vectors of any $|I| = \delta_a$ additional public key shares to be independent from already leaked data. Finally, note that [BL22] does not take computationally indistinguishable simulations into consideration. For better composability, we separate the computational and statistical aspects by introducing the property "Bad Event" and making the property "Indistinguishability" statistical.

3.2 Robust Threshold Signatures

In the following, we introduce the syntax and security notions for robust threshold signature schemes. These are in line with established definitions but adopted to the structure of our protocol.

Syntax and Completeness. In our model, a threshold signature scheme has the following structure. First, all parties $P_1, \ldots, P_n$ run a regular (t_c, t_r, n)-threshold ADKG protocol denoted by IDKG (called the *initial ADKG*). Having done this, each party P_i holds a secret key share sk_i and the public key shares $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ of other parties along with the public key pk. Following this, parties repeatedly run two parallel instances of an (ℓ, t_c, t_r, n)-packed ADKG protocol denoted by NDKG in the background. The keys generated by these executions are interpreted as nonces. In particular, after each parallel execution of the *Nonce-ADKG* protocol NDKG, the parties obtain 2ℓ new and independent nonces. To simplify matters, we assume that the nonces are output in pairs (R_j, R'_j). For each such public nonce R_j (respective R'_j), each party P_i also obtains its secret nonce share $r_{j,i}$ (respective $r'_{j,i}$) along with the public nonce shares $(R_{j,1}, \ldots, R_{j,n})$ (respective $(R'_{j,1}, \ldots, R'_{j,n})$) of other parties. For signing, we adapt the double-nonce approach introduced by Komlo and Goldberg [KG20] in order to prevent concurrent session attacks [BLL+21, DEF+19][5]. That is, we assume that parties have agreement on a previously generated but never before used nonce pair (R_j, R'_j) and use the effective nonce $\hat{R}_j = R_j R'^b_j$ to sign a message m where the scalar $b \in \mathbb{Z}_p$ is derived from a random oracle H_{non} as $b = \mathsf{H}_{\text{non}}(\mathsf{pk}, R_j, R'_j, m)$. Upon such a signing request, each party derives the effective nonce shares $(\hat{R}_{j,1}, \ldots, \hat{R}_{j,n})$ and its effective secret nonce share $\hat{r}_{j,i}$ analogously. In this light, the protocol essentially becomes *non-interactive*: When party P_i wants to sign message m with respect to effective nonce $\hat{R}_j$, it runs an algorithm SSign using its secret key sk_i and its secret nonce share $\hat{r}_{j,i}$ on message m. As a result, the party obtains a signature share σ_i that it sends to all other parties. This signature share can be verified with respect to the parties public key share pk_i and the public nonce share $\hat{R}_{j,i}$. Upon receiving $t_r + 1$ valid signature shares, a party can locally combine them into a full signature σ on m with randomness $\hat{R}_j$. This signature can now be verified with respect to the public key pk only. From this explanation of the execution model, it is clear that we can define such a threshold signature scheme by specifying the initial ADKG protocol,

[4]A dual-threshold might also be interesting in the synchronous setting, e.g., when trading off liveness with enhanced security or for responsiveness [MCK20].

[5]Nick et al. [NRS21] introduced essentially the same technique at the same time to construct a two-round Schnorr multi-signature with a rigorous security analysis.

the packed ADKG protocol for nonce generation, and algorithms for signing and verification similar to a non-interactive threshold signature scheme [BL22, LJY14].

Definition 3.3 (Threshold Signature Scheme). An (ℓ, t_c, t_r, n)-*threshold signature scheme* is a tuple of PPT protocols and algorithms $\Sigma = (\mathsf{IDKG}, \mathsf{NDKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Comb}, \mathsf{Ver})$ with the following syntax:

- IDKG: This is a (t_c, t_r, n)-threshold asynchronous DKG protocol as specified in Definition 3.1.
- NDKG: This is an (ℓ, t_c, t_r, n)-packed asynchronous DKG protocol as specified in Definition 3.1.
- SSign: The *signature share generation algorithm* takes as input a secret key share $\mathsf{sk}_i \in \mathbb{Z}_p$, a public key $\mathsf{pk} \in \mathbb{G}$, two secret nonce shares $r_i, r'_i \in \mathbb{Z}_p$, two public nonces $R, R' \in \mathbb{G}$, and a message $m \in \{0,1\}^*$. It outputs a signature share σ_i.
- SVer: The *signature share verification algorithm* takes as input a public key $\mathsf{pk} \in \mathbb{G}$, a public key share $\mathsf{pk}_i \in \mathbb{G}$, two public nonces $R, R' \in \mathbb{G}$, two public nonce shares $R_i, R'_i \in \mathbb{G}$, a message $m \in \{0,1\}^*$, and a signature share σ_i. It outputs 1 (accept) or 0 (reject).
- Comb: The *signature share combining algorithm* takes as input two public nonces $R, R' \in \mathbb{G}$, a message $m \in \{0,1\}^*$, and a set $\mathcal{S}$ of $t_r + 1$ signature shares (σ_i, i) with corresponding indices. It outputs either a signature σ or $\perp$.
- Ver: The *signature verification algorithm* takes as input a public key $\mathsf{pk} \in \mathbb{G}$, a message $m \in \{0,1\}^*$, and a signature σ. It outputs 1 (accept) or 0 (reject).

Further, we require the following correctness properties to hold:

- For any $m \in \{0,1\}^*$, any $\mathsf{sk}_i, r_i, r'_i \in \mathbb{Z}_p$, and any $\mathsf{pk}, R, R' \in \mathbb{G}$, we have

$$\Pr\left[\mathsf{SVer}(\mathsf{pk}, \mathsf{pk}_i, R, R', R_i, R'_i, m, \mathsf{SSign}(\mathsf{sk}_i, \mathsf{pk}, r_i, r'_i, R, R', m)) = 1\right] = 1,$$

  where $R_i = g^{r_i}$, $R'_i = g^{r'_i}$, and $\mathsf{pk}_i = g^{\mathsf{sk}_i}$.

- For all sets $I \subset [n]$ with $|I| = t_r + 1$, all messages $m \in \{0,1\}^*$, all polynomials $f, r, r' \in \mathbb{Z}_p[X]$ of degree t_r, and all possible sets $\mathcal{S}$ of the form $\{(\sigma_i, i)\}_{i \in I}$,

$$(\forall i \in I: \ \mathsf{SVer}(\mathsf{pk}, \mathsf{pk}_i, R, R', R_i, R'_i, m, \sigma_i) = 1) \implies \mathsf{Ver}(\mathsf{pk}, m, \mathsf{Comb}(R, R', m, \mathcal{S})) = 1,$$

  where $\mathsf{pk} = g^{f(0)}$, $R = g^{r(0)}$, $R' = g^{r'(0)}$, $\mathsf{pk}_i = g^{f(i)}$, $R_i = g^{r(i)}$, and $R'_i = g^{r'(i)}$ for all $i \in I$.

We emphasize that our definition models a *robust* threshold signing protocol [GJKR07]. The reason for this is that the protocol NDKG terminates with distributed nonces each having reconstruction threshold t_r. Since there are at least $n - t_c \geq t_r + 1$ honest parties in the system, it is guaranteed for every honest party to obtain enough valid signature shares (even if no corrupt party sends a valid signature share or anything at all) and thus to compute the full signature.

Security Model. We define the security of a threshold signature scheme following our syntax. The established security definition for non-interactive adaptively secure threshold signatures [BL22, LJY14] allows the adversary to adaptively ask for signature shares and corruptions for up to t_c parties of its choice. In the end, the adversary succeeds if it outputs a message m^* and a valid signature σ^* for it such that it obtained at most t_c signature shares for m^*. In the synchronous setting, the thresholds for corruption and reconstruction coincide[6]. As we work in an asynchronous network, we adjust their definition to a dual-threshold: the protocol should be resistant against t_c corruptions while providing security for even up to t_r leaked signature shares. Additionally, we let the adversary freely decide when parties execute a new (parallel) instance of the Nonce-ADKG protocol in which he also participates. Finally, signature shares are generated with respect to a specific nonce pair that has been generated but not used previously and is specified by the adversary.

[6] To guarantee robustness (i.e., guaranteed output delivery) in synchrony, $t < n/2$ is required. As a result, the optimal thresholds for corruption and reconstruction have to coincide.

Definition 3.4 (Unforgeability Under Chosen Message Attack). Let $\Sigma = (\mathsf{IDKG}, \mathsf{NDKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Comb}, \mathsf{Ver})$ be an (ℓ, t_c, t_r, n)-threshold signature scheme. For an algorithm A, we consider the following experiment:

1. *Setup.* Initialize a corruption set $\mathcal{C} := \emptyset$ and a signing query set $\mathcal{Q} := \emptyset$. For each party P_i, $i \in [n]$, initialize an empty state St_i. Run A on input par. At any point throughout the experiment, A can issue corruption queries:

   - *Corruption Query.* A corrupts a party by submitting an index $i \in [n] \setminus \mathcal{C}$. In this case, set $\mathcal{C} := \mathcal{C} \cup \{i\}$ and return the internal state St_i of party P_i to A. Henceforth, A has full control over P_i.

2. *Initial Asynchronous DKG.* Initiate an execution of IDKG among parties $P_1, \ldots, P_n$, where at any point in time, A controls all parties P_i with $i \in \mathcal{C}$, and the experiment simulates all other parties following the protocol, and adds their respective state to St_i. Denote by pk and $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ the public key and public key shares determined by IDKG. Denote by sk_i for all $i \in [n] \setminus \mathcal{C}$ the secret key shares of the honest parties. When the execution of IDKG is terminated, add sk_i to St_i for all $i \in [n] \setminus \mathcal{C}$.

3. *Online Phase.* During this phase, A gets additional access to oracles that answer queries of the following types:

   - *Nonce-ADKG Query.* When A queries this oracle, a new parallel protocol execution[7] of NDKG among the parties $P_1, \ldots, P_n$ is initiated. As for the initial distributed key generation, A controls all parties P_i with $i \in \mathcal{C}$, and the experiment simulates all other parties following the protocol, and adds their respective state to St_i. When this protocol terminates for the $(k+1)$-th time, let $(R_{k\ell+1}, R'_{k\ell+1}), \ldots, (R_{k\ell+\ell}, R'_{k\ell+\ell})$ be the respective public nonce pairs, and let $R_{k\ell+j,1}, \ldots, R_{k\ell+j,n}$ and $R'_{k\ell+j,1}, \ldots, R'_{k\ell+j,n}$ for each $j \in [\ell]$ be the respective public nonce shares. Further, for each party P_i with $i \in [n] \setminus \mathcal{C}$, let $(r_{k\ell+1,i}, r'_{k\ell+1,i}), \ldots, (r_{k\ell+\ell,i}, r'_{k\ell+\ell,i})$ be the respective secret nonce share pairs that party P_i obtains. These secret nonce share pairs are added to St_i.
   - *Signing Query.* When A submits a new tuple $(i, j, m) \notin \mathcal{Q}$ for an $i \in [n] \setminus \mathcal{C}$ and nonce index j such that (R_j, R'_j) is defined, then: If there is an $m' \neq m$ and an $i' \in [n]$ such that $(i', j, m') \in \mathcal{Q}$, then return $\perp$. Otherwise, set $\mathcal{Q} := \mathcal{Q} \cup \{(i, j, m)\}$ and return $\sigma \leftarrow \mathsf{SSign}(\mathsf{sk}_i, \mathsf{pk}, r_{j,i}, r'_{j,i}, R_j, R'_j, m)$.

4. *Winning Condition.* When A outputs a message m^* and a signature σ^*, let $\mathcal{S} \subset [n]$ denote the subset of parties for which A made a signing query for m^*, i.e., let

   $$\mathcal{S} := \{i \in [n] \mid \exists j \text{ s.t. } (i, j, m^*) \in \mathcal{Q}\}.$$

   Return 1 if $|\mathcal{C}| \leq t_c$, $|\mathcal{C} \cup \mathcal{S}| \leq t_r$, and $\mathsf{Ver}(\mathsf{pk}, m^*, \sigma^*) = 1$. Otherwise, return 0.

We say that Σ is *unforgeable under chosen message attacks (or UF-CMA secure)* if for any PPT algorithm A, the probability that the above experiment outputs 1 is negligible in λ.

4 Robust Threshold Schnorr Signatures

In this section, we provide a generic construction for a high-threshold, robust, and efficient threshold Schnorr signature scheme and analyze its security.

4.1 Our Construction

In the following, we give a generic construction for a robust threshold Schnorr signature scheme (also refer to Figure 2). Our construction is based on the technique introduced by Gennaro et al. [GJKR99, GJKR07]. In their work, they observed that in order to obtain a robust threshold Schnorr signature, the nonce itself

[7]By a parallel execution, we refer to a pair of instances of NDKG.

should be computed in a distributed threshold fashion realized via a DKG protocol. Building upon this idea, we implement the DKG protocol for nonce generation with a packed ADKG protocol. For this, let $\ell, t_c, t_r, n \in \mathbb{N}$ be natural numbers such that $t_c < n/3$ and $t_r \in [t_c, n - t_c)$.

Construction. Let IDKG be a (t_c, t_r, n)-threshold ADKG protocol and let NDKG be an (ℓ, t_c, t_r, n)-packed ADKG protocol. Further, let $\mathsf{H}, \mathsf{H}_{\text{non}} : \{0,1\}^* \to \mathbb{Z}_p$ be two hash functions (modeled as random oracles). Then, the threshold Schnorr signature scheme $\mathsf{SchnorrTS}[\mathsf{IDKG}, \mathsf{NDKG}] = (\mathsf{IDKG}, \mathsf{NDKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Comb}, \mathsf{Ver})$ is defined as follows:

- $\mathsf{SSign}(\mathsf{sk}_i, \mathsf{pk}, r_i, r'_i, R, R', m)$: Compute $b := \mathsf{H}_{\text{non}}(\mathsf{pk}, R, R', m)$ and the effective nonce $\hat{R} := RR'^b$. Further, compute $\hat{r}_i := r_i + b \cdot r'_i$ and $c := \mathsf{H}(\mathsf{pk}, \hat{R}, m)$. Return the signature share $\sigma_i := c \cdot \mathsf{sk}_i + \hat{r}_i \in \mathbb{Z}_p$.
- $\mathsf{SVer}(\mathsf{pk}, \mathsf{pk}_i, R, R', R_i, R'_i, m, \sigma_i)$: Compute $b := \mathsf{H}_{\text{non}}(\mathsf{pk}, R, R', m)$, the effective nonce $\hat{R} := RR'^b$, the effective nonce share $\hat{R}_i := R_i R_i'^b$, and further $c := \mathsf{H}(\mathsf{pk}, \hat{R}, m)$. Return 1 if $\mathsf{pk}_i^c \cdot \hat{R}_i = g^{\sigma_i}$ and 0 otherwise.
- $\mathsf{Comb}(R, R', m, \mathcal{S})$: Parse $\mathcal{S}$ as a set of $t_r + 1$ signature shares (σ_i, i) with corresponding indices. Denote the set of these indices by $\mathcal{I}$. Compute $s := \sum_{i \in \mathcal{I}} L_{i,\mathcal{I}} \sigma_i$ where $L_{i,\mathcal{I}}$ denotes the i-th Lagrange coefficient for the set $\mathcal{I}$. Further, compute the effective nonce $\hat{R} := RR'^b$ where $b := \mathsf{H}_{\text{non}}(\mathsf{pk}, R, R', m)$. Return the signature $\sigma := (\hat{R}, s)$.
- $\mathsf{Ver}(\mathsf{pk}, m, \sigma)$: Parse σ as $\sigma = (\hat{R}, s)$. Return 1 if $\mathsf{pk}^c \cdot \hat{R} = g^s$ and 0 otherwise.

4.2 Security Analysis

We proceed with the security proof of $\mathsf{SchnorrTS}[\mathsf{IDKG}, \mathsf{NDKG}]$ assuming oracle-aided security of IDKG and NDKG. For this, we give a security reduction from the hardness of the OMDL assumption to the unforgeability (cf. Definition 3.4) of our threshold signature scheme $\mathsf{SchnorrTS}[\mathsf{IDKG}, \mathsf{NDKG}]$.

Proof Intuition. We give here an intuition for our proof. The key idea of our reduction is to embed the OMDL challenge ξ into the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ of parties that are output by IDKG and into the public nonces $\{R_i, R'_i \mid i \in [\ell q_r]\}$ that are output by the q_r parallel executions of NDKG. Recall that each parallel execution outputs 2ℓ nonces that we interpret as ℓ nonce pairs. In order to do so, we employ the oracle-aided simulators Sim_0 for IDKG and $\mathsf{Sim}_j, \mathsf{Sim}'_j$ for the j-th parallel execution NDKG_j of NDKG. Corruption queries $i \in \mathcal{H}$ are handled by Sim_0 to return its secret key share sk_i (along with other internal data generated from IDKG related to P_i), and by $\mathsf{Sim}_j, \mathsf{Sim}'_j$ to return its 2ℓ secret nonce shares from NDKG_j (along with other internal data generated from NDKG_j related to P_i). Signature share queries (i, j, m) for an honest party P_i and (previously generated) nonce pair (R_j, R'_j) are handled in one of two ways. (i) If the reduction already knows $t_r + 1$ signature shares for (j, m)[8], then it computes the remaining shares by Lagrange interpolation and returns the signature share $\sigma_{j,i}$ of that party. (ii) If the reduction knows t_r or less signature shares for (j, m), then it calls the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $\mathsf{pk}^{c_j} \cdot \hat{R}_{j,i}$ to obtain $\sigma_{j,i} := c_j \cdot \mathsf{sk}_i + \hat{r}_{j,i}$ and returns it. Here, it can derive the values $c_j := \mathsf{H}(\mathsf{pk}, \hat{R}_j, m)$, $\hat{R}_{j,i} := R_{j,i} R_{j,i}'^{b_j}$, and $b_j := \mathsf{H}_{\text{non}}(\mathsf{pk}, R_j, R'_j, m)$ by itself from local computations and consistent lazy sampling for random oracle outputs (if not yet defined).

However, this approach has the subtlety that it exceeds the number of allowed calls to $\mathsf{DL}_{\mathbb{G},g}$. If the adversary A makes a signature share query (i, j, m) and later in the course of the protocol execution corrupts that same party P_i, then the reduction would have used $\mathsf{DL}_{\mathbb{G},g}$ too often: once for Sim_0 to return the secret key share sk_i, once for Sim_j to return the secret nonce share $r_{j,i}$, once for Sim'_j to return the secret nonce share $r'_{j,i}$, and once to compute the signature share $\sigma_{j,i}$. On the other hand, the identity $\sigma_{j,i} = c_j \cdot \mathsf{sk}_i + \hat{r}_{j,i}$ tells us that three calls are enough to derive those four values. To make use of this, we carefully leverage the "queries upon corruption" property of the simulators $\mathsf{Sim}_j, \mathsf{Sim}'_j$ for $j \geq 1$. More precisely, as we know that Sim_j queries the discrete logarithm oracle on the element $R_{j,i}$ upon corruption of party P_i, we simply answer this query on $R_{j,i}$ by computing $\hat{r}_{j,i} := \sigma_{j,i} - c_j \cdot \mathsf{sk}_i$ and returning the value $r_{j,i} = \hat{r}_{j,i} - b_j r'_{j,i}$ directly instead of calling $\mathsf{DL}_{\mathbb{G},g}$. In particular, we avoid redundant calls to $\mathsf{DL}_{\mathbb{G},g}$. At the end of the game, we obtain a forgery (m^*, σ^*) from A which we convert into a solution of the OMDL challenge ξ; recall that σ^* is of the form (R^*, s^*). This is done as follows. First, as A is an algebraic

[8]Recall that in our model, a message $m \in \{0,1\}^*$ is always signed with respect to a previously generated and agreed-upon nonce pair (R_j, R'_j). That is, when message m is signed, the parties have agreement on which nonce index j to use for it.

adversary, it returns the random oracle query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$ together with a representation of elements in $\mathbb{Z}_p$. Second, using the forgery (m^*, σ^*), known signature shares $\{\sigma_{j,1}, \dots, \sigma_{j,n}\}_j$, and known secret key shares sk_i from t_r parties, we can compute the secret key sk. Third, this allows us to compute all secret key shares $\mathsf{sk}_1, \dots, \mathsf{sk}_n$ and thus using the signature shares also all secret nonce shares $\{\hat{r}_{j,1}, \dots, \hat{r}_{j,n}\}_j$. Finally, by inverting the simulatability matrices of all oracle-aided simulators $\mathsf{Sim}_0, \mathsf{Sim}_1, \dots,$ we can translate the aforementioned values into an OMDL solution. We provide a full proof of the following theorem in Supplementary Material Section C.1.

Theorem 4.1 (ADKG $\longrightarrow$ Threshold Schnorr). *Let $\ell, t_c, t_r, n \in \mathbb{N}$ be natural numbers such that $t_c < n/3$ and $t_r \in [t_c, n - t_c)$. Let IDKG be an algebraic[9] oracle-aided secure (t_c, t_r, n)-threshold ADKG protocol and let NDKG be an algebraic oracle-aided secure (ℓ, t_c, t_r, n)-packed ADKG protocol. Further, let $\mathsf{H}, \mathsf{H}_{non} \colon \{0,1\}^* \to \mathbb{Z}_p$ be two random oracles. Then, the threshold Schnorr signature scheme $\mathsf{SchnorrTS}[\mathsf{IDKG}, \mathsf{NDKG}]$ (cf. Section 4.1) is UF-CMA secure in the algebraic group model under the OMDL assumption.*

5 Efficient Packed ADKG Protocol

In this section, we provide a generic construction for a packed ADKG protocol that is more efficient than naively executing many instances of a regular ADKG protocol in parallel.

5.1 Our Construction

In the following, we give a construction for an (ℓ, t_c, t_r, n)-packed ADKG protocol over $(\mathbb{G}, p, g)$ where $\ell = n - 2t_c$ and $t_r \in [t_c, n - t_c)$ is arbitrary. We start by describing the protocol informally.

Packed ADKG Description. A formal description of the protocol PADKG is given in Algorithm 1. Conceptually, parties agree on $n - t_c$ AVSS sharings and use a superinvertible (SI) matrix to extract as much randomness from these as possible. In more detail, our protocol has the following four steps:

1. **Sharing.** In the first step, each party P_i shares a secret $s_i \leftarrow_{\$} \mathbb{Z}_p$ via a high-threshold AVSS. That means that s_i lies on a polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t_r. Afterwards, each party waits for $n - t_c$ AVSS sharings to complete locally and stores the corresponding indices of the dealers in a set $\mathsf{dealers}_i$. Since the network is asynchronous, each party might have a different set $\mathsf{dealers}_i$ of locally completed sharings. Therefore, parties need to agree on exactly one such set using an MVBA protocol MVBA. However, the problem is that these sets as are cannot be checked via an *external validity function* which is needed for the MVBA protocol. This issue is resolved as follows.

2. **MVBA Execution.** Once its set $\mathsf{dealers}_i$ of completed sharings reaches size $n - t_c$, party P_i sends it as a proposal prop_i to all other parties with the aim to collect at least $t_c + 1$ signatures from other parties on it that it stores in a set sigs_i. Conversely, a party P_j only issues a signature on prop_i once all AVSS sharings specified by prop_i have completed at P_j itself. Once the set sigs_i of collected signatures on prop_i reaches size $t_c + 1$ (guaranteeing that these sharings completed at an honest party and thus by completeness of the AVSS eventually also at all other honest parties), party P_i invokes MVBA on input $(\mathsf{prop}_i, \mathsf{sigs}_i)$ with external validity function $\mathsf{checkValidity}$ given by:

$$(|\mathsf{prop}| = n - t_c) \ \wedge \ (|\mathsf{sigs}| \geq t_c + 1) \ \wedge \ (\forall (j, \sigma_j) \in \mathsf{sigs} : \ \mathsf{Ver}(\mathsf{vk}_j, \mathsf{prop}, \sigma_j) = 1) \,.$$

3. **AVSS Reconstruction.** Once the MVBA terminates and parties have agreement on a set $\mathsf{dealers}$ of $n - t_c$ dealers whose sharings completed (we assume that parties order the set $\mathsf{dealers}$ by increasing party index), parties proceed with the (possibly interactive) reconstruction phase. The result is that each party P_i obtains a vector $(S_j, S_{j,1}, \dots, S_{j,n})$ of group elements in $\mathbb{G}$ such that $S_{j,k} = g^{f_j(k)}$ for $k \in [n]$ along with its secret share $s_{j,i} = f_j(i)$ for all $j \in \mathsf{dealers}$. While this phase comes for free for *fully committing* schemes like Feldman's VSS, it is required for other schemes, e.g., those that build upon KZG commitments. We note that this phase comes for free with our AVSS scheme.

[9] That is, all parties behave algebraically and can be modeled as algebraic machines.

4. **SI Matrix Application.** Having done this, each party P_i locally applies (i.e., matrix multiplication from the left) the $(\ell, n - t_c)$-dimensional superinvertible matrix SI to its secret shares arranged in a vector $(s_{j,i})_{j \in \mathsf{dealers}}$ to obtain an ℓ-dimensional vector $(r_{1,i}, \ldots, r_{\ell,i})$ of new private outputs. Additionally, P_i applies SI in the exponent to the matrix with rows $(S_j, S_{j,1}, \ldots, S_{j,n})$ for $j \in \mathsf{dealers}$ to obtain an n-dimensional vector $(R_j, R_{j,1}, \ldots, R_{j,n})$ of new public outputs for each $j \in [\ell]$. Looking ahead, these vectors constitute the public nonces and public nonce shares. These operations are captured by the algorithm $\mathsf{ApplySI}$. At the end of this phase, each party P_i outputs a set $\{(j, r_{j,i}, (R_j, R_{j,1}, \ldots, R_{j,n}))\}_{j \in [\ell]}$. For each $j \in [\ell]$, R_j is the j-th public nonce with corresponding public shares $(R_{j,1}, \ldots, R_{j,n})$ of all parties and $r_{j,i}$ is party P_i's secret share of the nonce R_j.

The idea of the final phase is the following. Only $\ell = n - 2t_c$ of the polynomials f_j shared by the parties $j \in \mathsf{dealers}$ are guaranteed to be chosen from honest parties and thus uniformly random. By taking ℓ linearly independent linear combinations specified by the superinvertible matrix SI, parties obtain ℓ new polynomials $r_1, \ldots, r_\ell \in \mathbb{Z}_p[X]$ of degree t_r shared among them that are guaranteed to be uniformly random and hidden from the adversary. By applying the SI matrix also to public output related to f_j (i.e., the public elements $S_j, S_{j,1}, \ldots, S_{j,n}$) for all $j \in \mathsf{dealers}$, parties obtain regular Feldman commitments to the polynomials $r_1, \ldots, r_\ell$ (i.e., the public elements $R_j, R_{j,1}, \ldots, R_{j,n}$ for each polynomial r_j where $j \in [\ell]$) which makes subsequent threshold signing for Schnorr signatures possible.

Algorithm 1 PADKG from the view of P_i

initialize $\mathsf{prop}_i, \mathsf{dealers}_i, \mathsf{sigs}_i, \mathsf{sharings}_i, \mathsf{nonces}_i := \emptyset$
// Share a random secret via AVSS, and wait to complete $n - t_c$ sharings
sample $s_i \leftarrow\!\!\$\, \mathbb{Z}_p$ uniformly at random
call Share, sharing the secret s_i
upon completing AVSS invocation with P_j as dealer, **do**
  $\mathsf{dealers}_i := \mathsf{dealers}_i \cup \{j\}$
  if $|\mathsf{dealers}_i| = n - t_c$ **then**
    $\mathsf{prop}_i := \mathsf{dealers}_i$
    send $\langle$"proposal", $\mathsf{prop}_i\rangle$ to all parties
// Gather proof that share completed, and agree on a set of dealers
upon receiving the first $\langle$"proposal", $\mathsf{prop}_j\rangle$ message from party P_j, **do**
  upon $\mathsf{prop}_j \subseteq \mathsf{dealers}_i$, **do** ▷ Wait to complete Share for dealers in prop_j
    send $\langle$"signature", $\mathsf{Sig}(\mathsf{sik}_i, \mathsf{prop}_j)\rangle$ to party P_j
upon receiving $\langle$"signature", $\sigma_j\rangle$ from P_j, **do**
  if $\mathsf{prop}_i \neq \emptyset$ and $\mathsf{Ver}(\mathsf{vk}_j, \mathsf{prop}_i, \sigma_j) = 1$ **then**
    $\mathsf{sigs}_i := \mathsf{sigs}_i \cup \{(j, \sigma_j)\}$
    if $|\mathsf{sigs}_i| = t_c + 1$ **then**
      call MVBA on input $(\mathsf{prop}_i, \mathsf{sigs}_i)$ and external validity function $\mathsf{checkValidity}$ ▷ Agree on $n - t_c$ dealers with at least one honest approval
// Reconstruct shared values
upon MVBA terminating with output $(\mathsf{prop}, \mathsf{sigs})$, **do**
  upon $\mathsf{prop} \subseteq \mathsf{dealers}_i$, **do**
    call Rec for the AVSS instance with P_j as dealer for all $j \in \mathsf{prop}$
// Save reconstructed sharings and apply SI matrix
upon Rec terminating for dealer P_j with output $(S_j, S_{j,1}, \ldots, S_{j,n})$ and $s_{j,i}$, **do**
  $\mathsf{sharings}_i := \mathsf{sharings}_i \cup \{(j, s_{j,i}, (S, S_{j,1}, \ldots, S_{j,n}))\}$
  if $|\mathsf{sharings}_i| = n - t_c$ **then**
    $\mathsf{nonces}_i := \mathsf{ApplySI}(\mathsf{sharings}_i)$
    output nonces_i and **terminate**

5.2 Security Analysis

We proceed with the security analysis of our generic packed ADKG protocol PADKG described before (cf. Algorithm 1). For this, we first introduce a security notion for AVSS schemes that is very similar to the oracle-aided simulatability notion for packed ADKG (cf. Definition 3.1). Then, we show that this notion for the AVSS in combination with the (regular) security of the MVBA are sufficient to obtain an oracle-aided secure packed ADKG protocol à la Algorithm 1. Here, we emphasize that the proof crucially relies on the defining property of a superinvertible matrix which is that any square submatrix is invertible.

Definition 5.1 (Oracle-aided Simulatability for AVSS). Let $\Pi = (\mathsf{Share}, \mathsf{Rec})$ be a complete (t_c, t_r, n)-threshold AVSS scheme (cf. Definition 2.4). We say that Π is an *oracle-aided secure* (t_c, t_r, n)*-threshold AVSS scheme* if it additionally has oracle-aided simulatability: There exists $k \in \mathsf{poly}(\lambda)$ with $k \geq t_r + 1$ such that for any PPT algorithm A that corrupts at most t_c parties, there exists an algebraic PPT simulator Sim that on input $\zeta := (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$ makes $k' \in \{k, k - \delta_a\}$ queries to the oracle $\mathsf{DL}_{\mathbb{G},g}$ (recall that $\delta_a := t_r + 1 - t_c$) and such that:

- *Syntax.* Sim simulates the role of the honest parties in an execution of Π. At the end of the simulation, Sim outputs the tuple $(S, S_1, \ldots, S_n)$.
- *Dealer Corruption.* If the dealer remains honest at the end of the simulation, Sim makes $k' = k - \delta_a$ queries to $\mathsf{DL}_{\mathbb{G},g}$. Otherwise, it makes $k' = k$ queries.
- *Queries upon Corruption.* Denote by $\mathcal{C} \subset [n]$ the dynamic set of corrupted parties. Once the first honest party outputs $(S_1, \ldots, S_n)$, the following holds. Upon corruption query $i \in [n] \setminus \mathcal{C}$, Sim invokes $\mathsf{DL}_{\mathbb{G},g}$ on input $S_i = g^{f(i)}$ among (possibly) other input elements. Conversely, it does not query S_i before that corruption.
- *Query Independence.* Let $\mathcal{C}$ be as before and $\mathcal{H} := [n] \setminus \mathcal{C}$. Assume that $|\mathcal{C}| = t_c$ after a simulation of Π. For all $i \in [k']$, denote by $h_i \in \mathbb{G}$ the i-th query to $\mathsf{DL}_{\mathbb{G},g}$ and let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ be the corresponding algebraic vector, i.e., $h_i = g^{\hat{a}_i} \cdot \zeta_1^{a_{i,1}} \cdot \ldots \cdot \zeta_k^{a_{i,k}}$. Further, denote by $(\hat{b}_i, b_{i,1}, \ldots, b_{i,k})$ the algebraic vector of S_i for all $i \in \mathcal{H}$. Then for all $I \subset \mathcal{H}$ with $|I| = k - k'$[10], the following matrix is invertible over $\mathbb{Z}_p$

$$L(I, \mathcal{C}) := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k',1} & a_{k',2} & \cdots & a_{k',k} \\ b_{i_1,1} & b_{i_1,2} & \cdots & b_{i_1,k} \\ \vdots & \vdots & & \vdots \\ b_{i_{|I|},1} & b_{i_{|I|},2} & \cdots & b_{i_{|I|},k} \end{pmatrix} \in \mathbb{Z}_p^{k \times k},$$

  where the indices $i_{(\cdot)}$ range over the set I. Whenever Sim completes a simulation of an execution of Π, we call $L(I, \mathcal{C})$ the *simulatability matrix* of Sim (for this particular simulation and the set I). Further, we call k a *simulatability factor* of Π.
- *Bad Event.* There is an event Bad, such that for any PPT algorithm A, the probability of Bad in an execution of Π with adversary A is negligible.
- *Indistinguishability.* Denote by $\mathsf{view}_{\mathsf{A},\Pi}$ the view of A in an execution of Π. Denote by $\mathsf{view}_{\mathsf{A},\zeta,\mathsf{Sim}}$ the view of A when interacting with Sim on input ζ. Then, the distributions $(\zeta, \mathsf{view}_{\mathsf{A},\Pi})$ and $(\zeta, \mathsf{view}_{\mathsf{A},\xi,\mathsf{Sim}})$ where $\zeta \leftarrow_{\$} \mathbb{G}^k$ and both distributions are conditioned on $\neg\mathsf{Bad}$ are statistically close.

Proof Intuition. In the following, we describe how to construct an oracle-aided simulator Sim for PADKG given oracle-aided simulators for AVSS. For this informal overview, we omit the discussion on correctness, consistency, and termination for PADKG, since these follow from standard considerations. For each $i \in [n]$, denote by Sim_i the simulator for the instance AVSS_i with dealer P_i. Assume that AVSS has simulatability factor k. Then, our simulator Sim has simulatability factor kn. Let $\xi := (\xi_1, \ldots, \xi_n)$ be elements where $\xi_i := (\xi_{i,1}, \ldots, \xi_{i,k}) \in \mathbb{G}^k$ such that $\xi_{i,j} = g^{z_{i,j}}$ for some $z_{i,j} \in \mathbb{Z}_p$. On input (par, ξ), our simulator Sim

[10] If the dealer gets corrupted, this set is empty and the matrix is defined accordingly. Otherwise, the set is of size δ_a.

does the following in an execution of PADKG described by its four phases (cf. Section 5.1). First, in the *AVSS sharing* phase, it runs Sim_i on input (par, ξ_i) for all $i \in [n]$. Second, in the *MVBA execution* phase, it runs the protocol faithfully on behalf of all honest parties. Third, the *AVSS reconstruction* phase is also handled by the simulators Sim_i for $i \in [n]$. Finally, in the *SI matrix application* phase, Sim applies SI to the reconstructed vectors from the instances AVSS_i for all $i \in \mathsf{dealers}$ to obtain $(R_{j,1}, \ldots, R_{j,n})$ for $j \in [\ell]$. As a result, Sim can output all necessary group elements in the group $\mathbb{G}$ and terminate the protocol.

Throughout the simulation up until the point in which the first honest party outputs the elements $\{(R_{j,1}, \ldots, R_{j,n})\}_{j\in[\ell]}$, a corruption query $l \in \mathcal{H}$ is forwarded to all Sim_i, $i \in [n]$, simultaneously. In that case, to answer discrete logarithm oracle queries from Sim_i on an element $h' \in \mathbb{G}$, Sim forwards it to its oracle $\mathsf{DL}_{\mathbb{G},g}$ and returns the result to Sim_i. However, once the event happens in which the first honest party outputs $\{(R_{j,1}, \ldots, R_{j,n})\}_{j\in[\ell]}$, subsequent corruption queries have to be answered differently[11]. The subtle reason for this is that Sim otherwise would make redundant calls to its oracle $\mathsf{DL}_{\mathbb{G},g}$, thus violating the required notion of "query independence". We see this as follows. Assuming party P_l gets corrupted, the notion of oracle-aided simulatability for (packed) ADKG requires the simulator Sim to query $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,l}$ for all $j \in [\ell]$ at this point in time. On the other hand, by the the notion of oracle-aided simulatability for AVSS we also know that all simulators Sim_i for $i \in [n]$ will query the discrete logarithm oracle on input $S_{i,l}$ for all $i \in [n]$ at this point in time. By definition of $R_{j,l}$, we know that for all $j \in [\ell]$,

$$r_{j,l} = m_{j,1}s_{1,l} + \ldots + m_{j,n-t_c}s_{n-t_c,l}, \qquad (\heartsuit)$$

where the $m_{j,i} \in \mathbb{Z}_p$ are the entries of the superinvertible matrix $\mathsf{SI} := (m_{j,i})_{j,i}$ and we assume for simplicity that $\mathsf{dealers} = \{1, \ldots, n - t_c\}$. Obviously, the required calls from Sim to its oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,l}$, $j \in [\ell]$, cannot be independent from the ones the individual simulators Sim_i would make on input $S_{i,l}$, $i \in [n]$. In order to deal with this issue, the simulator Sim has to return the values $s_{i,l} = \mathsf{DL}_{\mathbb{G},g}(S_{i,l})$ to Sim_i for all $i \in [l]$ differently. Concretely, it will first query $\mathsf{DL}_{\mathbb{G},g}(R_{j,l})$ for all $j \in [\ell]$ to obtain the values $\{r_{j,l} \mid j \in [\ell]\}$. Next, it will choose a random subset $\mathcal{S} \subset \mathsf{dealers} \setminus \mathcal{C}$ of size $t_c - |\mathcal{C} \cap \mathsf{dealers}|$ and query $\mathsf{DL}_{\mathbb{G},g}(S_{i,l})$ for all $i \in \mathcal{S}$ to obtain a total of t_c values $\{s_{i,l} \mid i \in \mathcal{S} \cup (\mathcal{C} \cap \mathsf{dealers})\}$. From knowledge of these values, the identities in $(\heartsuit)$, and the property of SI, Sim can compute the remaining values $s_{i,l}$ from $(\heartsuit)$ by inverting a suitable submatrix of SI and return these values to the simulators Sim_i. Still, special care has to be taken as the set $\mathcal{C}$ of corrupt parties is dynamically increasing and we have to make the counting argument of calls to $\mathsf{DL}_{\mathbb{G},g}$ rigorous. We provide a full proof of the following theorem in Supplementary Material Section C.2.

Theorem 5.2 (AVSS $\longrightarrow$ ADKG). *Let $t_c, t_r, n \in \mathbb{N}$ be natural numbers such that $t_c < n/3$ and $t_r \in [t_c, n - t_c)$. Let AVSS be an oracle-aided secure (t_c, t_r, n)-threshold AVSS scheme and let MVBA be a (t_c, n)-secure MVBA protocol. Further, let SI be a superinvertible matrix over $\mathbb{Z}_p$ of dimension $(n - 2t_c, n - t_c)$. Then, PADKG (cf. Algorithm 1) is an oracle-aided secure (ℓ, t_c, t_r, n)-packed ADKG protocol with $\ell = n - 2t_c$.*

Remark 5.3 We note that our proof does not rely on the algebraic group model. However, if the AVSS scheme is algebraic (i.e., all parties behave algebraically) and the adversary is algebraic, then all our reductions are also algebraic.

6 High-Threshold AVSS Scheme

In this section, we design a new high-threshold AVSS scheme and show that it satisfies our notion of oracle-aided simulatability for AVSS under adaptive corruptions.

6.1 Our Construction

We construct a simple high-threshold AVSS scheme $\mathsf{HAVSS} = (\mathsf{HAVSS.Share}, \mathsf{HAVSS.Rec})$, relying on bivariate polynomials and NIZK proofs for inner product relations. We refer to Algorithms 2 to 4 for formal descriptions as pseudocode from the perspective of a party P_i.

[11] Note that this can only happen under adaptive corruptions.

Building Blocks. For the construction, we assume additional t_c+1 random generators $g_0, g_1, \dots, g_{t_c} \leftarrow\!\$\, \mathbb{G}$. These can for example be derived from a random oracle. We also assume a non-interactive proof system (cf. Definition B.2) $\mathsf{PS}_{\mathsf{open}} = (\mathsf{PProve}_{\mathsf{open}}, \mathsf{PVer}_{\mathsf{open}})$ for the relation

$$\mathcal{R}_{\mathsf{open}} := \left\{ ((g, g_0, \dots, g_{t_c}, \mathsf{cm}_i, \omega, y), C_i) \;\middle|\; \mathsf{cm}_i = \prod_{j=0}^{t_c} g_j^{c_{j,i}} \wedge C_i(\omega) = y \right\},$$

and a non-interactive proof system $\mathsf{PS}_{\mathsf{exp}} = (\mathsf{PProve}_{\mathsf{exp}}, \mathsf{PVer}_{\mathsf{exp}})$ for the relation

$$\mathcal{R}_{\mathsf{exp}} := \left\{ ((g, g_0, \dots, g_{t_c}, \mathsf{cm}_i, \omega, Y), C_i) \;\middle|\; \mathsf{cm}_i = \prod_{j=0}^{t_c} g_j^{c_{j,i}} \wedge Y = g^{C_i(\omega)} \right\}.$$

Note that both relations are inner-product relations, as evaluating a polynomial at a known location is an inner product. For simplicity, we write them using the same random oracle H, while formally we should understand this as two separate random oracles. Further, we omit the elements $g, g_0, \dots, g_{t_c}$ from the statements to avoid clutter. They are always clear from the context.
Finally, we make use of two deterministic algorithms $\mathsf{Interpolate}$ and $\mathsf{ExpInterpolate}$, which is Lagrange interpolation and Lagrange interpolation in the exponent, respectively. In more detail, these algorithms work as follows:

- $\mathsf{Interpolate}$: This algorithm takes as input a set of $t_c + 1$ pairs $\{(x_i, y_i)\}_{i \in [t_c+1]}$ of field elements where $x_i \neq x_j$ for $i \neq j$. It outputs the unique polynomial $C \in \mathbb{Z}_p[X]$ of degree at most t_c such that $C(x_i) = y_i$ for all $i \in [t_c + 1]$. This can be done by computing the coefficients of the polynomial using standard Lagrange interpolation.
- $\mathsf{ExpInterpolate}$: This algorithm takes as input a vector of $t_r + 1$ group elements $(S_1, \dots, S_{t_r+1})$. It outputs a vector of $n+1$ group elements $T = (T_0, \dots, T_n)$ where $T_j = \prod_{i \in [t_r+1]} S_i^{L_{i,j}}$ for all $j \in [\![n]\!]$ and $L_{i,j}$ denotes the i-th Lagrange coefficient for the set $\{1, \dots, t_r + 1\}$ at the evaluation point j. Concretely, for all polynomials $F \in \mathbb{Z}_p[X]$ of degree at most t_r, we have $F(j) = \sum_{i=1}^{t_r+1} L_{i,j} F(i)$.

Protocol Description. As said, the formal description of HAVSS from the perspective of a party P_i is given in Algorithms 2 to 4. Conceptually, HAVSS has the following four steps:

1. **Dealer Committing Phase.** The dealer P_d samples a uniform bivariate polynomial $S \in \mathbb{Z}_p[X, Y]$ of degree t_r in X and t_c in Y such that $S(0, 0) = s$. It then generates commitments $\mathsf{cm}_1, \dots, \mathsf{cm}_{t_r+1}$ to the (univariate) column polynomials $C_1(Y) := S(1, Y), \dots, C_{t_r+1}(Y) := S(t_r + 1, Y)$ of degree t_c. Concretely, these commitments are generalized Pedersen commitments and have the following form:

   $$\mathsf{cm}_i := \prod_{j=0}^{t_c} g_j^{c_{j,i}} \quad \text{where } C_i(Y) = \sum_{j=0}^{t_c} c_{j,i} Y^j \in \mathbb{Z}_p[Y].$$

   Additionally, the dealer P_d computes for all $i \in [t_r + 1]$ the exponentiated evaluations $S_i := g^{S(i,0)}$ of the polynomial $S(X, 0)$ and NIZK proofs π_i^{exp} for the relation $\mathcal{R}_{\mathsf{exp}}$. Having done this, the dealer reliably broadcasts the message $(\mathsf{CM}, \mathsf{row}_0)$ where $\mathsf{CM} = (\mathsf{cm}_1, \dots, \mathsf{cm}_{t_r+1})$ are the commitments and $\mathsf{row}_0 := ((S_1, \pi_1^{exp}), \dots, (S_{t_r+1}, \pi_{t_r+1}^{exp}))$ are the exponentiated evaluations along with the NIZK proofs of correctness. Upon receiving this message, parties can compute commitments $(\mathsf{cm}_0, \dots, \mathsf{com}_n)$ to all column polynomials $C_0(Y), \dots, C_n(Y)$ and the exponentiated evaluations S_i for all $i \in [\![n]\!]$ using $\mathsf{ExpInterpolate}$. Here, we rely on the homomorphic properties of the commitments (cf. Remark 6.1).

2. **Dealer Distributing Rows.** The dealer proceeds by sending each party P_i the evaluations $C_1(i), \dots, C_n(i)$ along the i-th row polynomial $S(X, i)$[12]. The dealer also sends for all $j \in [n]$ proofs $\pi_{j,i}$ for the relation $\mathcal{R}_{\mathsf{open}}$ attesting that the evaluation $C_j(i)$ is correct with respect to the commitment cm_j. Upon receiving such a row along with the evaluation proofs from the dealer, each party P_i checks the correctness of the evaluations by verifying the proofs. Only in case all

[12]Note that the identity $C_j(i) = S(j, i)$ holds by definition of C_j.

proofs verify, the party P_i distributes the row among all parties. This is done by sending to party P_j the evaluation $C_j(i)$ along with the proof $\pi_{j,i}$ in a "column" message. Additionally, it sends a "vote" message to all parties. Upon receiving $t_c + 1$ "column" messages with valid proofs from other parties, a party P_i interpolates these received values to obtain a polynomial $C_i(Y) \in \mathbb{Z}_p[Y]$ of degree t_c. This constitutes its column polynomial.

3. **Voting Phase.** Upon receiving "vote" messages from $n - t_c$ parties, every party knows that at least $n - 2t_c \geq t_c + 1$ honest parties received correct rows. These honest parties have evaluation points of other party's column polynomials (one evaluation point per column polynomial) which they will forward to them. Therefore, every party will eventually receive enough points to interpolate its column polynomial and will be able to terminate. In order to signify that it thinks that all parties will be able to terminate, a party also sends a "done" message (upon receiving $n - t_c$ vote messages).
4. **Termination.** Upon receiving "done" messages from $t_c + 1$ parties, every party also sends a "done" message. Upon receiving "done" messages from $n - t_c$ parties, and acquiring its polynomial $C_i(Y)$ and the exponentiated evaluations $S_0, \ldots, S_n$ of the polynomial $S(X, 0)$ of degree t_r, the party P_i terminates. This technique of echoing "done" messages is a Bracha-style termination gadget [Bra84]. Before terminating, every party waits to receive $n - t_c$ "done" messages. Out of those messages, at least $n - 2t_c \geq t_c + 1$ were sent by honest parties. As a consequence, every party will receive at least $t_c + 1$ "done" messages (those sent by honest parties) and thus send a "done" message as well. This guarantees eventual termination of all parties.
5. **Reconstruction.** During reconstruction, parties can simply output their shares $s_i := C_i(0)$ and the vector $S = (S_0, S_1, \ldots, S_n)$ from the information collected during the sharing phase.

Remark 6.1 In our construction, we rely on the homomorphic properties of the Pedersen commitment in order to compute Pedersen commitments for the remaining column polynomials. Here, we sketch that this is possible using algorithm $\mathsf{ExpInterpolate}$, i.e., Lagrange interpolation in the exponent. For the first $t_r + 1$ column polynomials $C_i(Y) = S(i, Y)$, $i \in [t_r + 1]$, we observe that $C_i(y) = S(i, y) = \sum_{j=1}^{t_r+1} L_{i,j} S(j, y) = \sum_{j=1}^{t_r+1} L_{i,j} C_j(y)$ for all $y \in \mathbb{Z}_p$ and all $j \in [\![n]\!]$. Since this identity holds for all $y \in \mathbb{Z}_p$, it also has to hold as an identity of polynomials $C_1, \ldots, C_{t_r+1}$ in $\mathbb{Z}_p[Y]$. As a result, we obtain the equivalent identity $S(i, Y) = \sum_{j=1}^{t_r+1} L_{i,j} S(j, Y)$ for the bivariate polynomial $S(X, Y)$. From this, we easily see that applying the same linear relation to the commitments of $S(1, Y), \ldots, S(t_r + 1, Y)$ (which are the first $t_r + 1$ column polynomials $C_1, \ldots, C_{t_r+1}$) yields a commitment to $S(i, Y)$ for any $i \in [\![n]\!]$, as required.

Algorithm 2 $\mathsf{HAVSS.Deal}(r)$

sample a uniform bivariate polynomial $S \in \mathbb{Z}_p[X, Y]$ of degree t_r in X and degree t_c in Y such that $S(0, 0) = r$
// Generate Pedersen commitments and opening proofs
for all $i \in [n]$ **do**
  $C_i(Y) = S(i, Y)$, $\mathsf{cm}_i := \prod_{j=0}^{t_c} g_j^{c_{j,i}}$ where $C_i(Y) = \sum_{j=0}^{t_c} c_{j,i} Y^j$
  $\forall j \in [n]$ $\pi_{j,i} \leftarrow \mathsf{PProve}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}_j, i, C_j(i)), C_j)$
  $\mathsf{row}_i := ((C_j(i), \pi_{j,i}))_{j \in [n]}$
  $S_i := g^{C_i(0)}$
  $\pi_i^{exp} \leftarrow \mathsf{PProve}^{\mathsf{H}}_{\mathsf{exp}}((\mathsf{cm}_i, 0, S_i), C_i)$
$\mathsf{CM} := (\mathsf{cm}_1, \ldots, \mathsf{cm}_{t_r+1})$, $\mathsf{row}_0 := ((S_1, \pi_1^{exp}), \ldots, (S_{t_r+1}, \pi_{t_r+1}^{exp}))$
securely erase the randomness for generating all proofs
reliably broadcast $\langle$"commits", $\mathsf{CM}, \mathsf{row}_0\rangle$
for every party P_i send $\langle$"row", $\mathsf{row}_i\rangle$ to P_i

6.2 Security Analysis

We proceed with the security analysis of our high-threshold AVSS scheme HAVSS (cf. Section 6.1). In the following, we give an intuition for the proof of oracle-aided simulatibility. We omit the correctness and termination properties, since these follow from standard considerations.

Algorithm 3 $\mathsf{HAVSS.Share}_i$

$\mathsf{CM} := \bot,\ S := \bot\ C_i := \bot,\ \mathsf{points}_{col,i} := \emptyset$
if P_i is the dealer with input $r \in \mathbb{Z}_p$ **then**
  $\mathsf{HAVSS.Deal}(r)$
// Check exponentiated openings, then store commitments
upon receiving a $\langle$"commits", $\mathsf{CM}', \mathsf{row}'_0\rangle$ broadcast from the dealer, **do**
  parse $\mathsf{CM}' = (\mathsf{cm}_1, \ldots, \mathsf{cm}_{t_r+1})$ and $\mathsf{row}'_0 = ((S_1, \pi_1^{exp}), \ldots, (S_{t_r+1}, \pi_{t_r+1}^{exp})$
  if $\forall j \in [t_r + 1]\ \mathsf{PVer}^{\mathsf{H}}_{\mathsf{exp}}((\mathsf{cm}_j, 0, S_j), \pi_j^{exp}) = 1$ **then**
    $\mathsf{CM} := \mathsf{ExpInterpolate}(\mathsf{CM}')$ ▷ $\mathsf{CM} = (\mathsf{cm}_0, \mathsf{cm}_1, \ldots, \mathsf{cm}_n)$
    $S := \mathsf{ExpInterpolate}((S_1, \ldots, S_{t_r+1}))$ ▷ $S = (S_0, \ldots, S_n)$
// Check all openings, then forward them
upon receiving a $\langle$"row", $\mathsf{row}_i\rangle$ message from the dealer, **do**
  upon $\mathsf{CM} \neq \bot$, **do**
    parse $\mathsf{row}_i = ((C_1(i), \pi_{1,i}), \ldots, (C_n(i), \pi_{n,i}))$
    if $\forall j \in [n]\ \mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}_j, i, C_j(i)), \pi_{j,i}) = 1$ **then**
      for every party P_j send $\langle$"column", $C_j(i), \pi_{j,i}\rangle$ to P_j
      send $\langle$"vote"$\rangle$ to every party P_i
// Collect forwarded points and interpolate them
upon receiving a $\langle$"column", $C_i(j), \pi_{i,j}\rangle$ from party P_j, **do**
  upon $\mathsf{CM} \neq \bot$, **do**
    if $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}_i, j, C_i(j)), \pi_{i,j}) = 1$ **then**
      $\mathsf{points}_{col,i} := \mathsf{points}_{col,i} \cup \{(j, C_i(j))\}$
      if $\left|\mathsf{points}_{col,i}\right| = t_c + 1$ **then** ▷ Interpolate points after receiving enough
        $C_i := \mathsf{Interpolate}(\mathsf{points}_{col,i})$
// Bracha-style termination gadget
upon receiving $\langle$"vote"$\rangle$ messages from $n - t_c$ different parties, **do**
  send a $\langle$"done"$\rangle$ message to all parties
upon receiving $\langle$"done"$\rangle$ messages from $t_c + 1$ different parties, **do**
  send a $\langle$"done"$\rangle$ message to all parties
upon receiving $\langle$"done"$\rangle$ from $n - t_c$ different parties and $S \neq \bot, C_i \neq \bot$, **do**
  terminate

Algorithm 4 $\mathsf{HAVSS.Rec}_i$

output $s_i := C_i(0)$ and $S = (S_0, S_1, \ldots, S_n)$ and **terminate**

Proof Intuition. The simulator Sim runs on an input of $k := t_r+1$ group elements $\zeta := (\zeta_1, \dots, \zeta_k) \in \mathbb{G}^k$. In order to simulate a sharing of a bivariate polynomial $S(X,Y) \in \mathbb{Z}_p[X,Y]$ of degree t_r in X and t_c in Y, the simulator Sim embeds the given t_r+1 elements $\zeta_1, \dots, \zeta_k$ into exponentiated evaluations of the polynomial $S(X,0)$ of degree t_r at the points $\{1, \dots, t_r+1\}$. Since $S(X,0)$ is of degree t_r, these t_r+1 evaluations determine the remaining evaluations in the exponent (to base g). By Lagrange interpolation in the exponent, Sim obtains evaluations of $S(X,0)$ in the exponent at all the points $\{1, \dots, n\}$. Next, it samples t_r+1 commitments $\mathsf{cm}_1, \dots, \mathsf{cm}_{t_r+1} \leftarrow\!\$\, \mathbb{G}$ to the first t_r+1 column polynomials $C_i(Y) := S(i,Y)$ uniformly at random, and interpolates them in the exponent to obtain the commitments $\mathsf{cm}_1, \dots, \mathsf{cm}_n$ to all column polynomials. From this point on, while simulating we make sure that parties' messages are consistent with the commitments and with the polynomial $S(X,0)$. This mainly involves sending messages normally while carefully generating a corrupted party P_i's view upon corruption. This is done by calling the discrete logarithm oracle (which is provided to Sim by definition of oracle-aided simulatibility) on input element $S_i := g^{S(i,0)}$ to obtain $S(i,0)$, and sampling polynomials for P_i that is consistent with these $S(i,0)$ and with the previously defined polynomials for all other corrupted parties. In this way, the simulator Sim makes at most $t_c = k - \delta_a$ calls to the discrete logarithm oracle which is the correct number of total calls according to our definition of oracle-aided simulatibility. All opening proofs, along with the exponentiated opening proofs for the S_i elements can produced by simulating the NIZKs for the relations $\mathcal{R}_{\mathsf{open}}$ and $\mathcal{R}_{\mathsf{exp}}$. We provide a full proof of the following theorem in Supplementary Material Section C.3.

Theorem 6.2 (AVSS). *Let $t_c, t_r, n \in \mathbb{N}$ be natural numbers such that $t_c < n/3$ and $t_r \in [t_c, n-t_c)$. Further, let $\mathsf{PS}_{\mathsf{open}}$ and $\mathsf{PS}_{\mathsf{exp}}$ be zero-knowledge proofs of knowledge and let the* DLOG *assumption hold relative to $(\mathbb{G}, p, g)$. Then, assuming secure erasures,* HAVSS *(cf. Algorithms 2 to 4) is an oracle-aided secure (t_c, t_r, n)-threshold AVSS scheme.*

Remark 6.3 We note that secure erasures are only used in the protocol to erase the randomness for generating the proofs (cf. Algorithm 2, Line 9). The reason for this is as follows: in our simulation, we simulate the proofs using the zero-knowledge property. Upon an adaptive corruption, we would have to provide the randomness used for generating the proofs if the protocol did not specify erasing it. Hence, we would need a stronger and non-standard kind of zero-knowledge in which one can efficiently generate consistent randomness for a simulated proof when learning the witness.

To illustrate that such an extension of zero-knowledge is natural, consider the Schnorr NIZK proof as an example. Here, we can compute the randomness r from the witness w and a simulated proof σ as $r := \sigma - c \cdot w$. For generic NIZK proofs or existing constructions of inner product arguments, however, it is not directly clear to us how to achieve this property.

7 Instantiation and Efficiency

In this section, we instantiate our framework with concrete building blocks to obtain HARTS and evaluate its communication and round complexity.

Instantiation. For an overview of our instantiation, we refer to Figure 1. Concretely, we use an upper-triangular Pascal matrix [GS23] for the superinvertible matrix. Further, we use VABA [AJM+23] for the MVBA protocol. Finally, we use our HAVSS (cf. Section 6) for the AVSS scheme with the following specifications: Bulletproofs [BBB+18] for the inner product arguments, and the protocol from [DXR22] for the reliable broadcast.

Efficiency. We evaluate the communication and round complexity of our threshold Schnorr signature scheme HARTS. In our AVSS scheme HAVSS, the dealer reliably broadcasts a vector of Pedersen commitments of size $O(\lambda n)$ along with $O(n)$ group elements and proofs of correctness. Using the reliable broadcast protocol from [DXR22] and Bulletproofs [BBB+18], this step has a communication complexity of $O(\lambda n^2 \log n)$. Further, the dealer privately sends n field elements and evaluation proofs to each party, who then disperses these values among all parties. This step also has a communication complexity of $O(\lambda n^2 \log n)$. As a result, we see that HAVSS has log-quadratic communication complexity. As each party invokes it once, the overall cost of the AVSS sharing phase is log-cubic. Next, the MVBA protocol VABA [AJM+23] has cubic communication complexity and terminates in expected constant rounds. Obviously, the application of the matrix SI in the subsequent phase is only local and does not affect the communication and round complexity of the protocol.

From this analysis, we see that the protocol PADKG (cf. Algorithm 1) for nonce generation generates $\ell = t_c + 1 \in O(n)$ nonces with a communication complexity of $O(\lambda n^3 \log n)$. Thus, we obtain an amortized communication cost of $O(\lambda n^2 \log n)$ per nonce. Finally, for signature generation, each party sends a threshold Schnorr signature share of size $O(\lambda)$ to all other parties. Since this step has a communication cost of $O(\lambda n^2)$, we find that a total of $O(\lambda n^3 \log n)$ communication is required to generate $O(n)$ signatures in expected constant rounds, as desired. We note that PADKG generates $\ell \in O(n)$ nonces in expected constant rounds, but only a single round is needed after that to sign a message or even a batch of up to ℓ messages if required.

Acknowledgments. CISPA authors are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585, and by the European Union, ERC-StG-2023-101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[ACR21] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 526–556. Springer, Heidelberg, December 2021. (Cited on page 6.)

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Heidelberg, May / June 2006. (Cited on page 5.)

[AJM+21] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *40th ACM Symposium Annual on Principles of Distributed Computing*, pages 363–373. Association for Computing Machinery, Portland, OR, USA, 2021. (Cited on page 6.)

[AJM+23] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 39–70, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 3, 4, 6, 22, 29, 46.)

[AVZ21] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold AVSS with optimal communication complexity. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 479–498. Springer, Heidelberg, March 2021. (Cited on page 4, 29.)

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. (Cited on page 22.)

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993. (Cited on page 8, 29.)

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Heidelberg, August 2022. (Cited on page 5.)

[BDK13] Michael Backes, Amit Datta, and Aniket Kate. Asynchronous computational VSS with reduced communication complexity. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 259–276. Springer, Heidelberg, February / March 2013. (Cited on page 4, 29.)

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. (Cited on page 9.)

[BH93] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 307–323. Springer, Heidelberg, May 1993. (Cited on page 7.)

[BHK+23] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. Sprint: High-throughput robust distributed schnorr signatures. Cryptology ePrint Archive, Paper 2023/427, 2023. https://eprint.iacr.org/2023/427. (Cited on page 2, 4, 8, 29, 30.)

[BJMS20] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: Laziness leads to GOD. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 120–150. Springer, Heidelberg, December 2020. (Cited on page 29.)

[BL22] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 193–207. ACM Press, November 2022. (Cited on page 4, 5, 11, 12.)

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021. (Cited on page 11.)

[BLL+23] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. Grandline: Adaptively secure dkg and randomness beacon with (almost) quadratic communication complexity. Cryptology ePrint Archive, Paper 2023/1887, 2023. https://eprint.iacr.org/2023/1887. (Cited on page 6.)

[BLT+23] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from ddh with full adaptive security. Cryptology ePrint Archive, Paper 2023/1482, 2023. https://eprint.iacr.org/2023/1482. (Cited on page 6.)

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003. (Cited on page 5, 7.)

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003. (Cited on page 6.)

[BP23] Luís T. A. N. Brandão and Rene Peralta. Nist first call for multi-partythreshold schemes, 2023. NIST IR 8214C (Initial Public Draft). (Cited on page 1.)

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, page 154–162, New York, NY, USA, 1984. Association for Computing Machinery. (Cited on page 4, 8, 20.)

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. https://eprint.iacr.org/2022/833. (Cited on page 5.)

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020. (Cited on page 5.)

[CGJ+99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Heidelberg, August 1999. (Cited on page 6.)

[CGRS23] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical schnorr threshold signatures without the algebraic group model. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 743–773, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 1, 29.)

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 88–97. ACM Press, November 2002. (Cited on page 4, 29.)

[CKM21] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. https://eprint.iacr.org/2021/1375. (Cited on page 5, 29.)

[CKM23a] Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive schnorr threshold signatures. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 678–709, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 1, 2, 5, 29.)

[CKM+23b] Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 710–742, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 5.)

[CKP+23] Elizabeth Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. In *Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part II*, page 348–382, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 6.)

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, Heidelberg, August 2001. (Cited on page 8.)

[CP17] Ashish Choudhury and Arpita Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theor.*, 63(1):428–468, jan 2017. (Cited on page 30.)

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51. ACM Press, May 1993. (Cited on page 8, 29.)

[CT05] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In Pierre Fraigniaud, editor, *Distributed Computing*, pages 503–504, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. (Cited on page 29.)

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019. (Cited on page 11.)

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988. (Cited on page 1.)

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990. (Cited on page 1.)

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 654–673. Springer, Heidelberg, September 2020. (Cited on page 5.)

[DR23] Sourav Das and Ling Ren. Adaptively secure bls threshold signatures from ddh and co-cdh. Cryptology ePrint Archive, Paper 2023/1553, 2023. `https://eprint.iacr.org/2023/1553`. (Cited on page 6.)

[DXKKR23] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5359–5376, Anaheim, CA, August 2023. USENIX Association. (Cited on page 3, 6.)

[DXR22] Sourav Das, Zhuolun Xiang, and Ling Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. Cryptology ePrint Archive, Report 2022/052, 2022. `https://eprint.iacr.org/2022/052`. (Cited on page 22.)

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE Computer Society Press, May 2022. (Cited on page 4, 6.)

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005. (Cited on page 31.)

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. (Cited on page 2, 7.)

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018. (Cited on page 5, 6.)

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999. (Cited on page 2, 3, 5, 13, 29.)

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007. (Cited on page 1, 2, 5, 6, 12, 13.)

[GJM+21] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 147–176. Springer, Heidelberg, October 2021. (Cited on page 6.)

[GKKS+22] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 296–315. Springer, Heidelberg, May 2022. (Cited on page 2, 29.)

[GS23] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2023/1175, 2023. https://eprint.iacr.org/2023/1175. (Cited on page 2, 3, 4, 8, 22, 29, 30.)

[HN06] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006. (Cited on page 2, 3, 8.)

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 221–242. Springer, Heidelberg, May 2000. (Cited on page 5, 6, 7.)

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020. (Cited on page 1, 2, 5, 6, 11, 29, 30.)

[KLX22] Julia Kastner, Julian Loss, and Jiayu Xu. The abe-okamoto partially blind signature scheme revisited. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 279–309. Springer, Heidelberg, December 2022. (Cited on page 31.)

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1751–1767. ACM Press, November 2020. (Cited on page 4, 6, 29.)

[KY02] Jonathan Katz and Moti Yung. Threshold cryptosystems based on factoring. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 192–205. Springer, Heidelberg, December 2002. (Cited on page 5.)

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010. (Cited on page 4, 29.)

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Paper 2022/374, 2022. https://eprint.iacr.org/2022/374. (Cited on page 1, 29.)

[LJY14] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014. (Cited on page 6, 12.)

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Heidelberg, December 2001. (Cited on page 5.)

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982. (Cited on page 2.)

[MCK20] Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-BFT: Optimistically responsive synchronous consensus with optimal latency or resilience. Cryptology ePrint Archive, Report 2020/406, 2020. https://eprint.iacr.org/2020/406. (Cited on page 11.)

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multisignatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 11, 40.)

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017. (Cited on page 29.)

[RRJ+22] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022. (Cited on page 1, 2, 29, 30.)

[SBKN21] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. Cryptology ePrint Archive, Report 2021/1635, 2021. `https://eprint.iacr.org/2021/1635`. (Cited on page 6.)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991. (Cited on page 1, 31.)

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000. (Cited on page 5.)

[Sho23] Victor Shoup. The many faces of schnorr. Cryptology ePrint Archive, Paper 2023/1019, 2023. `https://eprint.iacr.org/2023/1019`. (Cited on page 4, 29, 30.)

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001. (Cited on page 1.)

[SS23] Victor Shoup and Nigel P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. Cryptology ePrint Archive, Paper 2023/536, 2023. `https://eprint.iacr.org/2023/536`. (Cited on page 4, 29, 30.)

[TZ23] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, page 628–658, Berlin, Heidelberg, 2023. Springer-Verlag. (Cited on page 1.)

[WNR20] Pieter Wuille, Jonas Nick, and Tim Ruffing. Schnorr signatures for secp256k1. bitcoin improvement proposal 340. Github, January 2020. (Cited on page 1.)

[YLF+22] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbacss: How to robustly share many secrets. Proceedings of the Network and Distributed System Security Symposium (NDSS) 2022, 01 2022. (Cited on page 4.)

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 347–356. ACM, July / August 2019. (Cited on page 2, 29.)

Supplementary Material

A Additional Related Work

We discuss further related work on verifiable secret sharing and threshold Schnorr signatures with a focus on robustness, high-threshold, and efficiency.

Verifiable Secret Sharing. Verifiable secret sharing (VSS) is an essential tool in threshold cryptography and there has been done a huge amount of research on VSS for different network models and different security levels (information-theoretic and computational security). Therefore, we only focus on VSS protocols in the asynchronous setting (AVSS). There has been a long line of research on AVSS protocols [AVZ21, BDK13, BCG93, CT05, CR93, KMS20]. The earlier works in the 1990s [BCG93, CR93] provide information-theoretic security, but at the expense of huge communication complexity. The first practical AVSS was given by Cachin et al. [CKLS02] with computational security and achieves (suboptimal) cubic communication complexity. Backes et al. [BDK13] give the first AVSS with (asymptotically optimal) quadratic communication complexity. To achieves this, they use the KZG polynomial commitment scheme [KZG10] which requires trusted setup. All these protocols have low-threshold reconstruction and the computationally secure ones are only proven against a static adversary. Kokoris-Kogias et al. [KMS20] provide the first AVSS that supports high-threshold reconstruction. Their construction has cubic communication complexity. Alhaddad et al. [AVZ21] then gave the first construction of a high-threshold AVSS with optimal quadratic communication complexity. Abraham et al. [AJM+23] provide two constructions of AVSS with optimal quadratic communication complexity using KZG commitments: (i) a high-threshold AVSS that shares one secret, and (ii) a low-threshold AVSS that shares $t+1$ secrets while supporting individual reconstruction of the low-threshold secrets among other desirable properties. Further, their constructions are proven adaptively secure. Recently, Shoup and Smart [SS23] provide a (generic) low-threshold AVSS that can share $t+1$ secrets with quadratic communication complexity for a correct dealer while having cubic communication complexity for a dealer that provably misbehaves. Their AVSS works over any prime-power modulus ring and finite field.

Robustness. Many proposed schemes for threshold Schnorr signatures from the literature [CGRS23, CKM21, CKM23a, KG20, Lin22] fall short in providing the guarantee of signature output delivery even when only a single party misbehaves in a signing session. Despite many of these schemes providing identifiable aborts, i.e., the possibility to identify a signer responsible for failure in a signing session, they require strong *synchrony* assumptions on the network[13] to restore the property of robustness. Concretely, each time a signing session fails, parties replace the identified malicious signers and start a new session with a different set of signers. Despite this requiring strong synchrony assumptions, it can take up to $t_c + 1$ sequential runs of signing sessions to generate a signature successfully, thus incurring a linear blowup of $t_c + 1$ in both the communication and round complexity. Since any of these schemes has at least quadratic communication complexity, this results in potentially cubic communication complexity for a single signature. More critically, however, the synchrony assumption has been criticized for being too strong [BJMS20, PS17]: if an honest party ever experiences even a short outage or some other kind of malfunction, this party is now considered malicious. As a result, these schemes fail to work under more realistic network conditions [CR93, PS17].

High-Threshold. Initiated by the work of Ruffing et al. [RRJ+22], the last two years have seen a lot of attention on robust threshold Schnorr signatures [BHK+23, GS23, RRJ+22, Sho23]. All of them, with the exception of ROAST [RRJ+22], follow the technique introduced by Gennaro et al. [GJKR99] in order to maintain robustness. Concretely, parties run a (packed) (t_r, n)-threshold ADKG protocol with $t_r = t_c < n/3$ to generate the nonces used for Schnorr signatures. As a result, their protocols for threshold Schnorr signatures inherit this property and retain secrecy as long as at most t_c signature shares are revealed, despite there being $n - t_c > 2n/3$ honest parties in the system that keep the protocol alive. In modern systems [GKKS+22, YMR+19], however, one often demands security to hold even if up to $t_r > t_c$ signature shares are revealed. More precisely, the protocol should guarantee robustness in the presence of up to $t_c < n/3$ malicious parties, while retaining unforgeability for up to $t_r < n - t_c$ revealed signature shares. Unfortunately, all previously mentioned schemes for robust threshold Schnorr signatures [BHK+23, GS23, Sho23], with the exception of ROAST [RRJ+22], fall short in this regard

[13] In a synchronous network, messages are sent in synchronized rounds and arrive within a given time bound $\Delta > 0$ that is known to every party.

and only provide low-threshold security with $t_r = t_c < n/3$. The reason for this is that their techniques and building blocks are not applicable to the high-threshold setting. The protocols in [GS23, Sho23], e.g., employ a low-threshold AVSS [SS23] to retain efficiency and there is no obvious way to use this AVSS in the high-threshold setting. Further, these works use online error correction [CP17] which inherently requires $t_r < n/3$. Similarly, the protocol in [BHK$^+$23] employs packed secret sharing and certain consensus primitives, both of which require the low-threshold setting. Further, it only allows parties to sign batches of $O(n)$ messages (and no individual messages).

Efficiency. While ROAST [RRJ$^+$22] provides the desirable property of high-threshold reconstruction with $t_r < n - t_c$, it significantly falls back in terms of efficiency. We briefly elaborate on that. ROAST (which is an acronym for "RObust ASynchronous Threshold signatures") is a wrapper protocol that transforms a non-robust threshold signature scheme of a specific type (namely: semi-interactive, two-round, concurrently secure, identifiable aborts) into a protocol for robust and asynchronous threshold signatures. Essentially, the way it achieves this is by starting at most $n - t_c + 1$ concurrent signing sessions of the underlying threshold signature scheme in such a clever way that guarantees successful termination of at least one of these sessions. By applying the ROAST wrapper to the threshold Schnorr signature scheme FROST [KG20] (which has the required properties), this yields a robust threshold Schnorr signature scheme with $t_c < n$ and $t_r < n - t_c$[14]. Without making any further restrictions on the model, such as the existence of a semi-trusted coordinator, the resulting protocol has a total per-signature communication complexity of $O(\lambda n^3 + n^4)$ and a round complexity of $O(n)$. In contrast, the robust low-threshold schemes in [BHK$^+$23, GS23, Sho23] have per-signature communication complexity of $O(\lambda n^2)$ and a round complexity of $O(1)$. We emphasize that none of these protocols achieves adaptive security.

B Additional Preliminaries

In this section, we provide additional preliminaries.

Definition B.1 (**NP** Relation). Let $\mathcal{R}$ be a relation that contains pairs (x, w) of statement and witnesses such that (i) there exists a polynomial p such that $|w| \leq p(|x|)$ for all $(x, w) \in \mathcal{R}$, and (ii) the relation can be decided deterministically in polynomial time. In this case, we refer to $\mathcal{R}$ as an **NP** relation. We allow $\mathcal{R}$ to be parameterized implicitly by the security parameter and assume that $|x| \leq \mathsf{poly}(\lambda)$ for all $(x, w) \in \mathcal{R}$.

We define non-interactive proof systems for such **NP** relation.

Definition B.2 (Non-Interactive Proof System). Let $\mathcal{R}$ be an **NP** relation and H be a random oracle. A non-interactive proof system for $\mathcal{R}$ with respect to H is defined to be a pair $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ of PPT algorithms with oracle access to H and the following syntax:

- $\mathsf{PProve}^{\mathsf{H}}(x, w) \to \pi$ takes as input a statement x and a witness w, and outputs a proof π.
- $\mathsf{PVer}^{\mathsf{H}}(x, \pi) \to b$ is deterministic, takes as input a statement x and a proof π, and outputs a bit $b \in \{0, 1\}$.

Further, we require that the following completeness property holds: For any $(x, w) \in \mathcal{R}$, we have

$$\Pr\left[\mathsf{PVer}^{\mathsf{H}}(x, \pi) = 1 \mid \pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(x, w)\right] = 1.$$

In addition to completeness, we typically require proof systems to be sound (if no stronger notion is required). Namely, it should not be possible to come up with an accepting proof for a false statement.

Definition B.3 (Soundness). Let $\mathcal{R}$ be an **NP** relation, H be a random oracle, and $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ be a non-interactive proof system for $\mathcal{R}$ with respect to H. We say that PS is sound, if there is a PPT algorithm PExt such that for every PPT algorithm $\mathcal{A}$, the following advantage is negligible in λ:

$$\Pr\left[b = 1 \wedge \forall w : (x, w) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (x, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(1^\lambda), \\ b := \mathsf{PVer}^{\mathsf{H}}(x, \pi) \end{array}\right].$$

[14] Their notion of robustness is weaker than ours and does not guarantee signature generation for $t_c \geq n/3$ (without assuming a trusted dealer).

The non-interactive proof systems that we need should be zero-knowledge, meaning that one can simulate (by programming the random oracle) proofs without using the witness.

Definition B.4 (Zero-Knowledge). Let $\mathcal{R}$ be an **NP** relation, H be a random oracle, and $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ be a non-interactive proof system for $\mathcal{R}$ with respect to H. We say that PS is zero-knowledge, if there is a potentially stateful PPT algorithm PSim such that for every (potentially unbounded) algorithm $\mathcal{A}$ issuing a polynomial number of queries to H, the following advantage is negligible in λ:

$$\left|\Pr\left[\mathcal{A}^{\mathsf{H},\mathrm{O}_0}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{H}^{\mathsf{PSim}},\mathrm{O}_1}(1^\lambda) = 1\right]\right|,$$

where $\mathsf{H}^{\mathsf{PSim}}$ denotes a random oracle simulated by PSim, and O_b for $b \in \{0,1\}$ takes as input pairs $(x, w) \in \mathcal{R}$ and outputs $\pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(x_j, w_j)$ if $b = 0$ and $\pi \leftarrow \mathsf{PSim}(x_j)$ if $b = 1$.

Next, we define what constitutes a proof of knowledge.

Definition B.5 (Proof of Knowledge). Let $\mathcal{R}$ be an **NP** relation, H be a random oracle, and $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ be a non-interactive proof system for $\mathcal{R}$ with respect to H. We say that PS is a proof of knowledge, if there is a PPT algorithm PExt such that for every PPT algorithm $\mathcal{A}$, the following advantage is negligible in λ:

$$\Pr\left[b = 1 \wedge (x, w) \notin \mathcal{R} \;\middle|\; \begin{array}{l} (x, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(1^\lambda), \\ b := \mathsf{PVer}^{\mathsf{H}}(x, \pi), \\ w \leftarrow \mathsf{PExt}(x, \pi, \mathcal{Q}) \end{array}\right],$$

where $\mathcal{Q}$ denotes the list of all random oracle queries and the resulting outputs.

Our syntax assumes proof systems based on random oracles and that we can extract just by inspecting the random oracle queries. An instantiation of such an argument system would be the online-extractable system by Fischlin [Fis05]. However, we emphasize that this is just for readability and ease of presentation, and rewinding-based proofs, e.g., Schnorr [Sch91], are sufficient. Especially, we only extract once and do not rely on any witness indistinguishability which is known to cause problems in combination with rewinding [KLX22].

C Security Proofs

In this section, we provide the security proofs for the theorems given in the main body of the paper.

C.1 Proof for Threshold Schnorr

Proof of Theorem 4.1. Let A be an algebraic adversary against the unforgeability of the threshold Schnorr signature scheme $\mathsf{SchnorrTS}[\mathsf{IDKG}, \mathsf{NDKG}]$ under chosen message attacks (cf. Definition 3.4). We split our proof of the theorem into two parts. In the first part, we provide a simulation of the UF-CMA experiment to A via a sequence of games. In the second part, we bound A's winning probability in the final game by providing an efficient reduction against the OMDL assumption.

Before we start with the proof, we make some simplifications that are without loss of generality. First, we assume that A makes exactly t_c corruption queries. Second, we assume that A does not make signature share queries for corrupt parties. These assumptions are without loss of generality, since one could build a wrapper adversary that internally runs A, but makes enough corruption queries before producing the output forgery and provides signature shares of corrupt parties for A. Clearly, none of these assumptions changes the advantage of A, i.e., the wrapper adversary has the same advantage.

Additionally, the reader may recall that the security game assumes the adversary never issues the same signature share query twice.

Game $\mathbf{G}_0$: This is the real game. We recall the game to fix notation. The game samples system parameters $par = (\mathbb{G}, p, g)$ where $\mathbb{G}$ is a cyclic group of prime order p with a generator g. It also initializes a corruption set $\mathcal{C} := \emptyset$ and sets $\mathcal{H} := [n] \setminus \mathcal{C}$ throughout the game. Further, it initializes an empty state St_i for each party P_i, $i \in [n]$[15]. Then, the game runs A on input par with access to a corruption oracle.

[15] By abuse of notation, we will just write $P_i \in [n]$ instead of "P_i, $i \in [n]$". Similarly, we do the same for any index set of parties such as $\mathcal{C}$ or $\mathcal{H}$ instead of $[n]$.

Whenever A decides to corrupt a party $P_i \in \mathcal{H}$, the game returns the internal state St_i of party P_i to A and sets $\mathcal{C} := \mathcal{C} \cup \{i\}$. Henceforth, A gets full control over P_i. Following the setup phase, the game runs IDKG on behalf of the honest parties. Upon termination of IDKG, let pk and $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ denote the public key and public key shares determined by IDKG. Moreover, let sk_i for all $i \in \mathcal{H}$ denote the secret key shares of the honest parties. The game updates the state St_i for all honest parties accordingly. Following the termination of IDKG, the game starts the online phase of the game in which A gets additional access to the random oracles $\mathsf{H}, \mathsf{H}_{\text{non}}$, to the Nonce-ADKG oracle, and to the signing oracle. As standard, the game provides the random oracles $\mathsf{H}, \mathsf{H}_{\text{non}}$ by lazy sampling using maps $H[\cdot], H_{\text{non}}[\cdot]$. The game initializes an empty Nonce-ADKG set $\mathcal{R} := \emptyset$ and runs a new parallel execution of NDKG on behalf of the honest parties for each Nonce-ADKG query A makes. Upon termination of the $(k+1)$-th execution NDKG_{k+1} of NDKG, let $(R_{k\ell+1}, R'_{k\ell+1}), \ldots, (R_{k\ell+\ell}, R'_{k\ell+\ell})$ denote the respective public nonce pairs, and let $R_{k\ell+j,1}, \ldots, R_{k\ell+j,n}$ (respective $R'_{k\ell+j,1}, \ldots, R'_{k\ell+j,n}$) for each $j \in [\ell]$ denote the public nonce shares of $R_{k\ell+j}$ (respective $R'_{k\ell+j}$). Moreover, let $(r_{k\ell+1,i}, r'_{k\ell+1,i}), \ldots, (r_{k\ell+\ell,i}, r'_{k\ell+\ell,i})$ for $i \in \mathcal{H}$ denote the respective secret nonce share pairs of the honest party P_i. Recall that in our notation each NDKG_{k+1} is a parallel execution of two instances of the Nonce-ADKG protocol NDKG such that the nonces are output in pairs (R_j, R'_j). These nonce pairs are used for signing later to derive the effective nonce $\hat{R}_j$ upon signing request for message m. The game updates the state St_i for all honest parties accordingly. Additionally, the game updates $\mathcal{R} := \mathcal{R} \cup \{(k\ell + j, (R_{k\ell+j}, R'_{k\ell+j}))\}_{j \in [\ell]}$.

The game initializes an empty signing query set $\mathcal{Q} := \emptyset$. Whenever A submits a new query $(i, j, m) \notin \mathcal{Q}$, the game first checks if $i \in \mathcal{H}$ and $j \in \pi_1(\mathcal{R})$ where $\pi_1 : \mathbb{N} \times \mathbb{G}^2 \to \mathbb{N}$, $(x, y) \mapsto x$ is the projection onto the first coordinate; this checks if the j-th nonce pair is already defined. If one of the checks fails, the game returns $\perp$ to A. Otherwise, it further checks if there is an $m' \neq m$ and an $i' \in [n]$ such that $(i', j, m') \in \mathcal{Q}$; this checks if A already queried the signing oracle on a different message m' for this particular nonce pair. If this nonce pair was indeed already used for a signature on a different message m', then the game returns $\perp$ to A. Otherwise, it updates $\mathcal{Q} := \mathcal{Q} \cup \{(i, j, m)\}$ and returns $\sigma_{j,i} \leftarrow \mathsf{SSign}(\mathsf{sk}_i, \mathsf{pk}, r_{j,i}, r'_{j,i}, R_j, R'_j, m)$ to A. At the end of the game, A outputs a message m^* and a signature σ^*. It wins the game if the following conditions are satisfied: $|\mathcal{C}| \leq t_c$, $|\mathcal{C} \cup \mathcal{S}| \leq t_r$, and $\mathsf{Ver}(\mathsf{pk}, m^*, \sigma^*) = 1$ where $\mathcal{S} := \{i \in [n] \mid \exists j \text{ s.t. } (i, j, m^*) \in \mathcal{Q}\}$ denotes the set of parties for which A already made a signing query for m^*. Note that we do not require the nonce R^* of the forgery $\sigma^* = (R^*, s^*)$ to be among the set of already generated nonces $\mathcal{R}$, in which case $\mathcal{S}$ would be empty.

Game $\mathbf{G}_1$: This game is identical to the game before, except that the game aborts when there is a collision $H[\mathsf{pk}, \hat{R}_1, m_1] = H[\mathsf{pk}, \hat{R}_2, m_2]$ among distinct random oracle queries $(\mathsf{pk}, \hat{R}_1, m_1) \neq (\mathsf{pk}, \hat{R}_2, m_2)$ from A (for convenience, we omit pk from random oracle queries in our analysis hereafter). By a standard argument, we can bound the probability of this happening by q_h^2/p where q_h denotes an upper bound on the number of random oracle H queries A makes throughout the game. As a reminder, p denotes the order of the underlying system parameters group $\mathbb{G}$. As a consequence, we obtain the bound

$$\Pr[\mathbf{G}_0 \Rightarrow 1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \frac{q_h^2}{p} \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_2$: This game is identical to the game before, except that we introduce a coin flip $\theta \leftarrow_\$ \{0,1\}$ at the beginning of the game. In the following, we denote by q_r the number of queries A makes to the Nonce-ADKG oracle. For $i \in [q_r]$, we denote by NDKG_i the i-th execution of NDKG. Let $\sigma^* = (m^*, R^*, s^*)$ be A's forgery it outputs at the end of the game. The game aborts if (i) $\theta = 0$ and there exists an $\vartheta \in [q_r]$ and $m_\vartheta \in \{0,1\}^*$ for which $R^* = R_\vartheta R'^{b_\vartheta}_\vartheta$ where $b_\vartheta = H_{\text{non}}(\mathsf{pk}, (R_\vartheta, R'_\vartheta), m_\vartheta)$ and $(R_\vartheta, R'_\vartheta)$ has been output by NDKG_ϑ, or (ii) $\theta = 1$ and there does not exist such a tuple (ϑ, m_ϑ)[16]. Essentially, this means that the game correctly guesses whether the forgery is produced from a nonce pair along with a message that was previously output by some NDKG_i or not. Since the view of A is independent of the choice of θ, we obtain the bound

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \geq \frac{1}{2} \cdot \Pr[\mathbf{G}_1 \Rightarrow 1].$$

Game $\mathbf{G}_3$: In this game, we introduce another abort condition. By definition of oracle-aided security, there are events Bad_0 for IDKG and $\mathsf{Bad}_i, \mathsf{Bad}'_i$ for NDKG_i, $i \in [q_r]$, that each happen with negligible

[16]Hereafter, we will just write $R \leftarrow \mathsf{NDKG}_i$ to mean that R is output by NDKG_i.

probability. We let the game abort if any of these events occur. By a union bound, we obtain

$$\begin{aligned}\Pr[\mathbf{G}_2 \Rightarrow 1] &\le \Pr[\mathbf{G}_3 \Rightarrow 1] + \Pr[\mathsf{Bad}_0] + \sum_{i=1}^{q_r}\left(\Pr[\mathsf{Bad}_i] + \Pr[\mathsf{Bad}'_i]\right) \\ &\le \Pr[\mathbf{G}_3 \Rightarrow 1] + \mathsf{negl}(\lambda).\end{aligned}$$

More formally, we would build a reduction to bound the probability of event Bad_i (respectively Bad'_i). This reduction runs in an execution of a single ADKG instance and simulates the game $\mathbf{G}_2$ for the adversary by forwarding between (one of the two instances of) NDKG_i and the game in which it runs. Using the assumption that the DKGs are algebraic, this reduction can easily translate algebraic representations into the required form.

Game $\mathbf{G}_4$: In this game, we change the way the signing oracle computes the signature shares. Specifically, on input a tuple $(i,j,m) \notin \mathcal{Q}$ that is valid (recall that this means $i \in \mathcal{H}$, $j \in \pi_1(\mathcal{R})$, and there is no $m' \neq m$ and $i' \in [n]$ such that $(i', j, m') \in \mathcal{Q}$), the game does the following. It computes $g' := \hat{R}_{j,i} \cdot \mathsf{pk}_i^{c_j}$ where $c_j := \mathsf{H}(\mathsf{pk}, \hat{R}_j, m)$ and $\hat{R}_j := R_j R_j'^{b_j}$ is the effective nonce with $b_j := \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, R_j, R'_j, m)$ (the effective nonce shares $\{\hat{R}_{j,i}\}_{i\in[n]}$ are correspondingly defined) and returns the discrete logarithm value of g' to base g. Note that the game is no longer efficient now. By definition of the signature share generation algorithm SSign, this does not change A's view and we obtain

$$\Pr[\mathbf{G}_3 \Rightarrow 1] = \Pr[\mathbf{G}_4 \Rightarrow 1].$$

Game $\mathbf{G}_5$: In this game, we change the simulation of IDKG. By definition of oracle-aided security, there exists an algebraic simulator Sim_0 for IDKG with a simulatability factor $k_0 \ge t_r + 1$ that has all the properties specified in Definition 3.1. At the onset, the game samples an element $\xi_{(0)} := (\xi_1, \ldots, \xi_{k_0}) \leftarrow\!\!\$\, \mathbb{G}^{k_0}$ uniformly at random. It then simulates IDKG by running Sim_0 on input $\xi_{(0)}$. By definition of game $\mathbf{G}_3$ (event Bad_0 not happening), the view of A is statistically close to its view in the previous game. As a result, we obtain

$$\Pr[\mathbf{G}_4 \Rightarrow 1] \le \Pr[\mathbf{G}_5 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

For all $i \in [q_r]$, we define games $\mathbf{G}_{4+2i}, \mathbf{G}_{5+2i}$ in sequence as follows.

Game $\mathbf{G}_{4+2i}$: In this game, we change the simulation of the first instance of NDKG_i. Recall that we write NDKG_i to denote a parallel execution of NDKG instances. By definition of oracle-aided security, there exists an algebraic simulator Sim_i for NDKG_i with a simulatability factor $k_1 \ge \ell(t_r+1)$ that has all the properties specified in Definition 3.1. At the onset, the game samples an element $\xi_{(i)} \leftarrow\!\!\$\, \mathbb{G}^{k_1}$ uniformly at random[17]. It then simulates the first instance of NDKG_i by running Sim_i on input $\xi_{(i)}$. By definition of game $\mathbf{G}_3$ (event Bad_i not happening), the view of A is statistically close to its view in the previous game. As a result, we obtain

$$\forall i \in [q_r]: \ \Pr[\mathbf{G}_{3+2i} \Rightarrow 1] \le \Pr[\mathbf{G}_{4+2i} \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_{5+2i}$: In this game, we change the simulation of the second instance of NDKG_i. Again by definition of oracle-aided security, there exists an algebraic simulator Sim'_i for NDKG_i with a simulatability factor $k_1 \ge \ell(t_r+1)$. At the onset, the game samples an element $\xi'_{(i)} \leftarrow\!\!\$\, \mathbb{G}^{k_1}$ uniformly at random. It then simulates the second instance of NDKG_i by running Sim'_i on input $\xi'_{(i)}$. By definition of game $\mathbf{G}_3$ (event Bad'_i not happening), the view of A is statistically close to its view in the previous game. As a result, we obtain

$$\forall i \in [q_r]: \ \Pr[\mathbf{G}_{4+2i} \Rightarrow 1] \le \Pr[\mathbf{G}_{5+2i} \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Before we proceed, we summarize what we have achieved so far. We have ruled out collisions involving random oracle queries H. We have changed the signing oracle on input (i,j,m) to return the discrete logarithm value of $\hat{R}_{j,i} \cdot \mathsf{pk}_i^{c_j}$ to base g. Further, we have introduced algebraic simulators Sim_0 for the

[17] Hereafter, we will write $\xi_{(\cdot)}[j]$ to denote the j-th component of the tuple $\xi_{(\cdot)}$.

initial ADKG protocol IDKG and $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for the nonce generation protocols NDKG_i for all $i \in [q_r]$. As a result, corruption queries are now completely handled by these algebraic simulators. At this stage, we are not able to build an efficient reduction (simulating the preceding game) breaking the OMDL assumption. The reason for this is that we have not yet exploited the algebraic dependence between signature shares and secret key and nonce shares. A naive reduction would therefore exceed the number of allowed discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ calls to break the OMDL assumption. To make this point more precise, we count the number of $\mathsf{DL}_{\mathbb{G},g}$ calls a hypothetical reduction R_{hyp} would have to make in order to simulate the preceding game $\mathbf{G}_{5+2q_r}$. For the sake of simplicity, we assume that the adversary A corrupts t_c parties $\mathcal{C} \subset [n]$ right before outputting its forgery. The hypothetical reduction would make the following number of calls to the oracle: (i) up to $n\ell q_r$ calls to simulate signing for all n parties using the ℓq_r generated nonce pairs, (ii) $k_0 - \delta_a$ calls to simulate Sim_0 where $\delta_a = t_r + 1 - t_c$, and (iii) $2q_r(k_1 - \ell\delta_a)$ calls to simulate $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for all $i \in [q_r]$. On the other hand, the degree of the OMDL challenge should be $k_0 + 2q_r k_1$ to obtain the elements $\{\xi_{(0)}, \xi_{(i)}, \xi'_{(i)}\}_{i \in [q_r]}$. As a result, the total number of $\mathsf{DL}_{\mathbb{G},g}$ calls R_{hyp} would have to make is given by

$$n\ell q_r + (k_0 - \delta_a) + 2q_r(k_1 - \ell\delta_a) = (k_0 + 2q_r k_1) + (n\ell q_r - 2q_r\ell\delta_a - \delta_a).$$

The second term $(n\ell q_r - 2q_r\ell\delta_a - \delta_a)$ can be bound as

$$\ell q_r(n - 2\delta_a) - \delta_a \geq \ell q_r(4t_c - n) - (n - 2t_c) = \ell q_r(t_c - 1) - (t_c + 1),$$

which is non-negative for $t_c > 1$ and large enough q_r, and thus R_{hyp} could not solve the given OMDL challenge without exceeding the number of allowed discrete logarithm oracle calls. To resolve this issue, we make two further changes to the game that target the signing and corruption oracles. First, when the game already provided $t_r + 1$ signature shares for a particular tuple message-nonce pair $(\cdot, j, m)$, it computes the remaining shares via Lagrange interpolation (as the signature shares lie on a polynomial of degree t_r by definition). Second, the game exploits the linear equations $\sigma_{j,i} = c_j \cdot \mathsf{sk}_i + \hat{r}_{j,i}$ and $\hat{r}_{j,i} = r_{j,i} + b_j r'_{j,i}$ in order to obtain the secret nonce shares $r_{j,i}, r'_{j,i}$ upon corruption query for party P_i (instead of naively forwarding the corruption query to both simulators Sim_j and Sim'_j independently). These changes allow us to correctly limit the number of discrete logarithm calls for our reduction later. This requires some bookkeeping for which we introduce a binary array $\mathcal{S}_i := [0, \ldots, 0] \in \{0,1\}^{\ell q_r}$ for each party $i \in [n]$. Its purpose is to keep track for which nonces (R_j, R'_j) with $j \in [\ell q_r]$, the game already computed a signature share for party P_i on that particular nonce. We proceed with the next game description.

Game $\mathbf{G}_{6+2q_r}$: Recall the evolving sets $\mathcal{R}$ and $\mathcal{Q}$ of public nonces output by $\mathsf{NDKG}_1, \mathsf{NDKG}_2, \ldots$ and signing queries, respectively. Note that $|\mathcal{R}| \leq \ell q_r$, since each Nonce-ADKG generates ℓ public nonce pairs at once. At the onset, the game initializes a binary array $\mathcal{S}_i := [0, \ldots, 0] \in \{0,1\}^{\ell q_r}$ for each $i \in [n]$. Its purpose is to keep track for which nonce pairs (R_j, R'_j), $j \in [\ell q_r]$, the game already computed a signature share for P_i using that particular nonce (and some message bound to it). In this game, we further change the way the signing oracle works. This is done as follows. Whenever A makes a new signing query $(i, j, m) \notin \mathcal{Q}$, the game first checks if the query is valid. That is, if $i \in \mathcal{H}$, $j \in \pi_1(\mathcal{R})$, and there is no $m' \neq m$ such that $(\cdot, j, m') \in \mathcal{Q}$. If the validity check verifies, the game checks if $|\mathcal{C}| + \sum_{v \in \mathcal{H}} \mathcal{S}_v[j] \leq t_r$. In that case, the game retrieves $c_j := \mathsf{H}(\mathsf{pk}, \hat{R}_j, m)$, computes the discrete logarithm value of the element $\mathsf{pk}_i^{c_j} \cdot \hat{R}_{j,i}$, and returns the output denoted by $\sigma_{j,i}$ to A. Following this, the game updates the binary array for that particular party as $\mathcal{S}_i[j] := 1$ and also $\mathcal{Q} := \mathcal{Q} \cup \{(i, j, m)\}$. Note that the game does not make redundant computations of discrete logarithm values, since it by definition only replies to new signing queries that are not already stored in the set $\mathcal{Q}$. In the other case, i.e., if $|\mathcal{C}| + \sum_{v \in \mathcal{H}} \mathcal{S}_v[j] \geq t_r + 1$, the game gathers the signature shares $\{(v, \sigma_{j,v}) \mid v \in \mathcal{H},\ \mathcal{S}_v[j] = 1\} \cup \{(v, \sigma_{j,v}) \mid v \in \mathcal{C}\}$ that it already provided to A and computes the share $\sigma_{j,i}$ by standard Lagrange interpolation. By the correctness of Lagrange interpolation, this does not change A's view and we obtain

$$\forall i \in [q_r]: \ \Pr[\mathbf{G}_{5+2q_r} \Rightarrow 1] = \Pr[\mathbf{G}_{6+2q_r} \Rightarrow 1].$$

Game $\mathbf{G}_{7+2q_r}$: In this final game, we change the way the corruption oracle works. This is done as follows.

- Whenever A decides to corrupt a party P_i with $i \in [n]$, the game first checks if the query is valid, i.e., if $i \in \mathcal{H}$. If not, it returns $\perp$ to A.

- In case the query is valid, the game proceeds as follows. Let $r \in [\![q_r]\!]$ denote the number of completed and ongoing Nonce-ADKG executions up to this point. As before, the game provides discrete logarithm oracle access for Sim_0 by simulating this oracle for itself (directly computing the discrete logarithm value of the queried group element to base $g \in \mathbb{G}$). Since Sim_0 has to handle internal state data from the initial IDKG phase, this also reveals the secret key share sk_i of the corrupted party P_i to the game.

- For the remaining simulators $\{\mathsf{Sim}_j, \mathsf{Sim}'_j\}_{j\in[r]}$ (we define this set to be empty if $r = 0$), the game provides discrete logarithm oracle access differently. First, the game scans through the array $\mathcal{S}_i[\cdot]$ and computes for all $j \in [\ell r]$ such that $\mathcal{S}_i[j] = 1$ the value $\hat{r}_{j,i} := \sigma_{j,i} - \mathsf{H}(\mathsf{pk}, \hat{R}_j, m) \cdot \mathsf{sk}_i$ (note that $c_j := \mathsf{H}(\mathsf{pk}, \hat{R}_j, m)$ is already defined if $\mathcal{S}_i[j] = 1$). By definition of oracle-aided simulatability, we know that Sim_j accesses its discrete logarithm oracle on input element $R_{j,i}$ upon corruption query for P_i. For all those $j \in [\ell r]$ where $\mathcal{S}_i[j] = 1$, the game provides this oracle access on $R_{j,i}$ for Sim_j by retrieving both values $\hat{r}_{j,i}, r'_{j,i}$, where it knows $r'_{j,i}$ from the interaction with Sim'_j, and returns the computed value $r_{j,i} = \hat{r}_{j,i} - b_j r'_{j,i}$ to Sim_j. Finally, the game updates the i-th binary array of the corrupted party P_i as $\mathcal{S}_i := [1, \ldots, 1]$.

Since the interface provided to the simulators is the same, this change is independent from A's view and we therefore obtain

$$\Pr[\mathbf{G}_{7+2q_r} \Rightarrow 1] = \Pr[\mathbf{G}_{6+2q_r} \Rightarrow 1].$$

It remains to bound the probability that the final game $\mathbf{G}_{7+2q_r}$ outputs 1. For that, we build an efficient reduction R against the OMDL assumption of degree $k := k_0 + 2q_r k_1$. The way we have defined the sequence of games, R's simulation of $\mathbf{G}_{7+2q_r}$ on input an k-OMDL instance $\xi \in \mathbb{G}^k$ is straightforward. We will describe it in the following and then convert the adversary A's forgery into a solution of the challenge ξ.

Building a reduction R. The reduction R gets as input $\xi \in \mathbb{G}^k$ and simulates game $\mathbf{G}_{7+2q_r}$ for A as follows. First, it parses ξ as $(\xi_{(0)}, \xi_{(1)}, \xi'_{(1)}, \ldots, \xi_{(q_r)}, \xi'_{(q_r)})$ where $\xi_{(0)} \in \mathbb{G}^{k_0}$ and $\xi_{(i)}, \xi'_{(i)} \in \mathbb{G}^{k_1}$ for all $i \in [q_r]$. With that, it runs Sim_0 on $\xi_{(0)}$, Sim_i on $\xi_{(i)}$ and Sim'_i on $\xi'_{(i)}$ for all $i \in [q_r]$. Whenever the game simulates a discrete logarithm oracle for itself, the reduction R uses its oracle $\mathsf{DL}_{\mathbb{G},g}$ for this purpose. In the following, we describe the simulation R provides in more detail.

- *Initial ADKG Protocol.* The reduction R invokes Sim_0 on input $\xi_{(0)}$ in order to simulate the initial distributed key generation protocol. Whenever Sim_0 queries the discrete logarithm oracle, the reduction simply forwards this query to its own oracle $\mathsf{DL}_{\mathbb{G},g}$. Corruption queries for this phase are completely handled by the simulator Sim_0.

- *Nonce-ADKG Protocols.* For all $i \in [q_r]$, the reduction R invokes Sim_i on input $\xi_{(i)}$ and likewise Sim'_i on input $\xi'_{(i)}$ to simulate the i-th (parallel) execution of NDKG_i. In general, the reduction does not know the entire internal states of the honest parties as it delegated parts of the simulation so far to the oracle-aided simulators.

- *Signing Query.* Upon a new signing query $(i, j, m) \notin \mathcal{Q}$, the reduction checks if the query is valid. In that case, it checks if the sum $|\mathcal{C}| + \sum_{v\in\mathcal{H}} \mathcal{S}_v[j]$ exceeds the value t_r or not. In the first case, R calls its oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $\mathsf{pk}_i^{c_j} \cdot \hat{R}_{j,i}$, returns the output $\sigma_{j,i}$ to A, and updates the array as $\mathcal{S}_i[j] := 1$ and $\mathcal{Q} := \mathcal{Q} \cup \{(i, j, m)\}$. In the second case, R gathers already computed signature shares $\{(v, \sigma_{j,v})\}$ (where $v \in \mathcal{C}$ or $v \in \mathcal{H}$ such that $\mathcal{S}_v[j] = 1$) and computes $\sigma_{j,i}$ via Lagrange interpolation.

- *Corruption Query.* Upon a new corruption query $i \in \mathcal{H}$, the reduction does the following. Again, $r \in [\![q_r]\!]$ denotes the number of completed NDKG executions. Corruption queries are handled by the simulators $\{\mathsf{Sim}_0, \mathsf{Sim}_i, \mathsf{Sim}'_i\}_{i\in[r]}$, whereby providing access to a discrete logarithm oracle by its own oracle $\mathsf{DL}_{\mathbb{G},g}$ except for queries in $\{R_{j,i} \mid j \in [\ell r],\ \mathcal{S}_i[j] = 1\}$. For these elements, R computes $\hat{r}_{j,i} := \sigma_{j,i} - c_j \cdot \mathsf{sk}_i$ and returns $r_{j,i} := \hat{r}_{j,i} - b_j r'_{j,i}$. Note that it knows all the values on the right-hand side of the equations. Finally, it updates $\mathcal{C} := \mathcal{C} \cup \{i\}$ and the i-th binary array as $\mathcal{S}_i := [1, \ldots, 1]$.

It is clear that the reduction perfectly simulates the game for the adversary, and that its running time is dominated by the running time of the adversary.

Counting calls to $\mathsf{DL}_{\mathbb{G},g}$. Before we proceed with the forgery (m^*, R^*, s^*) output by the adversary, we count the number of queries the reduction makes to its discrete logarithm oracle. We need to show that it does not exceed the number of allowed queries for the given challenge ξ. To this end, we assume without loss of generality that (i) A makes exactly t_c corruption queries $\mathcal{C} \subset [n]$, (ii) for all non-forgery indices $j \in \pi_1(\mathcal{R}) \setminus \{\vartheta\}$, A makes at least $t_r + 1 - |\mathcal{C}|$ signature share queries (i, j, m_j) for that nonce index where $i \in \mathcal{H} = [n] \setminus \mathcal{C}$[18], and (iii) if $\vartheta \neq \perp$, A makes exactly t_r signature share queries for R^*. Again, this can trivially be enforced by building a wrapper adversary that internally runs A, but makes enough corruption and signing queries before outputting its forgery.

Recall that the given OMDL challenge ξ is of degree $k = k_0 + 2q_r k_1$ and thus at most $k - 1$ calls to the oracle are allowed. First, we observe that the simulator Sim_0 for the initial ADKG protocol IDKG gets k_0 elements as input, while the simulators $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for all $i \in [q_r]$ for the Nonce-ADKG protocols NDKG_i each gets k_1 elements as input. The way R is built, these input elements $\{\xi_{(0)}, \xi_{(i)}, \xi'_{(i)}\}_{i \in [q_r]}$ are chosen disjointly from the OMDL challenge ξ of degree $k = k_0 + 2q_r k_1$. Second, we observe that the reduction delegates corruption queries to the simulators $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for all $i \in [\![q_r]\!]$ that each handles the internal state data from NDKG_i (by abuse of notation, we interpret NDKG_0 as IDKG). By definition of oracle-aided simulatability, each simulator $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for $i \in [q_r]$ makes $k_1 - \ell\delta_a$ calls to the discrete logarithm oracle where $\delta_a = t_r + 1 - t_c$. On the other hand, the initial simulator Sim_0 makes $k_0 - \delta_a$ queries. Third, we observe that the reduction makes at most $t_r + 1$ calls to its discrete logarithm oracle for each nonce index $j \in \pi_1(\mathcal{R}) \setminus \{\vartheta\}$ to answer signing queries for (R_j, R'_j, m). Having said that, we obtain the following:

- For each $j \in [\ell q_r]$, denote by $S_j \in [\![t_r + 1]\!]$ the number of calls the reduction makes to $\mathsf{DL}_{\mathbb{G},g}$ in order to answer signing queries for (R_j, R'_j). In case a party $P_i \in \mathcal{H}$ gets corrupted after the signature share $\sigma_{j,i}$ for that party and nonce index j was computed, by design the reduction saves one call to its discrete logarithm oracle. The reason for this is that by definition the oracle-aided simulator Sim_j queries the discrete logarithm value on input $R_{j,i}$ upon corruption query for party P_i. In case a party $P_i \in \mathcal{H}$ gets corrupted before the signature share $\sigma_{j,i}$ was computed, the reduction does not call its discrete logarithm oracle anymore to answer signing queries for that particular party P_i and nonce index j. Further, we denote by $\mathcal{C}_j[+1]$ the subset of parties that got corrupted after their signature share for (R_j, R'_j) was computed and by $\mathcal{C}_j[-1]$ the remaining corrupted parties (i.e., the parties for which the reduction never explicitly computed a signature share for (R_j, R'_j)). Then, we find that $S_j + |\mathcal{C}_j[-1]| = t_r + 1$ by assumption (ii) and $|\mathcal{C}_j[-1] \cup \mathcal{C}_j[+1]| = t_c$ by assumption (i). Since the reduction by design saves one call to its discrete logarithm oracle for all parties in $\mathcal{C}_j[+1]$, the difference $S_j - |\mathcal{C}_j[+1]|$ exactly represents the number of calls the reduction would make to $\mathsf{DL}_{\mathbb{G},g}$ in order to compute the signature shares for all parties and nonce (R_j, R'_j) under the assumption that all t_c corruptions happened before any signing query for (R_j, R'_j). In combination, the above expression $S_j - |\mathcal{C}_j[+1]|$ is equal to

$$S_j - (t_c - \mathcal{C}_j[-1]) = S_j + \mathcal{C}_j[-1] - t_c = t_r + 1 - t_c = \delta_a.$$

  Interpreted in another way, this value gives the number of additional calls the reduction makes for each nonce (R_j, R'_j) besides the ones for corruptions. This calculation is true for all indices $j \in [\ell q_r]$ in case $\theta = 0$ (i.e., the forgery nonce R^* is not among previously queried signature shares for some nonce pair $(R_\vartheta, R'_\vartheta)$). In case $\theta = 1$, however, this calculation is true for all $j \neq \vartheta$. For $j = \vartheta$, this number is equal to $\delta_a - 1$, since A makes at most t_r signing queries for the nonce R^* (which is derived from $(R_\vartheta, R'_\vartheta)$) and a message m). By summing up over all indices $j \in [\ell q_r]$, we obtain for the total number of discrete logarithm oracle calls of this type $S := \ell q_r \delta_a - \theta$.

- As a result, the total number of calls the reduction makes to its discrete logarithm oracle, including the queries from the oracle-aided simulators, is given by the following sum:

$$(k_0 - \delta_a) + 2q_r \cdot (k_1 - \ell\delta_a) + (\ell q_r \delta_a - \theta) = k - \delta_a - q_r \ell \delta_a - \theta.$$

  The first summand is the number of calls the initial simulator Sim_0 makes, the second summand is the number of calls the Nonce-ADKG simulators $\mathsf{Sim}_i, \mathsf{Sim}'_i$ for $i \in [q_r]$ make, and the third

[18] Effectively, this should reveal at least $t_r + 1$ signature shares for each such index j.

summand is the number of calls the game additionally makes for the signing queries (refer to the preceding discussion).

- Finally, when A outputs its forgery, we let the reduction additionally call its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input pk_i for all $i \in \mathcal{H}$ such that $\mathcal{S}_i[\vartheta] = 1$ (in the other case where $\theta = 0$, the reduction randomly chooses a set of indices $i \in \mathcal{H}$ of size $t_r - |\mathcal{C}|$). We denote this set of indices by I_*. Since we assume that A makes t_r signature share queries for R^*, this number is given by $t_r - |\mathcal{C}| = t_r - t_c = \delta_a - 1$. Further, we let the reduction additionally call its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,i}$ for all $j \in [\ell q_r]$ and $i \in I_*$ (note that these are $\delta_a - 1$ additional calls for each j). In case $\theta = 1$, we let the reduction make an additional discrete logarithm oracle call on input R_ϑ. Adding this up with the computed number in the previous bullet point, we thus obtain

$$(k - \delta_a - q_r\ell\delta_a - \theta) + (\delta_a - 1) + q_r\ell \cdot (\delta_a - 1) + \theta = k - 1 - q_r\ell$$

  as the total number of calls the reduction has made so far to the discrete logarithm oracle provided by the OMDL challenge of degree k.

Now that we have counted queries, let us continue with a look at the bigger picture of the proof. Informally, what we have done so far is the following. During the game, we have obtained the secret key shares $\{\mathsf{sk}_i\}_{i\in\mathcal{C}}$ and secret nonce shares $\{r_{j,i}, r'_{j,i} \mid j \in [\ell q_r]\}_{i\in\mathcal{C}}$ of the t_c corrupt parties $\mathcal{C} \subset [n]$. We have also obtained $t_r + 1$ signature shares $\{\sigma_{j,i} \mid j \in [\ell q_r],\ j \neq \vartheta\}_{i\in[n]}$ (by Lagrange interpolation we obtain all n shares) where $\vartheta \in [\ell q_r] \cup \{\bot\}$ denotes the index used for the forgery nonce (recall that $\vartheta = \bot$ in case $\theta = 0$, and $\vartheta \in [\ell q_r]$ otherwise). Finally, we have obtained the additional secret key shares $\{\mathsf{sk}_i\}_{i\in I_*}$ (thus in total $|\mathcal{C} \cup I_*| = t_r$ secret key shares), the secret nonce shares $\{r_{j,i} \mid j \in [\ell q_r],\ j \neq \vartheta\}_{i\in I_*}$ (thus in total $|\mathcal{C} \cup I_*| = t_r$ secret nonce shares $\{r_{j,i}\}$ for each $j \in [\ell q_r] \setminus \{\vartheta\}$), and for the case $\theta = 1$ also the secret nonce r_ϑ. In any case, the reduction can obtain from this data the additional secret nonce shares $\{r'_{j,i} \mid j \in [\ell q_r],\ j \neq \vartheta\}_{i\in I_*}$. This can be seen as follows. For $i \in I_*$ and $j \in [\ell q_r] \setminus \{\vartheta\}$, it can from the knowledge of sk_i and $\sigma_{j,i}$ derive the value for $\hat{r}_{j,i}$ and together with $r_{j,i}$ thus obtain $r'_{j,i}$. At this point, we note that the reduction still has $q_r\ell$ discrete logarithm oracle calls at its disposal. Later, the reduction will use these calls to compute all but one value from among the set $\{\mathsf{sk}, r_j\}_{j\in[\ell q_r]}$ (via a case distinction on the algebraic equation obtained from the adversary's random oracle query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$) and derive the remaining value from the adversary's forgery signature. As described before, it can then also derive the value r'_j. Invertibility of the simulatability matrices of the oracle-aided simulators then allows the reduction to obtain a solution to the OMDL challenge ξ. Having conveyed the intuition, we now make the final part of our proof formal.

Converting forgery to solution of OMDL. In the final part of our proof, we show how the reduction can efficiently convert the forgery produced by A into a solution of the underlying OMDL challenge ξ. We give a brief intuition for this part. First, since A is an algebraic adversary, it outputs the random oracle query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$ together with an algebraic representation of elements in $\mathbb{Z}_p$. Second, using the forgery (m^*, σ^*), the remaining signature shares $\{\sigma_{j,1}, \ldots, \sigma_{j,n}\}_{j=1}^{\ell q_r}$, and the secret key shares sk_i for $i \in \mathcal{C} \cup I_*$, we can solve for the secret key sk. Third, this enables us to compute all secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ and thus by going back to the signature shares also all effective secret nonce shares $\{\hat{r}_{j,1}, \ldots, \hat{r}_{j,n}\}_{j=1}^{\ell q_r}$ from which we can then derive the secret nonce shares $\{r_{j,1}, r'_{j,1}, \ldots, r_{j,n}, r'_{j,n}\}_{j=1}^{\ell q_r}$. Finally, by inverting the simulatability matrices of the oracle-aided simulators $\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}'_1, \ldots, \mathsf{Sim}_{q_r}, \mathsf{Sim}'_{q_r}$, we can use the aforementioned values to obtain a solution to the OMDL challenge ξ. This ends our informal discussion on the proof strategy for converting the forgery into a solution to OMDL. We proceed with the actual analysis now.

Hereafter, let $\mathcal{C} \subset [n]$ and $\mathcal{H} = [n] \setminus \mathcal{C}$ denote the set of corrupt and honest parties, respectively, right before A outputs its forgery (m^*, R^*, s^*). Let $I_* \subset \mathcal{H}$ denote the set of parties for which the reduction made a call to its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input pk_i after obtaining the forgery from A. This set was already defined in the preceding paragraph. In the following, let $L \in [\![\ell q_r]\!]$ denote the index of the nonce after whose generation A queried the random oracle on the triple (pk, R^*, m^*). Since A is an algebraic adversary, when it makes the query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$, it also outputs an algebraic representation

$$\mathbf{a} := \big(\hat{a}, a', a_1, \ldots, a_n, \{a_{1,1}, a'_{1,1}, \ldots, a_{1,n}, a'_{1,n}\}, \ldots, \{a_{L,1}, a'_{L,1}, \ldots, a_{L,n}, a'_{L,n}\}\big)$$

of elements in $\mathbb{Z}_p$ such that

$$R^* = g^{\hat{a}} \cdot \mathsf{pk}^{a'} \cdot \mathsf{pk}_1^{a_1} \cdot \ldots \cdot \mathsf{pk}_n^{a_n} \cdot \prod_{j=1}^{L} R_{j,1}^{a_{j,1}} R'^{a'_{j,1}}_{j,1} \cdot \ldots \cdot R_{j,n}^{a_{j,n}} R'^{a'_{j,n}}_{j,n}.$$

Here, the representation is split from left to right into powers of the generator g, the public key pk, the public key shares $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and the public nonce shares $\{R_{j,i}, R'_{j,i}\}_{i\in[n]}$ for $j \in [L]$ returned by the Nonce-ADKG oracle. We omit the combined nonces $\{R_j, R'_j\}_{j\in[L]}$ here, since these can be written as known linear combinations (Lagrange interpolation in the exponent) of the respective shares $\{R_{j,1}, \ldots, R_{j,n}\}$ (analogously for R'_j). Verification of the forgery (m^*, R^*, s^*) implies $R^* = g^{s^*} \cdot \mathsf{pk}^{-c^*}$ where $c^* = \mathsf{H}(\mathsf{pk}, R^*, m^*)$. Further, recall that $R_{j,i} = g^{r_{j,i}}$ and $R'_{j,i} = g^{r'_{j,i}}$ for all $j \in [L]$ and $i \in [n]$. Recall by assumption (ii) that the reductions knows the entire set of signature shares $\{\sigma_{j,i} \mid j \in [L],\ i \in [n]\}$ and that they satisfy $\hat{R}_{j,i} = g^{\sigma_{j,i}} \cdot \mathsf{pk}_i^{-c_j}$. Here, we have $c_j := \mathsf{H}(\mathsf{pk}, \hat{R}_j, m_j)$ and $\hat{R}_j := R_j R'^{b_j}_j$ where $b_j := \mathsf{H}_{\mathrm{non}}(\mathsf{pk}, R_j, R'_j, m_j)$ and m_j is the message bound to the nonce pair (R_j, R'_j). Having said that, the above equation is over $\mathbb{Z}_p$ equivalent to

$$s^* - c^* \cdot \mathsf{sk} = \hat{a} + a' \cdot \mathsf{sk} + \sum_{i=1}^{n} a_i \cdot \mathsf{sk}_i + \sum_{j=1}^{L}\sum_{i=1}^{n} a_{j,i} r_{j,i} + a'_{j,i} b_j^{-1}(\sigma_{j,i} - c_j \cdot \mathsf{sk}_i - r_{j,i}). \qquad (\spadesuit)$$

By construction, when A outputs its forgery, the reduction calls its discrete logarithm oracle on input pk_i for all $i \in I_*$ and thus knows sk_i for all $i \in \mathcal{C} \cup I_*$. Without loss of generality we assume that $\mathcal{C} \cup I_* = \{1, \ldots, t_r\}$. Since the secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ interpolate a polynomial $f \in \mathbb{Z}_p[X]$ of degree $t_r + 1$, there are coefficients $\alpha, \alpha' \in \mathbb{Z}_p$ (depending on the scalars $a_1, \ldots, a_n$ and $\mathsf{sk}_1, \ldots, \mathsf{sk}_{t_r}$) such that $\sum_{i\in[n]} a_i \cdot \mathsf{sk}_i = \alpha \cdot \mathsf{sk} + \alpha'$. These coefficients α, α' can efficiently be obtained from Lagrange interpolation operations using knowledge of $\mathsf{sk}_1, \ldots, \mathsf{sk}_{t_r}$. Similarly, for every $j \in [L]$ there are coefficients $\alpha_j, \alpha'_j \in \mathbb{Z}_p$ (depending on the scalars $a_{j,1}, \ldots, a_{j,n}$ and $\mathsf{sk}_1, \ldots, \mathsf{sk}_{t_r}$) such that $\sum_{i\in[n]} a'_{j,i} \cdot \mathsf{sk}_i = \alpha_j \cdot \mathsf{sk} + \alpha'_j$. As the known index set for the secret key shares $\{\mathsf{sk}_i\}_i$ and the secret nonce shares $\{r_{j,i}\}_i$ is the same t_r-sized set $\mathcal{C} \cup I_*$, the interpolation coefficient to obtain $\mathsf{sk}_0 = \mathsf{sk}$ (from the remaining shares $\{\mathsf{sk}_i\}_{i\in\mathcal{H}_*}$ where $\mathcal{H}_* := \mathcal{H} \setminus I_*$ is the complement of $\mathcal{C} \cup I_*$) and $r_j = r_{j,0}$ (from the remaining shares $\{r_{j,i}\}_{i\in\mathcal{H}_*}$), we similarly have the identity $\sum_{i\in[n]} a'_{j,i} \cdot r_{j,i} = \alpha_j \cdot r_j + \beta_j$ for some known scalars $\beta_j \in \mathbb{Z}_p$. In the above equation $(\spadesuit)$, we put together the known values for $\{s^*, \hat{a}, \alpha', \sigma_{j,i}, \alpha'_j, \beta_j, r_{j,i}, \mathsf{sk}_i\}$ (where the set ranges over $j \in [L]$ and $i \in \mathcal{C} \cup I_*$) and thus obtain

$$\tilde{s} - c^* \cdot \mathsf{sk} = \tilde{a} \cdot \mathsf{sk} - \sum_{j\in[L]} c_j \alpha_j b_j^{-1} \cdot \mathsf{sk} + \sum_{j\in[L]} (\tilde{a}_j - \alpha_j b_j^{-1}) \cdot r_j, \qquad (\diamondsuit)$$

where $\tilde{s}$, $\tilde{a}$, and $\tilde{a}_j$ are appropriately defined known values. We make the following observation. As A queries the random oracle on R^* before obtaining the value c^*, it fixes the algebraic coefficients $\mathbf{a}$ before c^* is chosen (uniformly at random) by the reduction. The same is true for all (c_j, b_j), $j \in [L]$, values that were already fixed at the time A made the random oracle query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$; we denote the corresponding index set by $\mathcal{L}_{\mathrm{pre}} \subseteq \mathcal{L} := [L]$. However, this does not apply to the pairs (c_j, b_j) that were only defined after c^* was chosen; we denote the corresponding index set by $\mathcal{L}_{\mathrm{post}} \subseteq \mathcal{L}$ (note that $[L] = \mathcal{L}_{\mathrm{pre}} \cup \mathcal{L}_{\mathrm{post}}$). Having said that, we define the following event ROS by: there exists an $j \in \mathcal{L}_{\mathrm{post}}$ such that $\alpha_j \neq 0$. The reduction proceeds as follows with the goal to compute all the secret shares $\mathsf{sk}_i, r_{j,i}, r'_{j,i}$ for all $j \in [\ell q_r]$ and all $i \in [n]$. We make a case distinction with two cases.

Case I: ROS *occurs.* In this case, there is an $\hat{j} \in \mathcal{L}_{\mathrm{post}}$ such that $\alpha_{\hat{j}} \neq 0$. Now we let the reduction call its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input R_j for all $j \in [\ell q_r] \setminus \{\hat{j}\}$ and additionally on input pk. As a result, the reduction obtains the secret key sk and the secret nonces $\{r_j\}_{j\neq\hat{j}}$. Therefore, the above equation $(\diamondsuit)$ reduces to $(\tilde{a}_{\hat{j}} - \alpha_{\hat{j}} b_{\hat{j}}^{-1}) \cdot r_{\hat{j}} = (\ldots)$ where the right-hand side is some now known value. By assumption of $\hat{j} \in \mathcal{L}_{\mathrm{post}}$, we know that the value for $b_{\hat{j}}$ was chosen by the reduction uniformly at random and after the values for $\tilde{a}_{\hat{j}}, \alpha_{\hat{j}}$ were fixed by the adversary (note that these values are derived from the algebraic coefficients $\mathbf{a}$ the adversary fixed at the time of its random oracle query $\mathsf{H}(\mathsf{pk}, R^*, m^*)$, the secret key shares sk_i for $i \in \mathcal{C} \cup I_*$, the secret nonce shares $r_{j,i}$ for $j \in [L]$ and $i \in \mathcal{C} \cup I_*$, and fixed Lagrange interpolation coefficients). As a result, the expression $\tilde{a}_{\hat{j}} - \alpha_{\hat{j}} b_{\hat{j}}^{-1}$ is only zero with negligible probability $1/p$. In particular, we find that $r_{\hat{j}} = (\dots) \cdot (\tilde{a}_{\hat{j}} - \alpha_{\hat{j}} b_{\hat{j}}^{-1})^{-1}$ is well-defined with probability $1 - 1/p$. By knowledge of secret key shares $\{\mathsf{sk}_i\}_{i \in [n]}$, signature shares $\{\sigma_{j,i}\}$, and additionally all secret nonce shares $\{r_j\}_{j \in [\ell q_r]}$, the reduction can efficiently compute all secret nonce shares $\{r'_j\}_{j \in [\ell q_r]}$. From this data it can then also compute all the secret nonce shares $\{r_{j,i}, r'_{j,i}\}_{j,i}$. Next, we consider the case where the event ROS does not occur.

Case II: ROS *does not occur.* In this case, for all $j \in \mathcal{L}_{\text{post}}$ we have $\alpha_j = 0$. Then, the above equation ($\diamondsuit$) reduces to

$$\tilde{s} + \left(\sum_{j \in \mathcal{L}_{\text{pre}}} c_j \alpha_j b_j^{-1} - \tilde{a} - c^* \right) \cdot \mathsf{sk} = \sum_{j \in [L]} \left(\tilde{a}_j - \alpha_j b_j^{-1} \right) \cdot r_j. \tag{$\heartsuit$}$$

Similarly, we observe that the sum $\sum_{j \in \mathcal{L}_{\text{pre}}} c_j \alpha_j b_j^{-1} - \tilde{a}$ on the left-hand side of the above equation ($\heartsuit$) only contains values that were fixed by the adversary before the reduction chose c^* uniformly at random. As a consequence, the expression $\sum_{j \in \mathcal{L}_{\text{pre}}} c_j \alpha_j b_j^{-1} - \tilde{a} - c^*$ is only zero with negligible probability $1/p$. In particular, we find from equation ($\heartsuit$) that

$$\mathsf{sk} = \left(\sum_{j \in [L]} \left(\tilde{a}_j - \alpha_j b_j^{-1} \right) \cdot r_j - \tilde{s} \right) \cdot \left(\sum_{j \in \mathcal{L}_{\text{pre}}} c_j \alpha_j b_j^{-1} - \tilde{a} - c^* \right)^{-1}$$

is well-defined with probability $1 - 1/p$. Now, we let the reduction call its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input R_j for all $j \in [\ell q_r]$ so that it derives sk from the preceding identity. As before, by knowledge of secret key shares $\{\mathsf{sk}_i\}_{i \in [n]}$, signature shares $\{\sigma_{j,i}\}$, and secret nonce shares $\{r_j\}_{j \in [\ell q_r]}$, it can efficiently compute all secret nonce shares $\{r'_j\}_{j \in [\ell q_r]}$. Finally, from this data it can then compute all the secret nonce shares $\{r_{j,i}, r'_{j,i}\}_{j,i}$.

Final Step. What we have achieved so far is that the reduction has obtained from the forgery all the secret shares $\mathsf{sk}_i, r_{j,i}, r'_{j,i}$ for all $j \in [\ell q_r]$ and all $i \in [n]$. From this, we want to derive the solution to the given OMDL challenge ξ using properties of the oracle-aided simulators $\{\mathsf{Sim}_0, \mathsf{Sim}_j, \mathsf{Sim}'_j\}_{j \in [q_r]}$. In the following, denote by $g_1, \dots, g_{k_0 - \delta_a} \in \mathbb{G}$ the queries the initial simulator Sim_0 makes to the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ and let $(\hat{a}_i, a_{i,1}, \dots, a_{i,k_0})$ for $i \in [k_0 - \delta_a]$ be g_i's corresponding algebraic vector. Further, for $i \in [q_r]$ denote by $g_{i,1}, \dots, g_{i,k_1 - \ell\delta_a} \in \mathbb{G}$ the queries the simulator Sim_i makes to $\mathsf{DL}_{\mathbb{G},g}$ and let $(\hat{a}_{\mathbf{i}_j}, a_{\mathbf{i}_j,1}, \dots, a_{\mathbf{i}_j,k_1})$ for $\mathbf{i}_j := (i,j) \in [q_r] \times [k_1 - \ell\delta_a]$ be $g_{\mathbf{i}_j}$ its corresponding algebraic vector, i.e.,

$$g_{\mathbf{i}_j} = g^{\hat{a}_{\mathbf{i}_j}} \cdot \xi_{i,1}^{a_{\mathbf{i}_j,1}} \cdot \ldots \cdot \xi_{i,k_1}^{a_{\mathbf{i}_j,k_1}}.$$

For $i \in [q_r]$ denote by $g'_{i,1}, \dots, g'_{i,k_1 - \ell\delta_a} \in \mathbb{G}$ the queries the simulator Sim'_i makes to $\mathsf{DL}_{\mathbb{G},g}$ and let $(\hat{a}'_{\mathbf{i}_j}, a'_{\mathbf{i}_j,1}, \dots, a'_{\mathbf{i}_j,k_1})$ for $\mathbf{i}_j := (i,j) \in [q_r] \times [k_1 - \ell\delta_a]$ be $g'_{\mathbf{i}_j}$ its corresponding algebraic vector, i.e.,

$$g'_{\mathbf{i}_j} = g^{\hat{a}'_{\mathbf{i}_j}} \cdot \xi'^{a'_{\mathbf{i}_j,1}}_{i,1} \cdot \ldots \cdot \xi'^{a'_{\mathbf{i}_j,k_1}}_{i,k_1}.$$

Now let $\mathcal{I}_0 \subset \mathcal{H}$ be any set of size δ_a, and for $i \in [q_r]$ let $\mathcal{I}_i := I_i^\ell \subset \mathcal{H}^\ell$ be any set for which $|I_i| = \delta_a$. Further, let $(\hat{a}_i, a_{i,1}, \dots, a_{i,k_0})$ for $i \in [k_0 - \delta_a + 1, k_0]$ be the algebraic vectors of the elements in $\{\mathsf{pk}_i\}_{i \in \mathcal{I}_0}$ (in some fixed order), and for $i \in [q_r]$ let $(\hat{a}_{\mathbf{i}_j}, a_{\mathbf{i}_j,1}, \dots, a_{\mathbf{i}_j,k_1})$ with $\mathbf{i}_j := (i,j) \in [q_r] \times [k_1 - \ell\delta_a + 1, k_1]$ be the algebraic vectors of the elements in $\{R_{i,j}\}_{j \in \mathcal{I}_i}$ (in some fixed order). Further, for $i \in [q_r]$ let $(\hat{a}'_{\mathbf{i}_j}, a'_{\mathbf{i}_j,1}, \dots, a'_{\mathbf{i}_j,k_1})$ with $\mathbf{i}_j := (i,j) \in [q_r] \times [k_1 - \ell\delta_a + 1, k_1]$ be the algebraic vectors of the elements in $\{R'_{i,j}\}_{j \in \mathcal{I}_i}$ (in some fixed order). Essentially, this means that we consider the δ_a public key shares $\{\mathsf{pk}_i\}_{i \in \mathcal{I}_0}$ and their algebraic representation, and for each Nonce-ADKG index $i \in [q_r]$ we consider $\ell\delta_a$ public nonce shares $\{R_{\ell i+1,j}, \dots, R_{\ell i+\ell,j}\}_j$ (respectively the public nonce shares $\{R'_{\ell i+1,j}, \dots, R'_{\ell i+\ell,j}\}_j$) and their algebraic representation where j ranges over the set I_i (for each of the ℓ public nonces $\{R_{\ell i+k}\}_{k \in [\ell]}$ that the i-th Nonce-ADKG generates in the first instance, we consider the δ_a public nonce shares with indices ranging over the set I_i). Collectively, this gives k_0 elements with known discrete logarithm values $\{v_i\}_{i \in [k_0]}$ associated to Sim_0, k_1 elements with known discrete logarithm values $\{v_{i,j}\}_{j \in [k_1]}$ associated to Sim_i for all $i \in [q_r]$, and k_1 elements with known discrete logarithm values $\{v'_{i,j}\}_{j \in [k_1]}$ associated to Sim'_i for all $i \in [q_r]$. We denote the set of these elements (with known value) by $\mathcal{V} := \{v_l, v_{i,j}, v'_{i,j} \mid l \in [k_0], i \in [q_r], j \in [k_1]\}$

which is of size $k := k_0 + 2q_r k_1$. Note that the secret key shares $\{\mathsf{sk}_i\}_i$ for $i \in \mathcal{C} \cup \mathcal{I}_0$ and the secret nonce shares $\{r_{i,j}, r'_{i,j}\}_{i,j}$ for $i \in [q_r]$ and $j \in (\mathcal{C} \cup I_i)^\ell$ (recall that $\mathcal{I}_i = I_i^\ell \subset \mathcal{H}^\ell$ and $|I_i| = \delta_a$, and so the set $\mathcal{C} \cup I_i$ is of size $t_c + \delta_a = t_r + 1$) are among the set of known discrete logarithm values and thus contained in $\mathcal{V}$. In more detail, we have $v_i = \mathsf{sk}_i$ for $i \in \mathcal{C} \cup I_0$, and $v_{i,j} = r_{i,j}$, $v'_{i,j} = r'_{i,j}$ for $i \in [q_r]$ and $j \in (\mathcal{C} \cup I_i)^\ell$. We denote by $L_i := L(\mathcal{I}_i, \mathcal{C})$ for $i \in [\![q_r]\!]$ the simulatability matrix of Sim_i with respect to the set $\mathcal{I}_i$. Similarly, we denote by $L'_i := L(\mathcal{I}_i, \mathcal{C})$ for $i \in [q_r]$ the simulatability matrix of Sim'_i with respect to the set $\mathcal{I}_i$. Recall for all $i \in [k_0 - \delta_a]$ the identities

$$g_i = g^{\hat{a}_i} \cdot \xi_1^{a_{i,1}} \cdot \ldots \cdot \xi_{k_0}^{a_{i,k_0}} \quad \Longleftrightarrow \quad g^{v_i} \cdot g^{-\hat{a}_i} = \xi_1^{a_{i,1}} \cdot \ldots \cdot \xi_{k_0}^{a_{i,k_0}}$$

from the initial ADKG execution, where the $g_i = g^{v_i} \in \mathbb{G}$ are the queries the simulator Sim_0 makes to the discrete logarithm oracle. Further, for all $i \in [k_0 - \delta_a + 1, k_0]$ we have the identities

$$\mathsf{pk}_i = g^{\hat{a}_i} \cdot \xi_1^{a_{i,1}} \cdot \ldots \cdot \xi_{k_0}^{a_{i,k_0}} \quad \Longleftrightarrow \quad g^{v_i} \cdot g^{-\hat{a}_i} = \xi_1^{a_{i,1}} \cdot \ldots \cdot \xi_{k_0}^{a_{i,k_0}}.$$

From these equations and by definition of the matrix L_0, the identity holds:

$$L_0 \cdot \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{k_0} \end{pmatrix} = \begin{pmatrix} v_1 - \hat{a}_1 \\ v_2 - \hat{a}_2 \\ \vdots \\ v_{k_0} - \hat{a}_{k_0} \end{pmatrix},$$

where z_i denotes the discrete logarithm value of the i-th element ξ_i of the OMDL challenge $\xi \in \mathbb{G}^k$. Analogous considerations for the Nonce-ADKG executions yield the identities

$$L_i \cdot \begin{pmatrix} z_{i,1} \\ z_{i,2} \\ \vdots \\ z_{i,k_1} \end{pmatrix} = \begin{pmatrix} v_{i,1} - \hat{a}_{i,1} \\ v_{i,2} - \hat{a}_{i,2} \\ \vdots \\ v_{i,k_1} - \hat{a}_{i,k_1} \end{pmatrix}, \quad L'_i \cdot \begin{pmatrix} z'_{i,1} \\ z'_{i,2} \\ \vdots \\ z'_{i,k_1} \end{pmatrix} = \begin{pmatrix} v'_{i,1} - \hat{a}'_{i,1} \\ v'_{i,2} - \hat{a}'_{i,2} \\ \vdots \\ v'_{i,k_1} - \hat{a}'_{i,k_1} \end{pmatrix}$$

for all $i \in [q_r]$. As before, we denote by $z_{i,j}$ the discrete logarithm value of the OMDL element $\xi_{i,j}$ and by $z'_{i,j}$ the discrete logarithm value of $\xi'_{i,j}$ for all $i \in [q_r]$ and $j \in [k_1]$. By definition of oracle-aided simulatability, the matrices L_0, L_i and L'_i for $i \in [q_r]$ are invertible and explicitly known (defined by the algebraic vectors provided by Sim_0, Sim_i and Sim'_i). Moreover, the vectors on the right-hand side contain explicitly known values (from knowledge of the set $\mathcal{V}$ and the algebraic vectors provided by the simulators). As a result, by inversion of the simulatability matrices L_0, L_i and L'_i for $i \in [q_r]$, we can efficiently compute $(z_1, \ldots, z_{k_0+2q_r k_1})$ and thus solve the OMDL challenge ξ of degree $k = k_0 + 2q_r k_1$ (with at most $k - 1$ calls to the discrete logarithm oracle). This concludes the proof. □

Remark C.1 We note that the above proof also applies to an algebraic version of the OMDL assumption as defined in [NRS21]. For this, we observe that our security reduction queries the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ only in the following two cases: (i) when an algebraic simulator Sim_i makes a discrete logarithm oracle query, and (ii) when it needs to compute a signature share $\sigma_{j,i}$, it calls the oracle on input $\Sigma_{j,i} := \mathsf{pk}_i^{c_j} \cdot \hat{R}_{j,i}$. By algebraicity of all simulators $\{\mathsf{Sim}_0, \mathsf{Sim}_i, \mathsf{Sim}'_i\}_{i \in [q_r]}$, these elements are linear combinations of the OMDL elements on which the simulators are run. Since $\hat{R}_{j,i}$ itself is also an algebraic linear combination of the nonces $R_{j,i}$ and $R'_{j,i}$, the reduction can provide (in an efficient manner) an algebraic representation for the elements $\Sigma_{j,i}$ when calling the oracle $\mathsf{DL}_{\mathbb{G},g}$. As a result, the reduction itself calls $\mathsf{DL}_{\mathbb{G},g}$ only on algebraic linear combinations of the OMDL challenge it is given.

C.2 Proof for Packed ADKG

Proof of Theorem 5.2. We show the properties of an oracle-aided secure packed ADKG protocol (cf. Definition 3.1). We begin with the termination property. To this end, we also recall the protocol PADKG and go through each of its phases. Hereafter, we use the term *party P_i multicasts a message m* to mean that P_i sends the message m to all parties in the system. Further, we use the term *(a,b)-dimensional matrix* to mean a matrix of dimension $a \times b$ specified over the field $\mathbb{Z}_p$. For the following discussion, we split the protocol PADKG into five phases in contrast to the four phases described in Section 5.1. Specifically, we split the second *MVBA execution* phase into a *gather proof* part and the actual MVBA execution part. This will make the argumentation easier to follow.

Termination. In the first phase of the protocol, each party P_i samples a secret element $s_i \leftarrow_{\$} \mathbb{Z}_p$ uniformly at random and shares it among all parties via an execution of the high-threshold AVSS scheme that we denote by AVSS_i for $i \in [n]$. Let $f_i \in \mathbb{Z}_p[X]$ denote the corresponding polynomial of degree t_r that interpolates the secret $f_i(0) = s_i$. More specifically, parties execute the Share protocol of $\mathsf{AVSS} = (\mathsf{Share}, \mathsf{Rec})$ at the beginning of this phase. By the correctness property of AVSS we know that if an instance AVSS_j of the AVSS scheme terminates at an honest party P_i, then it will eventually terminate at all honest parties. Since each honest party executes its AVSS instance correctly, we know that there are at least $n - t_c$ AVSS instances that will terminate at each honest party. Having said that, each party waits for $n - t_c$ AVSS instances to terminate at it locally. By the observation made previously, this phase will eventually terminate for all honest parties, as there are at least $n - t_c$ honest parties and these execute their AVSS instance correctly. For this, each party P_i locally maintains a set $\mathsf{dealers}_i$ that is initially empty but throughout keeps track of the dealers whose AVSS instance AVSS_j terminates at P_i.

Once this set has size $n - t_c$, the second phase for the parties begins. This is the typical gather proof phase in asynchronous consensus protocols. However, we note here that the set $\mathsf{dealers}_i$ is still maintained and being filled with completed AVSS instances even in subsequent phases so that it could even potentially reach a size of n (for instance in case each party behaves honestly and the whole network progresses fast with minimal delays). The goal of the second phase is to extend a message that is a priori not externally valid in such a way that it becomes externally valid. Here, the idea is to collect at least $t_c + 1$ signatures from other parties on its set of dealers to ensure there is at least one honest party that approves this set. As a consequence, parties can be sure that this set is a valid set of AVSS instances that will eventually terminate at each honest party. More concretely, each party P_i sends its set of dealers as a proposal to all parties. For this, it defines the proposal set $\mathsf{prop}_i := \mathsf{dealers}_i$ and multicasts this set. Upon receiving such a set prop_j from another party P_j, the party P_i waits until all AVSS instances declared in prop_j terminate at it locally, i.e., until $\mathsf{dealers}_i \supseteq \mathsf{prop}_j$ (note that the local set $\mathsf{dealers}_i$ keeps growing in general). Only after this condition is satisfied, party P_i knows that each of the instances $\mathsf{AVSS}_{j'}$ for $j' \in \mathsf{prop}_j$ will also eventually terminate at each other honest party and approves this set by providing a digital signature $\mathsf{Sig}(\mathsf{sik}_i, \mathsf{prop}_j)$ on it with its signing key sik_i and sending this signature back to party P_j. To this end, each party P_i also maintains a local set of signatures sigs_i that is initially empty but keeps track of exactly these signatures $\mathsf{Sig}(\mathsf{sik}_j, \mathsf{prop}_i)$ from other parties P_j that approve its proposal prop_i. Since we know that each AVSS instance AVSS_j with $j \in \mathsf{prop}_i$ that terminated at P_i, will eventually also terminate at any other honest party and there are at least $n - t_c \geq t_c + 1$ honest parties, P_i will eventually collect at least $t_c + 1$ signatures on its proposal and thus terminate the second phase. Once a party collects $t_c + 1$ signatures on its proposal (i.e., once $|\mathsf{sigs}_i| = t_c + 1$), it progresses to the next phase.

In the third phase of the protocol, parties execute an instance of the MVBA protocol MVBA whose goal is to agree on such a set prop of $n - t_c$ dealers along with the approval sigs that satisfies the external validity predicate (i.e., the set prop of dealers is of size $n - t_c$ and the approval sigs consists of at least $t_c + 1$ valid signatures on prop). From the previous phase we know that each honest party P_i will invoke MVBA with an externally valid input $(\mathsf{prop}_i, \mathsf{sigs}_i)$. As a consequence, by the termination property of the MVBA protocol, each honest party will terminate MVBA with the same externally valid output $(\mathsf{prop}, \mathsf{sigs})$ and progress to the next phase.

In the fourth phase of the protocol, parties execute the (exponentiated) reconstruction phase of the AVSS instance AVSS_j for all $j \in \mathsf{prop}$ for the agreed upon instances from the MVBA protocol. By the termination property of the MVBA protocol and the AVSS scheme, each honest party P_i will invoke the reconstruction protocol Rec of AVSS_j and terminate with the public output $(S_j, S_{j,1}, \ldots, S_{j,n})$ and private output $s_{j,i}$. In particular, each honest party will progress to the next phase. In the following, define $\ell := n - 2t_c$ (which is equal to $t_c + 1$ in the optimal-resilience case). In the fifth phase of the protocol, each party P_i locally applies the $(\ell, n - t_c)$-dimensional superinvertible matrix SI to its $(n - t_c)$-dimensional vector of private outputs to obtain an ℓ-dimensional vector $(x_{1,i}, \ldots, x_{\ell,i})$ of new private outputs. Additionally, P_i locally applies the superinvertible matrix SI in the exponent to the $(n - t_c, n)$-dimensional matrix consisting of rows $(S_{j,1}, \ldots, S_{j,n})$ for $j \in \mathsf{prop}$ to obtain an n-dimensional vector $(R_{j,1}, \ldots, R_{j,n})$ of new public outputs for each $j \in [\ell]$. Since all these operations are done locally and efficiently computable, each honest party will terminate this phase. Additionally, by the correctness property of the AVSS scheme and the consistency property of the MVBA protocol, the honest parties have agreement on the public output data which is the resulting n-dimensional vectors $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$. Finally, each party terminates the whole protocol with public output $(R_j, R_{j,1}, \ldots, R_{j,n})$ for $j \in [\ell]$ and private output $(x_{1,i}, \ldots, x_{\ell,i})$ where R_j is also obtained from the $R_{j,(\cdot)}$ by Lagrange interpolation in the exponent. Here, the elements $(R_j, R_{j,1}, \ldots, R_{j,n})$ are the public nonce shares in the j-th slot of the packed ADKG protocol with corresponding secret nonce share $x_{j,i}$ for party P_i. This concludes the discussion on termination of the protocol.

Correctness and Consistency. We proceed with the consistency and correctness properties of the protocol. We do not consider these properties separately, since the correctness property will immediately follow from our discussion on consistency. First of all, we show that the public output data $(R_j, R_{j,1}, \ldots, R_{j,n})$ for each slot $j \in [\ell]$ corresponds to a polynomial $r_j \in \mathbb{Z}_p[X]$ of degree at most t_r. By the correctness property of the AVSS scheme, we know that each instance AVSS_i for the agreed upon set of instances $i \in \mathsf{prop}$ output by MVBA corresponds to a polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t_r; hereafter, we take w.l.o.g. $\mathsf{prop} = \{1, \ldots, n - t_c\}$. In particular, this property is preserved after the application of the superinvertible matrix SI. The reason for this is that SI considered as an $(n - t_c, \ell)$-dimensional matrix over the polynomial ring $\mathbb{Z}_p[X]$ (via the natural embedding $\mathbb{Z}_p \hookrightarrow \mathbb{Z}_p[X]$) is just a linear transformation that does

$$(f_1, \ldots, f_{n-t_c}) \mapsto (r_1, \ldots, r_\ell), \quad \forall i \in [\ell] : r_i := m_{i,1} f_1 + \ldots + m_{i,n-t_c} f_{n-t_c}$$

where $\mathsf{SI} = (m_{i,j})_{i,j}$. As a result, the n-dimensional vector $(R_{j,1}, \ldots, R_{j,n})$ for each $j \in [\ell]$ corresponds to a polynomial $r_j \in \mathbb{Z}_p[X]$ of degree at most t_r. Additionally, there is agreement on these elements from the correctness properties of MVBA and AVSS. Additionally, by correctness of the AVSS scheme we know that $R_{j,i} = g^{r_j(i)}$ for all $(j, i) \in [\ell] \times [n]$. By Lagrange interpolation, a party reconstructs the full vector of elements $(R_j, R_{j,1}, \ldots, R_{j,n})$. In particular, each honest party outputs the same public nonce R_j and the same vector of public nonce shares $(R_{j,1}, \ldots, R_{j,n})$ for each $j \in [\ell]$, and these elements come from polynomials $r_j \in \mathbb{Z}_p[X]$ of degree at most t_r. This concludes the discussion on correctness and consistency of the protocol.

Oracle-aided Simulatability. We proceed with the oracle-aided simulatability property of the protocol. Here, we have to be careful with how we choose the parameter $\tilde{k}$ for the packed ADKG protocol and how we use the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ in order to not exceed the allowed number of queries to it. However, the basic idea of the simulator Sim for the packed ADKG protocol is quite simple. In essence, Sim is build upon the individual oracle-aided simulators Sim_i for each AVSS instance AVSS_i with P_i as dealer and corruptions are handled by each Sim_i individually. By carefully choosing when to query the discrete logarithm oracle and other subtle details, we can design Sim in such a way that it satisfies all the properties the oracle-aided simulatability property requires, including the maximal allowed number of queries to the oracle $\mathsf{DL}_{\mathbb{G},g}$.

In the following, we provide a full and detailed description. Let AVSS have oracle-aided simulatability as described in Definition 5.1. In particular, there exists a natural number $k \in \mathsf{poly}(\lambda)$ with $k \geq t_r + 1$ and an algebraic simulator that on input k group elements with oracle access simulates the role of the honest parties in an execution of the AVSS instance and with additional specifications. For each $i \in [n]$, let Sim_i be the algebraic simulator that simulates the instance AVSS_i with P_i as dealer. We build an oracle-aided simulator Sim for the packed ADKG protocol denoted by PADKG as follows. First, define the simulatability factor of PADKG to be $\tilde{k} := kn \in \mathsf{poly}(\lambda)$. In particular, the condition $\tilde{k} \geq \ell(t_r + 1)$ is satisfied. Next, let $\xi := (\xi_1, \ldots, \xi_n) \in \mathbb{G}^{kn}$ be a tuple of kn group elements where $\xi_i = (g^{z_{i,1}}, \ldots, g^{z_{i,k}}) \in \mathbb{G}^k$ for each $i \in [n]$. We choose to write the input element ξ in this way for reasons that will be clear in a second. The element $\xi \in \mathbb{G}^{kn}$ is the one that Sim gets as input. Additionally, Sim gets access to a discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ in the group $\mathbb{G}$ (to base g)[19]. To have a clear understanding of how the simulator works, we divide the protocol PADKG into two conceptual phases. The first phase begins with the invocation of the protocol and ends with the reconstruction of the elements $(S_{j,1}, \ldots, S_{j,n})$ for all AVSS instances $j \in \mathsf{prop}$. The second phase begins with the application of the superinvertible matrix SI to public and private output data and ends after these local operations with public output data $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ and private output data.

[19] In a security proof, these elements along with the discrete logarithm oracle are instantiated with the OMDL assumption of degree kn.

We start with the description of the simulator in the first phase of the protocol. Let A be the PPT adversary that corrupts at most t_c parties. Further, let $\mathcal{C} \subset [n]$ and $\mathcal{H} := [n] \setminus \mathcal{C}$ denote the dynamically evolving sets of corrupted and honest parties, respectively. For each $i \in [n]$, the simulator Sim invokes Sim_i on input $\xi_i \in \mathbb{G}^k$ to handle the instance AVSS_i with party P_i as dealer. Whenever A decides to corrupt a party P_j with $j \in \mathcal{H}$, Sim forwards that corruption query to Sim_i for all $i \in [n]$. To simulate the discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ for Sim_i on an element $g' \in \mathbb{G}$ (with algebraic representation), the simulator Sim simply queries its own discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on this element g' and returns the result to Sim_i. The simulator Sim works that way throughout the first phase. For the execution of the MVBA protocol MVBA, the simulator Sim behaves on behalf of the honest parties correctly and according to the protocol instructions. Upon termination of the MVBA protocol the set prop of $n - t_c$ AVSS instances to be used is fixed. Following this, parties start the exponentiated reconstruction phase of these AVSS instances and terminate the first conceptual phase of the protocol with the public elements $(S_{j,1}, \ldots, S_{j,n})$ for all $j \in \mathsf{prop}$ along with some private output data. In the second conceptual phase of the protocol, the simulator Sim applies the superinvertible matrix SI to the n-dimensional public vectors $(S_{j,1}, \ldots, S_{j,n})$ for all $j \in \mathsf{prop}$ to obtain the vectors $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ and terminate the second conceptual phase. Since all involved algorithms are algebraic and the application of the superinvertible matrix is a linear operation, Sim can provide its output element with a corresponding algebraic representation. Having done this, Sim can terminate the protocol simulation.

Throughout the simulation up until the point in which the first corrupt party outputs the elements $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ (by design of the protocol, we can always assume that all slots $j \in [\ell]$ of the packed ADKG protocol output simultaneously), the simulator Sim delegates corruption queries entirely to the individual simulators Sim_i for all $i \in [n]$ without interfering. However, once the aforementioned event (i.e., the first corrupt party outputs $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$) happens and a party P_i with $i \in \mathcal{H}$ gets corrupted, the simulator proceeds as follows: it queries its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,i}$ for all $j \in [\ell]$ and then forwards this corruption query to all simulators $\mathsf{Sim}_{i'}$ with $i' \in [n]$. Discrete logarithm oracle queries from simulator $\mathsf{Sim}_{i'}$ on any input element $g' \neq S_{i',i}$ are answered as usual (querying its own oracle $\mathsf{DL}_{\mathbb{G},g}$). However, queries on the elements $S_{i',i}$ for all $i' \in [n]$ are answered as follows[20]. By definition of $R_{j,i}$, we know that

$$r_{j,i} = m_{j,1}s_{1,i} + \ldots + m_{j,n-t_c}s_{n-t_c,i} \quad \forall j \in [\ell], \tag{1}$$

where $\mathsf{SI} = (m_{j,i})_{j,i}$. Recall that the indices $1, \ldots, n - t_c$ for $s_{(\cdot),i}$ are the labels for parties in $\mathsf{dealers}$ (we assumed w.l.o.g. that $\mathsf{dealers} = \{1, \ldots, n - t_c\}$). Let $\mathcal{C}' := \mathcal{C} \cap \{\mathsf{dealers}\}$ denote the dynamic set of corrupt parties that are at the same time among the agreed upon set of dealers. At this stage, Sim already knows the internal data for all parties in $\mathcal{C}'$ and in particular $s_{i',i}$ for all $i' \in \mathcal{C}'$. Now, Sim chooses some set $\mathcal{S} \subset (\{\mathsf{dealers}\} \setminus \mathcal{C}')$ of size $t_c - |\mathcal{C}'|$ uniformly at random and queries its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$ on input elements $S_{i',i}$ for all $i' \in ([n] \setminus \mathsf{dealers}) \sqcup (\mathcal{S} \cup \mathcal{C}') := \mathcal{S}_d$. In particular, this is the usual behavior as would be done by the simulators $\mathsf{Sim}_{i'}$ for these particular $i' \in \mathcal{S}_d$. Note that the set $\mathcal{S}_d$ is of size $t_c + t_c = 2t_c$ and thus its complement $\mathcal{S}_d^c := [n] \setminus \mathcal{S}_d$ is of size $\ell = n - 2t_c$. Having done this, Sim knows the discrete logarithm value $s_{i',i}$ of $S_{i',i}$ for the $n - \ell$ indices $i' \in \mathcal{S}_d$. In particular, it knows the values for $i' \in \mathcal{S} \cup \mathcal{C}'$ which itself is of size t_c. By identities (1) and the properties of the superinvertible matrix SI, we know that knowledge of the elements $\{r_{j,i}\}_{j \in [\ell]}$ and any t_c elements among the $\{s_{1,i}, \ldots, s_{n-t_c,i}\}$ is sufficient to recover the remaining $n - 2t_c = \ell$ elements $s_{(\cdot),i}$ in this set (since the equations written in matrix form exactly give an $(\ell \times \ell)$-dimensional submatrix of SI which is invertible by definition). By taking the set $\mathcal{S} \cup \mathcal{C}'$ of size t_c for which Sim knows the corresponding $s_{(\cdot),i}$ values, it can compute all values $s_{1,i}, \ldots, s_{n-t_c,i}$ and answer the corresponding discrete logarithm oracle queries from $\mathsf{Sim}_{i'}$ on input $S_{i',i}$ for $i' \in \mathcal{S}_d^c$ (note that this set is the complement of $\mathcal{S} \cup \mathcal{C}'$ in $\mathsf{dealers}$). In total, Sim made ℓ calls to its discrete logarithm oracle for the $R_{j,i}$ with $j \in [\ell]$, but afterwards saved $(n - t_c) - t_c = \ell$ calls for those $S_{i',i}$ with $i' \in \mathcal{S}_d^c$. Finally, before Sim terminates, it invokes the simulators Sim_j for $j \notin \mathsf{dealers}$ only in such a way as as if P_j was being corrupted.

One should think of the simulators Sim_i as being responsible for the private data related to the instance AVSS_i only, and not to P_i's entire internal state from the data of all the different instances AVSS_j of other parties it obtains. In particular, any party P_i's internal state should be thought of as a disjoint union of data from each of the instances AVSS_j with $j \in [n]$ that is handled by Sim_j each.

[20] By definition of oracle-aided simulatability we know that the simulators $\mathsf{Sim}_{i'}$ ask the discrete logarithm oracle on these input elements upon corruption of party P_i.

Having said all that, we begin with the analysis of the simulator Sim in order to show that it is indeed an oracle-aided simulator for PADKG.

Properties. For this, we step-by-step go through each property required for an oracle-aided simulator according to Definition 3.1.

- *Simulatability Factor.* In the beginning of the discussion, we defined the simulatability factor $\tilde{k} := kn$ for Sim. Since k is the simulatability factor of the underlying AVSS, we know that $k \geq t_r + 1$. In particular, we have $kn \geq n(t_r + 1) \geq \ell(t_r + 1)$ since $\ell = n - 2t_c \leq n$ and thus also $\tilde{k} \in \mathsf{poly}(\lambda)$. This shows that our simulatability factor is well-defined.

- *Syntax.* Since the simulators Sim_i for all $i \in [n]$ have this property, and MVBA is secure, by design the simulator Sim outputs the correct public keys $R_1, \ldots, R_\ell$ and public key shares $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$.

- *Queries upon Corruption.* By design, the simulator Sim forwards the corruption queries directly to the individual simulators Sim_i for all $i \in [n]$. Since these have the property that they query the discrete logarithm oracle only upon corruption queries and Sim answers their discrete logarithm oracle queries by (i) calling its own oracle $\mathsf{DL}_{\mathbb{G},g}$ or (ii) by calling the oracle on $R_{j,i}$ for all $j \in [\ell]$ (once these elements are defined from the protocol execution), it directly follows that Sim has this property. As already mentioned, by design we let Sim call the oracle $\mathsf{DL}_{\mathbb{G},g}$ on input $R_{j,i}$ for all $j \in [\ell]$ (once they are defined) upon a corruption query.

- *Bad Event.* We define the bad event Bad for an execution of PADKG with the simulator Sim as $\mathsf{Bad} := \mathsf{Bad}_1 \vee \ldots \vee \mathsf{Bad}_n$ where Bad_i is the existing bad event for the execution of instance AVSS_i with the simulator Sim_i for $i \in [n]$. The standard union bound theorem tells us that

$$\Pr[\mathsf{Bad}] \leq \Pr[\mathsf{Bad}_1] + \ldots + \Pr[\mathsf{Bad}_n] \leq n \cdot \mathsf{negl}(\lambda),$$

  which itself is negligible.

- *Indistinguishability.* Conditioned on the previously defined event Bad does not happen, we know that none of the events Bad_i for $i \in [n]$ happens. In particular, each individual simulator Sim_i generates a simulation for the adversary A that is statistically indistinguishable from a real execution of AVSS_i. By a hybrid argument, it follows that the simulation Sim provides to A is also statistically indistinguishable from a real execution of PADKG, since it simulates the MVBA protocol honestly.

- *Number of $\mathsf{DL}_{\mathbb{G},g}$ Queries.* We count the total number of call Sim makes to its discrete logarithm oracle $\mathsf{DL}_{\mathbb{G},g}$. For this, we assume without loss of generality that the adversary A corrupts exactly t_c parties during an execution of PADKG. Again, let $\mathsf{dealers}$ be the agreed upon set of dealers after termination of the MVBA protocol. By construction, Sim calls its discrete logarithm oracle in one of two ways upon a corruption query on P_i. (i) To answer a discrete logarithm query from Sim_j for $j \in [n]$, and (ii) to compute $r_{j,i}$ for all $j \in [\ell]$ which is the discrete logarithm value of $R_{j,i}$. We have designed Sim in such a way that it answers the discrete logarithm queries from Sim_j for $j \notin \mathsf{dealers}$ simply by calling its own oracle $\mathsf{DL}_{\mathbb{G},g}$ on the queried element, even for $h' = S_{j,i}$ in that case. If the corruption query happened before the public key shares $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ are defined, the previous sentence is also true for any $j \in [n]$, and not only the once outside the set $\mathsf{dealers}$. However, if the corruption query happened after the public key shares $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ are defined, then by design Sim calls its oracle $\mathsf{DL}_{\mathbb{G},g}$ on input the elements $R_{1,i}, \ldots, R_{\ell,i}$ and picks a set $\mathcal{S} \subset (\{\mathsf{dealers}\} \setminus \mathcal{C}')$ of size $t_c - |\mathcal{C}'|$. Having done this, it also answers the discrete logarithm queries from Sim_j for $j \in (\mathcal{S} \cup \mathcal{C}')$ simply by calling its own oracle $\mathsf{DL}_{\mathbb{G},g}$ on the queried element, even for the elements $h' = S_{j,i}$. For the remaining ℓ simulators with indices specified by the set $\mathcal{S}_d^c$, it answers their discrete logarithm queries for all $h' \neq S_{j,i}$ as usual by calling its own oracle on the queried input. The ℓ values $s_{j,i}$ for $j \in \mathcal{S}_d^c$ are computed by knowledge of other values and the property of the superinvertible matrix SI as specified before. That way it saved ℓ further discrete logarithm oracle calls and we can think of the $\mathsf{DL}_{\mathbb{G},g}$ calls for the elements $R_{1,i}, \ldots, R_{\ell,i}$ as being a pulled back oracle call for $S_{j,i}$ with $j \in \mathcal{S}_d^c$. This observation tells us that we can without loss of generality assume that all corruptions happened before the public key shares are defined from the protocol execution. With this in mind, we can count the number of total discrete logarithm oracle calls Sim makes by summing up the number of calls from the individual simulators Sim_i. Since each simulator Sim_i is run on input k elements, we know that it makes exactly $k' = k$ calls to the discrete logarithm oracle if party P_i gets corrupted and $k' = k - \delta_a$ calls otherwise. Assuming that exactly t_c parties get corrupted and $\mathsf{dealers}$ contains these corrupt parties, we find that

$$\begin{aligned} 2t_c \cdot k + (n - 2t_c) \cdot (k - \delta_a) &= 2t_c k + nk - n\delta_a - 2t_c k + 2t_c\delta_a \\ &= nk - n\delta_a + 2t_c\delta_a \\ &= nk - (n - 2t_c)\delta_a \\ &= nk - \ell\delta_a, \end{aligned}$$

where the first summand comes from the queries for the t_c corrupt parties and the queries for the t_c parties outside the set $\mathsf{dealers}$, and the second summand comes from the queries for the remaining $(n - 2t_c)$ parties. This shows that on input nk elements, our simulator Sim makes exactly $nk - \ell\delta_a$ queries to the discrete logarithm oracle which is in line with the requirement.

- *Query Independence.* In this final bullet point we show that the discrete logarithm oracle calls the simulator Sim makes are independent in the sense that the algebraic vectors from their representation in $\xi \in \mathbb{G}^{kn}$ are independent. For this, we again assume without loss of generality that the adversary A corrupts exactly t_c parties during an execution of PADKG. Again, let $\mathsf{dealers}$ be the agreed upon set of dealers after termination of the MVBA protocol. Further, we assume that $\mathcal{C} = \{1, \ldots, t_c\}$ and $\mathsf{dealers} = \{1, \ldots, n - t_c\}$. By construction, Sim calls its discrete logarithm oracle in one of two ways upon a corruption query on P_i. (i) To answer a discrete logarithm query from Sim_j for $j \in [n]$, and (ii) to compute $r_{j,i}$ for all $j \in [\ell]$ which is the discrete logarithm value of $R_{j,i}$. If the adversary A corrupted all t_c parties before the public key shares $(R_{j,1}, \ldots, R_{j,n})$ for all $j \in [\ell]$ are defined from the protocol execution, then Sim does not make any separate calls to any $R_{j,i}$ and it simply answers all discrete logarithm oracle queries from the simulators Sim_i for all $i \in [n]$ by calling its own oracle $\mathsf{DL}_{\mathbb{G},g}$. Now we know that Sim_i for all $i \in (\mathcal{C} \cup \mathsf{dealers}^c)$ (where we denote $X^c := [n] \setminus X$ for a set $X \subset [n]$ to be the complement of X in $[n]$) makes exactly k discrete logarithm queries and that the corresponding $(k \times k)$-dimensional simulatability matrix $L_i(\emptyset, \mathcal{C})$ is invertible over $\mathbb{Z}_p$. For the remaining $n - 2t_c = \ell$ indices, we know that the simulators Sim_i each make $k - \delta_a$ discrete logarithm queries. Now let $I \subset \mathcal{H}$ be some set of size δ_a. Hereafter, for convenience we assume that $t_r = n - t_c - 1$ and $I = \{t_c + 1, \ldots, n - t_c\} \subset \mathsf{dealers}$. The cases $t_r < n - t_c - 1$ and $I \not\subseteq \mathsf{dealers}$ work analogously. By definition of the simulatability matrix for Sim_i with $i \in I$, we know that the $(k \times k)$-dimensional matrix $L_i(I, \mathcal{C})$ is invertible. Recall that the matrix $L_i(I, \mathcal{C})$ consists of the algebraic vectors of the discrete logarithm queries Sim_i makes (which are in total $k - \delta_a$, since $i \notin \mathcal{C}$) and the additional algebraic vectors corresponding to the elements $S_{i,j}$ for $j \in I$ (which are $|I| = \delta_a$ elements).

  Subsequently, we write $L_i(0I, \mathcal{C})$ to denote the $(k - \delta_a, k)$-dimensional matrix resulting from $L_i(I, \mathcal{C})$ by deleting the rows corresponding to the elements $S_{i,j}$ for $j \in I$ (i.e., the last δ_a rows). On the other hand, these additional elements (or better, their corresponding algebraic vectors) $S_{i,j}$ for $(i, j) \in I \times I$ (note that $\ell = n - 2t_c = t_r - t_c + 1 = \delta_a$) are not considered in the simulatability matrix for Sim, but instead the elements $R_{j,i}$ with (j, i) ranging over $j \in [\ell]$ and $i \in I$. Since each simulator Sim_i takes a disjoint part of the kn-wise vector ξ as input, we can arrange the matrices $L_i(\emptyset, \mathcal{C})$ for $i \in I^c$ and $L_i(0I, \mathcal{C})$ in a block diagonal matrix L where each block has full rank and has size $(k \times k)$ for $i \in I^c$ and $(k - \delta_a, k)$ for $i \in I$. In order to make the matrix L quadratic, we need to add additional $|I| \cdot \delta_a = \ell\delta_a$ rows of length kn to it. This is done by the algebraic vectors for the elements $R_{j,i}$ with (j, i) ranging over the set $[\ell] \times I$. By definition of $R_{j,i}$ and its counterpart $R'_{j,i}$, we recall that $r_{j,i} = m_{j,1}s_{1,i} + \ldots + m_{j,n-t_c}s_{n-t_c,i}$ for all $j \in [\ell]$, where the $m_{j,i}$ are the coefficients of the superinvertible matrix SI. As a result, the algebraic representation of $R_{j,i}$ is a long vector with the first k coordinates being $m_{j,1}$ multiplied by the representation for $S_{1,i}$, the next k coordinates being $m_{j,2}$ multiplied by the representation for $S_{2,i}$, and so on. Adding these long vectors as additional rows into the matrix L, we obtain an (kn, kn)-dimensional square matrix whose upper part L_u consists of blocks and its lower part L_l of long vectors as described before. The first block column (by this we mean the first k columns) of the lower part L_l are the vectors $\{m_{1,1}\mathsf{rep}(S_{1,i}), \ldots, m_{\ell,1}\mathsf{rep}(S_{1,i})\}_{i \in I}$. The second up to $(n - t_c)$-th block columns are framed in a similar way with respective factors $m_{j,i}$ and vectors $\mathsf{rep}(S_{j,i})$. Since we want to show that this resulting matrix L is invertible, which is factually the simulatability matrix $L(\mathcal{I}, \mathcal{C})$ for Sim for the set $\mathcal{I} := I \times \ldots \times I \subset \mathcal{H}^\ell$ (ℓ-fold Cartesian product), it suffices to show that the rows given in L_l are linearly independent from the rows in L_u.

Since L_u consists of full rank blocks, it is even enough to show that the rows in L_l cut in between the (t_c+1)-th and $(n-t_c)$-th block column are linearly independent from the rows in L_u cut in between the (t_c+1)-th and $(n-t_c)$-th block column. By carefully looking, we observe that the former rows when written in matrix form just result from the multiplication of the $(\ell \times \ell)$-square submatrix $\mathsf{SI}_{[t_c]}$ of SI (where the first t_c columns are deleted) with the matrix formed by the vectors $S_{t_c+1,i}, \ldots, S_{n-t_c,i}$ with $i \in I$. By definition of an SI matrix, we know that $\mathsf{SI}_{[t_c]}$ is invertible and since the vectors $S_{j,i}$ are linearly independent from the other vectors in the j-th block column by definition of the simulatability matrix for Sim_j, we know that this final transformation is isomorphic and also gives rows that are linearly independent from the rows in the upper part L_l (in between the aforementioned block columns). As a result, we find that the final matrix $L := L(\mathcal{I}, \mathcal{C})$ for the simulator Sim is also invertible of dimension $(\tilde{k}, \tilde{k})$ which is exactly what we want. This proves the query independence property.

This concludes the proof of Theorem 5.2. □

C.3 Proof for AVSS

Before we start with a proof of Theorem 6.2, we show that the Pedersen commitment used in our construction of HAVSS (cf. Algorithms 2 to 4) satisfies three properties that we show in Lemmata C.2 to C.4. In Lemma C.2, we want to show the property of interpolation-binding as defined in [AJM+23]. Informally, this notion tells us that if an adversary generates a commitment for a polynomial of degree t_c and opens t_c+1 evaluations of it, then these evaluations fully determine the committed polynomial. More concretely, interpolating these points and computing the commitment to the interpolated polynomial will yield the same commitment.

Lemma C.2 *Consider parameters $(\mathbb{G}, p, g)$. Assume that $\mathsf{PS}_{\mathsf{open}}$ is a proof of knowledge and that the DLOG assumption (i.e., the one-more discrete logarithm assumption of degree 1) holds relative to $(\mathbb{G}, p, g)$. Further, consider an algorithm A and the following experiment:*

1. *Sample $g_0, \ldots, g_{t_c} \leftarrow\$ \mathbb{G}$ and run A on input $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.*
2. *Obtain from A a group element $\mathsf{cm} \in \mathbb{G}$ and t_c+1 triples $((x_i, y_i, \pi_i))_{i\in[t_c+1]}$ where $x_i, y_i \in \mathbb{Z}_p$.*
3. *Output 1 if and only if the following properties hold, otherwise output 0:*
   (a) *All x_i are distinct, i.e., for all $i \neq j$, we have $x_i \neq x_j$.*
   (b) *All proofs verify, i.e., $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}, x_i, y_i), \pi_i) = 1$.*
   (c) *We have $\mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{a_j}$, where $A(X) = \sum_{j=0}^{t_c} a_j X^j \in \mathbb{Z}_p[X]$ is the unique polynomial of degree at most t_c such that $A(x_i) = y_i$ for all $i \in [t_c+1]$.*

Then, for every PPT algorithm A, the probability that the game outputs 1 is negligible.

Proof. Let A be a PPT algorithm and let ε be the probability that the game in the lemma outputs 1. Our goal is to upper bound ε. To do so, we provide a sequence of games.

Game $\mathbf{G}_0$: This game is the game in the lemma. By definition, we have

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

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but when $\mathbf{G}_0$ would output 1, the new game $\mathbf{G}_1$ additionally runs $(b_0, \ldots, b_{t_c}) \leftarrow \mathsf{PExt}((\mathsf{cm}, x_1, y_1), \pi_1, \mathcal{Q})$. Here, PExt is the extractor from the proof of knowledge property of $\mathsf{PS}_{\mathsf{open}}$ and $\mathcal{Q}$ is the list of random oracle queries with respect to H. That is, the game tries to extract a witness $(b_0, \ldots, b_{t_c}) \in \mathbb{Z}_p^{t_c+1}$ specifying a polynomial $B(X) = \sum_{j=0}^{t_c} b_j X^j$ from the proof π_1. If either $\mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{b_j}$ or $y_1 \neq B(x_1)$, the game aborts and we say the event Bad_1 occurs. Otherwise, it outputs 1 as $\mathbf{G}_0$ does. Clearly, $\mathbf{G}_0$ and $\mathbf{G}_1$ differ only if Bad_1 occurs. We can easily bound the probability of event Bad_1 using a reduction breaking the proof of knowledge property of $\mathsf{PS}_{\mathsf{open}}$. The reduction forwards random oracle queries and responses between A and the proof of knowledge game and outputs the statement (cm, x_1, y_1) and the proof π_1. We get

$$\Pr[\mathbf{G}_0 \Rightarrow 1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \Pr[\mathsf{Bad}_1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but when $\mathbf{G}_1$ would output 1, the new game $\mathbf{G}_2$ additionally identifies the minimal index $\hat{i} \in [t_c + 1]$ such that $A(x_{\hat{i}}) \neq B(x_{\hat{i}})$. We can see that such an index exists as follows: first, since $\prod_{j=0}^{t_c} g_j^{b_j} = \mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{a_j}$, it must be the case that $B \neq A$. As both are of degree t_c, $B(x_i) = A(x_i)$ can only hold for at most t_c indices $i \in [t_c + 1]$. Once this minimal index $\hat{i}$ is identified, the game computes a witness $(c_0, \ldots, c_{t_c}) \leftarrow \mathsf{PExt}((\mathsf{cm}, x_{\hat{i}}, y_{\hat{i}}), \pi_{\hat{i}}, \mathcal{Q})$, i.e., it computes a polynomial $C = \sum_{j=0}^{t_c} c_j X^j$ by running the proof of knowledge extractor another time. If either $\mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{c_j}$ or $y_{\hat{i}} \neq C(x_{\hat{i}})$, the game aborts and we say the event Bad_2 occurs. Otherwise, it outputs 1 as $\mathbf{G}_1$ does. Again, $\mathbf{G}_1$ and $\mathbf{G}_2$ differ only if Bad_2 occurs and we can bound the probability of Bad_2 using a reduction breaking the proof of knowledge property of $\mathsf{PS}_{\mathsf{open}}$. We get

$$\Pr[\mathbf{G}_1 \Rightarrow 1] \leq \Pr[\mathbf{G}_2 \Rightarrow 1] + \Pr[\mathsf{Bad}_2] \leq \Pr[\mathbf{G}_2 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Note what we have achieved so far: if $\mathbf{G}_2$ outputs 1, then we know $A(x_{\hat{i}}) \neq B(x_{\hat{i}})$ and $A(x_{\hat{i}}) = y_{\hat{i}} = C(x_{\hat{i}})$. In combination, we get $B(x_{\hat{i}}) \neq C(x_{\hat{i}})$ and especially $B \neq C$. Thus, there has to exists at least one j^* such that $b_{j^*} \neq c_{j^*}$. But B and C commit to the same element cm. We will now use this to break DLOG.

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but in the beginning it samples $j' \leftarrow\$ [\![t_c]\!]$. Then, if $\mathbf{G}_2$ would output 1 but $j' \neq j^*$, where j^* is as above, the game aborts. By the discussion above, and as A's view is independent from j', we get

$$\Pr[\mathbf{G}_3 \Rightarrow 1] \geq \frac{1}{t_c + 1} \cdot \Pr[\mathbf{G}_2 \Rightarrow 1].$$

Finally, we bound the probability that $\mathbf{G}_3$ outputs 1. For that, we sketch a reduction that breaks the DLOG assumption whenever $\mathbf{G}_3$ outputs 1:

1. The reduction gets as input parameters $(\mathbb{G}, p, g)$ and an element $h = g^x \in \mathbb{G}$. The goal is to compute the discrete logarithm x of h to base g.

2. The reduction samples $j' \leftarrow\$ [\![t_c]\!]$ as $\mathbf{G}_3$ does and sets $g_{j'} := h$. For every $j \in [\![t_c]\!]$ with $j \neq j'$ the reduction samples $\delta_j \leftarrow\$ \mathbb{Z}_p$ and sets $g_j := g^{\delta_j}$.

3. The reduction continues running $\mathbf{G}_3$, which includes extracting the polynomials B and C.

4. If $\mathbf{G}_3$ outputs 1, we know that

$$x \cdot b_{j^*} + \sum_{j \neq j^*} b_j \delta_j = x \cdot c_{j^*} + \sum_{j \neq j^*} c_j \delta_j.$$

   Isolating x, we get

$$x = \frac{\sum_{j \neq j^*} \delta_j (b_j - c_j)}{c_{j^*} - b_{j^*}}.$$

   The reduction now computes x in this way and outputs it.

It is clear that the reduction perfectly simulates $\mathbf{G}_3$ for A and finds the correct x whenever $\mathbf{G}_3$ outputs 1. Using the DLOG assumption, we get

$$\Pr[\mathbf{G}_3 \Rightarrow 1] \leq \mathsf{negl}(\lambda).$$

□

In Lemma C.3, we prove that if an adversary generates a commitment cm, a polynomial A whose commitment is cm, and a valid opening of the commitment, then the opening is consistent with A. This can be seen as a relaxed version of the well-known evaluation-binding property of polynomial commitments.

Lemma C.3 *Consider parameters $(\mathbb{G}, p, g)$. Assume that $\mathsf{PS}_{\mathsf{open}}$ is a proof of knowledge and that the DLOG assumption (i.e., the one-more discrete logarithm assumption of degree 1) holds relative to $(\mathbb{G}, p, g)$. Further, consider an algorithm A and the following experiment:*

1. *Sample $g_0, \ldots, g_{t_c} \leftarrow_{\$} \mathbb{G}$ and run A on input $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.*
2. *Obtain from A a group element $\mathsf{cm} \in \mathbb{G}$, a polynomial $A \in \mathbb{Z}_p[x]$, and a triple (x, y, π) where $x, y \in \mathbb{Z}_p$.*
3. *Output 1 if and only if the following properties hold, otherwise output 0:*
   (a) A is of degree at most t_c.
   (b) The proof verifies, i.e., $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}, x, y), \pi) = 1$.
   (c) We have $\mathsf{cm} = \prod_{j=0}^{t_c} g_j^{a_j}$, where $A(X) = \sum_{j=0}^{t_c} a_j X^j$.
   (d) We have $A(x) \neq y$.

Then, for every PPT algorithm A, the probability that the game outputs 1 is negligible.

Proof. Let A be a PPT algorithm and let ε be the probability that the game in the lemma outputs 1. Our goal is to upper bound ε. To do so, we provide a sequence of games.

Game $\mathbf{G}_0$: This game is the game in the lemma. By definition, we have

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

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but when $\mathbf{G}_0$ would output 1, the new game $\mathbf{G}_1$ additionally runs $(b_0, \ldots, b_{t_c}) \leftarrow \mathsf{PExt}((\mathsf{cm}, x, y), \pi, \mathcal{Q})$. Here, PExt is the extractor from the proof of knowledge property of $\mathsf{PS}_{\mathsf{open}}$ and $\mathcal{Q}$ is the list of random oracle queries with respect to H. That is, the game tries to extract a witness $(b_0, \ldots, b_{t_c}) \in \mathbb{Z}_p^{t_c+1}$ specifying a polynomial $B(X) = \sum_{j=0}^{t_c} b_j X^j$ from the proof π. If either $\mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{b_j}$ or $y \neq B(x)$, the game aborts and we say the event Bad_1 occurs. Otherwise, it outputs 1 as $\mathbf{G}_0$ does. Clearly, $\mathbf{G}_0$ and $\mathbf{G}_1$ differ only if Bad_1 occurs. We can easily bound the probability of event Bad_1 using a reduction breaking the proof of knowledge property of $\mathsf{PS}_{\mathsf{open}}$. The reduction forwards random oracle queries and responses between A and the proof of knowledge game and outputs the statement (cm, x, y) and the proof π. We get

$$\Pr[\mathbf{G}_0 \Rightarrow 1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \Pr[\mathsf{Bad}_1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Note that if $\mathbf{G}_1$ outputs 1, then we know that $A(x) \neq y = B(x)$, and thus $A \neq B$. As a consequence, there exists an index $j^* \in [\![t_c]\!]$ such that $a_{j^*} \neq b_{j^*}$. We will now use this to break DLOG, similarly to what we have done in Lemma C.2.

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but in the beginning it samples $j' \leftarrow_{\$} [\![t_c]\!]$. Then, if $\mathbf{G}_2$ would output 1 but $j' \neq j^*$, where j^* is as above, the game aborts. By the discussion above, and as A's view is independent from j', we get

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \geq \frac{1}{t_c + 1} \cdot \Pr[\mathbf{G}_1 \Rightarrow 1].$$

Finally, we bound the probability that $\mathbf{G}_2$ outputs 1. For that, we sketch a reduction that breaks the DLOG assumption whenever $\mathbf{G}_2$ outputs 1:

1. The reduction gets as input parameters $(\mathbb{G}, p, g)$ and an element $h = g^x \in \mathbb{G}$. The goal is to compute the discrete logarithm x of h to base g.
2. The reduction samples $j' \leftarrow_{\$} [\![t_c]\!]$ as $\mathbf{G}_2$ does and sets $g_{j'} := h$. For every $j \in [\![t_c]\!]$ with $j \neq j'$ the reduction samples $\delta_j \leftarrow_{\$} \mathbb{Z}_p$ and sets $g_j := g^{\delta_j}$.
3. The reduction continues running $\mathbf{G}_2$, which includes extracting B.

4. If $\mathbf{G}_2$ outputs 1, we know that
$$x \cdot a_{j^*} + \sum_{j \neq j^*} a_j \delta_j = x \cdot b_{j^*} + \sum_{j \neq j^*} b_j \delta_j.$$
Isolating x, we get
$$x = \frac{\sum_{j \neq j^*} \delta_j (a_j - b_j)}{b_{j^*} - a_{j^*}}.$$
The reduction now computes x in this way and outputs it.

It is clear that the reduction perfectly simulates $\mathbf{G}_2$ for A and finds the correct x whenever $\mathbf{G}_2$ outputs 1. Using the DLOG assumption, we get
$$\Pr\left[\mathbf{G}_2 \Rightarrow 1\right] \leq \mathsf{negl}(\lambda).$$
□

In Lemma C.4, we prove a variation of the property shown in Lemma C.3. Namely, we show that if an adversary generates a commitment cm, a polynomial A whose commitment is cm, and a valid exponentiated opening of cm, then the exponentiated opening is consistent with A. That is, if it provides a group element Y and a valid proof π of an exponentiated opening at a point x, then $Y = g^{A(x)}$.

Lemma C.4 *Consider parameters $(\mathbb{G}, p, g)$. Assume that $\mathsf{PS}_{\mathsf{exp}}$ is a proof of knowledge and that the DLOG assumption (i.e., the one-more discrete logarithm assumption of degree 1) holds relative to $(\mathbb{G}, p, g)$. Further, consider an algorithm A and the following experiment:*

1. *Sample $g_0, \ldots, g_{t_c} \leftarrow\$\, \mathbb{G}$ and run A on input $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.*
2. *Obtain from A a group element $\mathsf{cm} \in \mathbb{G}$, a polynomial $A \in \mathbb{Z}_p[x]$, and a triple (x, Y, π) where $x \in \mathbb{Z}_p$ and $y \in \mathbb{G}$.*
3. *Output 1 if and only if the following properties hold, otherwise output 0:*
   - *(a) A is of degree at most t_c.*
   - *(b) The proof verifies, i.e., $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{exp}}((\mathsf{cm}, x, Y), \pi) = 1$.*
   - *(c) We have $\mathsf{cm} = \prod_{j=0}^{t_c} g_j^{a_j}$, where $A(X) = \sum_{j=0}^{t_c} a_j X^j$.*
   - *(d) We have $g^{A(x)} \neq Y$.*

Then, for every PPT algorithm A, the probability that the game outputs 1 is negligible.

Proof. The proof is very similar to that of Lemma C.3. Let A be a PPT algorithm and let ε be the probability that the game in the lemma outputs 1. Our goal is to upper bound ε. To do so, we provide a sequence of games.

Game $\mathbf{G}_0$: This game is the game in the lemma. By definition, we have
$$\varepsilon = \Pr\left[\mathbf{G}_0 \Rightarrow 1\right].$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but when $\mathbf{G}_0$ would output 1, the new game $\mathbf{G}_1$ additionally runs $(b_0, \ldots, b_{t_c}) \leftarrow \mathsf{PExt}((\mathsf{cm}, x, Y), \pi, \mathcal{Q})$. Here, PExt is the extractor from the proof of knowledge property of $\mathsf{PS}_{\mathsf{exp}}$ and $\mathcal{Q}$ is the list of random oracle queries with respect to H. That is, the game tries to extract a witness $(b_0, \ldots, b_{t_c}) \in \mathbb{Z}_p^{t_c+1}$ specifying a polynomial $B(X) = \sum_{j=0}^{t_c} b_j X^j$ from the proof π. If either $\mathsf{cm} \neq \prod_{j=0}^{t_c} g_j^{b_j}$ or $Y \neq g^{B(x)}$, the game aborts and we say the event Bad_1 occurs. Otherwise, it outputs 1 as $\mathbf{G}_0$ does. Clearly, $\mathbf{G}_0$ and $\mathbf{G}_1$ differ only if Bad_1 occurs. We can easily bound the probability of event Bad_1 using a reduction breaking the proof of knowledge property of $\mathsf{PS}_{\mathsf{exp}}$. The reduction forwards random oracle queries and responses between A and the proof of knowledge game and outputs the statement (cm, x, Y) and the proof π. We get
$$\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] \leq \Pr\left[\mathbf{G}_1 \Rightarrow 1\right] + \Pr\left[\mathsf{Bad}_1\right] \leq \Pr\left[\mathbf{G}_1 \Rightarrow 1\right] + \mathsf{negl}(\lambda).$$

Note that if $\mathbf{G}_1$ outputs 1, then we know that $g^{A(x)} \neq Y = g^{B(x)}$, and thus $A \neq B$. As a consequence, there exists an index $j^* \in [\![t_c]\!]$ such that $a_{j^*} \neq b_{j^*}$. But A and B commit to the same element cm. We will now use this to break DLOG, similarly to what we have done in previous lemmata.

<u>**Game $\mathbf{G}_2$:**</u> This game is as $\mathbf{G}_1$, but in the beginning it samples $j' \leftarrow_{\$} [\![t_c]\!]$. Then, if $\mathbf{G}_2$ would output 1 but $j' \neq j^*$, where j^* is as above, the game aborts. By the discussion above, and as A's view is independent from j', we get

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \geq \frac{1}{t_c+1} \cdot \Pr[\mathbf{G}_1 \Rightarrow 1].$$

Finally, we bound the probability that $\mathbf{G}_2$ outputs 1. For that, we sketch a reduction that breaks the DLOG assumption whenever $\mathbf{G}_2$ outputs 1:

1. The reduction gets as input parameters $(\mathbb{G}, p, g)$ and an element $h = g^x \in \mathbb{G}$. The goal is to compute the discrete logarithm x of h to base g.

2. The reduction samples $j' \leftarrow_{\$} [\![t_c]\!]$ as $\mathbf{G}_2$ does and sets $g_{j'} := h$. For every $j \in [\![t_c]\!]$ with $j \neq j'$ the reduction samples $\delta_j \leftarrow_{\$} \mathbb{Z}_p$ and sets $g_j := g^{\delta_j}$.

3. The reduction continues running $\mathbf{G}_2$, which includes extracting B.

4. If $\mathbf{G}_2$ outputs 1, we know that

$$x \cdot a_{j^*} + \sum_{j \neq j^*} a_j \delta_j = x \cdot b_{j^*} + \sum_{j \neq j^*} b_j \delta_j.$$

   Isolating x, we get

$$x = \frac{\sum_{j \neq j^*} \delta_j (a_j - b_j)}{b_{j^*} - a_{j^*}}.$$

   The reduction now computes x in this way and outputs it.

It is clear that the reduction perfectly simulates $\mathbf{G}_2$ for A and finds the correct x whenever $\mathbf{G}_2$ outputs 1. Using the DLOG assumption, we get

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \leq \mathsf{negl}(\lambda).$$

□

In Lemma C.5, we show that once t_c+1 honest parties send their "column" messages as in the protocol specification (cf. Algorithm 3), it is possible to efficiently extract a polynomial for the AVSS instance from their views. By this, we mean that it is possible to compute a bivariate polynomial $S(X,Y)$ of correct degrees in X and Y such that all opened evaluations are consistent with S and all of the exponentiated openings S_i are such that $S_i = g^{S(i,0)}$ holds. Informally, we do this by taking the evaluations of these t_c+1 honest parties along each of the columns. Each set of t_c+1 evaluations should fully define the committed polynomial, as shown in Lemma C.2. All opened evaluations and exponentiated evaluations should be consistent with these polynomials, as shown in Lemmata C.3 and C.4. Once this is established, it will be easy to show that all honest parties' outputs will be consistent with S because they compute their outputs as functions of the openings they received. Note that this process can actually be done with any set of t_c+1 honest parties that sent "column" messages, but we choose the first t_c+1 parties that do so. If we construct a polynomial in this way from any other set of t_c+1 parties, the evaluations must also be consistent with S (as shown in the following lemma), and thus we will also construct S.

Lemma C.5 *Consider parameters $(\mathbb{G}, p, g)$ and $g_0, \ldots, g_{t_c} \leftarrow_{\$} \mathbb{G}$. Assume that $\mathsf{PS}_{\mathsf{open}}$ and $\mathsf{PS}_{\mathsf{exp}}$ are proofs of knowledge and that the DLOG assumption holds relative to $(\mathbb{G}, p, g)$. Consider the experiment of running $\mathsf{HAVSS.Share}$ as specified in Algorithm 3. Assume that in this experiment t_c+1 honest parties send "column" messages at some point throughout the protocol after they receive a $\langle$"commits"$, \mathsf{CM}', (S_i, \pi_i^{exp})_{i \in [t_r+1]}\rangle$ broadcast from the dealer.*

Then, it is possible to efficiently compute a bivariate polynomial $S(X,Y)$ of degree t_r in X and t_c in Y from the views of the first t_c+1 honest parties that do so (i.e., send "column" messages) such that for every PPT adversary corrupting at most t_c parties, the following hold with all but negligible probability:

(a) For all $i \in [t_r + 1]$, we have $S_i = g^{S(i,0)}$, and

(b) if any honest P_i adds $(j, C_i(j))$ to $\mathsf{points}_{col,i}$, then $C_i(j) = S(i, j)$.

Further, if the dealer is honest, then $S(X, Y)$ is the polynomial it sampled in HAVSS.Deal *(cf. Algorithm 2).*

Proof. Assume that $t_c + 1$ honest parties sent such messages, and let $I \subseteq [n]$ be the indices of the first honest parties to do so. Before doing so, they must have had $\mathsf{CM} \neq \bot$, and thus they received a "commits" message from the dealer, verified it, and updated both CM and S by using the $\mathsf{ExpInterpolate}$ algorithm on the received values. In addition, each P_i received a "row" message containing a vector $((C_j(i), \pi_{j,i}))_{j \in [n]}$ and verifying it before sending the "column" message. For each $i \in I$, define $R_i(X)$ to be the unique polynomial of degree t_r or less such that $R_i(j) = C_j(i)$ for every $i \in [t_r + 1]$. In addition, let $S(X, Y)$ be the unique polynomial of degree at most t_r in X and at most t_c in Y such that $S(X, i) = R_i(X)$ for every $i \in I$. Note that it is indeed possible to efficiently interpolate both of these polynomials from the views of the aforementioned parties. Also, if the dealer is honest, then it sampled a polynomial $S'(X, Y)$ and sent each party P_i "row" messages with evaluations on the polynomial $R'_i(X) = S'(X, i)$. These means that the interpolated R_i polynomials are the R'_i polynomials and thus $S(X, Y)$ is the polynomial $S'(X, Y)$ sampled by the dealer.

Now that this is established, we need to bound the probability that one of the two properties in the lemma do not hold, with respect to $S(X, Y)$. To this end, let A be a PPT adversary running in the protocol. We denote by ε the probability that there is an $i \in [t_r + 1]$ with $S_i \neq g^{S(i,0)}$ or some honest party P_i adds $(j, C_i(j))$ to $\mathsf{points}_{col,i}$ but $C_i(j) \neq S(i, j)$.

<u>**Game $\mathbf{G}_0$:**</u> This game is the game that runs the protocol with the adversary. It outputs 1 if there is an $i \in [t_r + 1]$ with $S_i \neq g^{S(i,0)}$ or some honest party P_i adds $(j, C_i(j))$ to $\mathsf{points}_{col,i}$ but $C_i(j) \neq S(i, j)$. We have

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

<u>**Game $\mathbf{G}_1$:**</u> This game is the same as $\mathbf{G}_0$, but if at any point $t_c + 1$ honest parties send "column" messages after accepting a "commits" message from the dealer, it computes $R_i(X)$ for every such party and $S(X, Y)$ as defined above. It then computes $C_i(Y) = S(i, Y) = \sum_{k=0}^{t_c} a_{k,i} Y^k$ for every $i \in [t_r + 1]$. If $\mathsf{cm}_i \neq \prod_{k=0}^{t_c} g_k^{a_{k,i}}$ for any $i \in [t_r + 1]$, then $\mathbf{G}_1$ aborts and we say that event Bad_1 occurred. Otherwise, it acts identically to $\mathbf{G}_0$ and outputs 1 when it does.

Clearly, $\mathbf{G}_0$ and $\mathbf{G}_1$ only differ if the event Bad_1 occurs. We can bound the probability that the event occurs by the following reduction to Lemma C.2:

1. The reduction gets as input parameters $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.
2. The reduction runs the protocol as $\mathbf{G}_1$ does, and if the event Bad_1 ever occurs, it finds an index $i \in [t_r + 1]$ such that $\mathsf{cm}_i \neq \prod_{k=0}^{t_c} g_k^{a_{k,i}}$. It then outputs cm_i and the triples $((x_i, y_i, \pi_i))_{i \in I}$ where I is the set of $t_c + 1$ honest parties that sent "row" messages first.

Note that before sending a "column" message, every honest party makes sure that all of the openings and proofs it received in its "row" message verify. This means that if the event Bad_1 occurred, the reduction wins in the game described in Lemma C.2. Therefore:

$$\Pr[\mathbf{G}_0 \Rightarrow 1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \Pr[\mathsf{Bad}_1] \leq \Pr[\mathbf{G}_1 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

<u>**Game $\mathbf{G}_2$:**</u> This game is the same as $\mathbf{G}_1$, and it computes S in the same way as $\mathbf{G}_1$. However, it stores all "column" messages received by honest parties throughout the protocol. If at any point, an honest party i receives a $\langle$"column"$, C'_i(j), \pi_{i,j}\rangle$ message such that $C'_i(j) \neq S(i, j)$ and $\mathsf{PVer}_{\mathsf{open}}((\mathsf{cm}_i, j, C'_i(j)), \pi_{i,j}) = 1$, $\mathbf{G}_2$ aborts and we say that event Bad_2 occurred. Otherwise, $\mathbf{G}_2$ continues the same as $\mathbf{G}_1$ and outputs 1 in the same situations. Clearly, G_1 and G_2 only differ if the event Bad_1 does not occur and the event Bad_2 does occur. We can bound the probability that Bad_2 takes place given that Bad_1 did not take place by the following reduction to Lemma C.3:

1. The reduction gets as input parameters $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.
2. The reduction runs the protocol as $\mathbf{G}_2$ does, and if the event Bad_2 ever occurs without Bad_1 taking place, it find the indices i, j described above such that $\mathsf{PVer}_{\mathsf{open}}((\mathsf{cm}_i, j, C'_i(j)), \pi_{i,j}) = 1$, but $C'_i(j) \neq S(i, j)$. It then outputs cm_i, the polynomial $C_i(Y) = S(i, Y)$ and $(j, C'_i(j), \pi_{i,j})$.

Given that Bad_1 does not occur, cm_i is indeed a commitment to C_i. That is, if $C_i(Y) = \sum_{k=0}^{t_c} a_k X^k$, then $\mathsf{cm}_i = \prod_{k=0}^{t_c} g_k^{a_k}$. Therefore, the reduction outputs a commitment, a consistent polynomial and an inconsistent opening that verifies, winning in the game described in Lemma C.3. Note that $\mathbf{G}_1$ and $\mathbf{G}_2$ only differ if Bad_2 occurs, and thus:

$$\Pr[\mathbf{G}_1 \Rightarrow 1] \leq \Pr[\mathbf{G}_2 \Rightarrow 1] + \Pr[\mathsf{Bad}_2 | \neg\mathsf{Bad}_1] \leq \Pr[\mathbf{G}_2 \Rightarrow 1] + \mathsf{negl}(\lambda).$$

Finally, we bound the probability of $\mathbf{G}_2$ outputting 1. First of all, the game never outputs 1 if the events Bad_1 or Bad_2 take place. This means that no honest party i received a "column" message with an evaluation and verifying proof that is inconsistent with $C_i(Y) = S(i, Y)$. Since those are the only values it adds to $\mathsf{points}_{col,i}$, $\mathbf{G}_2$ cannot output 1 as a result of the second bullet described in the lemma. Therefore, $\mathbf{G}_2$ only outputs 1 if there exists some $i \in [t_r + 1]$ such that $S_i \neq g^{S(i,0)}$. We denote this event Bad_3. Note that honest parties only send their "column" messages if they receive a "commits" broadcast from the dealer with verifying proofs. Thus, we can construct the following reduction to Lemma C.4:

1. The reduction gets as input parameters $(\mathbb{G}, p, g, g_0, \ldots, g_{t_c})$.
2. The reduction runs the protocol as $\mathbf{G}_2$ does, and if the event Bad_3 ever occurs without Bad_1 taking place, it find the index i described above such that $\mathsf{PVer}_{\mathsf{exp}}((\mathsf{cm}_i, 0, S_i), \pi_i^{exp}) = 1$, but $S_i \neq g^{S(i,0)}$. It then outputs cm_i, the polynomial $C_i(Y) = S(i, Y)$ and $(0, S_i, \pi_i^{exp})$.

As before, $\mathbf{G}_2$ only outputs 1 if the event Bad_3 occurs and the event Bad_1 does not occur and thus:

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \leq \Pr[\mathsf{Bad}_3 | \neg\mathsf{Bad}_1] \leq \mathsf{negl}(\lambda).$$

□

To prove security of our AVSS (Theorem 6.2), we need to prove correctness and termination as defined in Definition 2.4 and oracle-aided simulatability as defined in Definition 5.1. We do so in the following lemmata. We begin with the correctness property.

Lemma C.6 *Assume that* $\mathsf{PS}_{\mathsf{open}}$ *and* $\mathsf{PS}_{\mathsf{exp}}$ *are proofs of knowledge and that the* DLOG *assumption holds relative to the parameters of the protocol* $(\mathbb{G}, p, g)$. *Then,* $\mathsf{HAVSS} = (\mathsf{HAVSS.Share}, \mathsf{HAVSS.Rec})$ *has the Correctness property.*

Proof. To recall, we need to show that – in presence of at most t_c corruptions – when the first honest party completes HAVSS.Share, then there is a unique polynomial $f \in \mathbb{Z}_p[X]$ of degree at most t_r such that every honest party completing HAVSS.Rec outputs $(S, S_1, \ldots, S_n)$ and $f(i) \in \mathbb{Z}_p$ with $S = g^{f(0)}$ and $S_j = g^{f(j)}$ for all $j \in [n]$. If the dealer is honest, it has to hold that $f(0)$ is equal to the dealer's input.

We now give the proof. Assume some honest party completes the protocol. Before it does so, it receives "done" messages from $n - t_c$ different parties. We know that $n - t_c \geq 2t_c + 1$ and thus at least one of these messages was sent by an honest party. Consider the first honest party to send a "done" message. At that time, it did not receive a "done" message from another honest party, so it received at most t_c "done" messages. This means that it must have sent its "done" message after receiving "vote" messages from at least $n - t_c$ parties. Out of those, at least $n - 2t_c \geq t_c + 1$ are honest. Honest parties only send such messages if they have $\mathsf{CM} \neq \bot$, and they only update CM after receiving a "commits" broadcast from the dealer. In addition, before sending "vote" messages, these parties also send "column" message to everybody. Therefore we have successfully argued that the conditions of Lemma C.5 hold. Now, recall Lemma C.5 implies the existence of a bivariate polynomial $S(X, Y)$ with certain properties. We will show that the Correctness property holds with respect to $f(X) = S(X, 0)$.

First, S is of degree t_r in X and thus f is also of degree t_r. Let $\mathsf{row}'_0 = ((S_i, \pi_i^{\mathrm{exp}}))_{i \in [t_r+1]}$ be the value broadcast by the dealer in its "commits" message. By Lemma C.5, we have $S_i = g^{S(i,0)} = g^{f(0)}$ for every $i \in [t_r + 1]$. Parties then interpolate the S_i values and evaluate them at 0 to get S_0 and at $t_r + 2, \ldots, n$ to get $S_{t_r+2}, \ldots, S_n$. Since the discrete logarithms of the S_i elements lie on $f(X)$ for every $i \in [t_r + 1]$, it is also true that $S_i = g^{f(i)}$ for every one of the interpolated values. In addition, assume some honest party P_i completed the HAVSS.Share protocol. It only did so after having $C_i \neq \bot$, which it updated by interpolating the pairs $(j, C_i(j))$ in $\mathsf{points}_{col,i}$ once there are $t_c + 1$ such pairs. By Lemma C.5, for every such pair $(j, C_i(j))$, we have $C_i(j) = S(i, j)$. This means that the interpolated polynomial $C_i(Y)$ must be

$S(i, Y)$ the unique polynomial of degree t_c or less that is consistent with all of these evaluations. When parties output values in $\mathsf{HAVSS.Rec}$, they output $(S_0, \ldots, S_n)$ and $C_i(0)$. As shown above, $S_i = g^{f(i)}$ for every $i \in [0, n]$. In addition, $C_i(0) = S(i, 0) = f(i)$, as required. Finally, if the dealer is honest, then S is the polynomial sampled by it, meaning that $f(0) = S(0, 0) = s$. ☐

Lemma C.7 *Assume that* $\mathsf{PS_{open}}$ *is a proof of knowledge and that the* DLOG *assumption holds relative to the parameters of the protocol* $(\mathbb{G}, p, g)$*. Then,* $\mathsf{HAVSS} = (\mathsf{HAVSS.Share}, \mathsf{HAVSS.Rec})$ *has the Termination property.*

Proof. We have to prove three properties, namely (1) if the dealer is honest and all honest parties call $\mathsf{HAVSS.Share}$, then all honest parties complete $\mathsf{HAVSS.Share}$, (2) if all honest parties call $\mathsf{HAVSS.Share}$ and an honest party completes $\mathsf{HAVSS.Share}$, then all honest parties complete $\mathsf{HAVSS.Share}$, and (3) if all honest parties call $\mathsf{HAVSS.Rec}$, then all honest parties complete $\mathsf{HAVSS.Rec}$.

We start with (1). So, assume that the dealer is honest and that all honest parties called $\mathsf{HAVSS.Share}$. Recall that the dealer starts by sampling a polynomial $S(X, Y)$ and computing $S_i = g^{S(i,0)}$, $C_i(Y) = \sum_{k=0}^{t_c} a_{k,i} Y^k$ and $\mathsf{cm}_i = \prod_{k=0}^{t_c} g_k^{a_{k,i}}$ for every $i \in [t_r + 1]$. It then reliably broadcasts a "commits" message containing these S_i and cm_i values along with proofs that the discrete log of S_i is indeed $C_i(0)$. It similarly computes cm_i and $C_i(Y)$ for every other $i \in [t_r + 2, n]$, and sends every P_i the values $C_j(i)$ for every $j \in [n]$ along with proofs that these are the correct values in "row" messages. Since all of these proofs are generated honestly, all honest parties will accept these proofs (by completeness of the non-interactive proof system), update their CM and S variables and send "vote" messages. They will also accept the "row" message and send "column" messages to all parties with the same values and proofs. Every honest party will accept "column" messages from every other honest party, for a total of at least $n - t_c$ messages. After accepting $t_c + 1$ such messages, every honest party P_i will interpolate a polynomial C_i. In addition, after receiving "vote" messages from $n - t_c$ parties, every honest party will send a "done" message to all parties. After receiving those messages from all honest parties, every party will have received "done" messages from $n - t_c$ parties and will have $S \neq \bot$ and $C_i \neq \bot$. Consequently, it will terminate.

We now turn to proving (2). To this end, assume some honest party terminates. This means that it received "done" messages from $n - t_c$ parties. Out of those, at least $n - 2t_c \geq t_c + 1$ are honest, so every honest party will receive $t_c + 1$ such messages and also send a "done" message. This means that every party will receive "done" messages from at least $n - t_c$ parties. Consider the first honest party to send a "done" message. It could not have done so as a result of receiving "done" messages from $t_c + 1$ different parties, because that would mean it received one of those messages from an honest party, contradicting the fact that it is the first to do so. This means that it sent the message after receiving "vote" messages from $n - t_c$ different parties. Out of those parties, at least $n - 2t_c \geq t_c + 1$ are honest. Before sending their "vote" messages, every one of those parties receives a "commits" broadcast with verifying proofs. Accordingly, it updates CM and S. Following that, they also receive a "row" message with a vector $\mathsf{row}_i = ((C_j(i), \pi_{j,i}))_{j \in [n]}$ with verifying proofs. Every one of those parties P_i sends a $\langle$"column"$, C_j(i), \pi_{j,i}\rangle$ to every P_j. Every honest P_j receives the same "commits" broadcast, sees that the proofs verify and updates CM and S. They also receive the "column" messages sent by the honest parties that sent "vote" messages, see that the proofs verify, and add a tuple to $\mathsf{points}_{col,j}$ each time. After adding $t_c + 1$ different tuples, P_j interpolates $\mathsf{points}_{col,j}$ and stores the result in C_i. In total, every honest party P_i eventually receives "done" messages from at least $n - t_c$ parties and has $S \neq \bot$ and $C_i \neq \bot$. At that point, every honest party terminates.

Finally, (3) follows directly from the fact that parties immediately terminate upon starting $\mathsf{HAVSS.Rec}$. This completes the proof. ☐

In order to prove that the protocol has the Oracle-Aided Simulatibility property, we first define a bad event for the Bad Event property, and show that there is a negligible property of it taking place.

Lemma C.8 *Assume that* $\mathsf{PS_{open}}$ *is a proof of knowledge and that the* DLOG *assumption holds relative to the values* $(\mathbb{G}, p, g)$ *used in the public parameters of the protocol. Then,* $\mathsf{HAVSS} = (\mathsf{HAVSS.Share}, \mathsf{HAVSS.Rec})$ *has the Bad Event property.*

Proof. First, we will define two separate sub-events as follows:

- $\mathsf{Bad_{Open}}$: This is the event in an execution of the real protocol in which an honest dealer sampled the polynomial S and some honest party P_i received a $\langle$"column"$, C_i(j), \pi_{i,j}\rangle$ message from a corrupt party P_j such that $C_i(j) \neq S(j, i)$ and $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}_i, j, C_i(j)), \pi_{i,j}) = 1$.

- $\mathsf{Bad_M}$: Define the parameters to be $(\mathbb{G}, p, g, g_0, \dots, g_{t_c})$ and let $\delta_0, \dots, \delta_{t_c} \in \mathbb{Z}_p$ be field elements such that $\forall i \in [\![t_c]\!]$ $g_i = g^{\delta_i}$. Observe the set of corrupted parties $\mathcal{C} \subseteq [n]$ at any time throughout the protocol and let $I \subseteq [\![n]\!]$ be the set of $t_c - |\mathcal{C}|$ minimal indices that aren't in $\mathcal{C}$. Let M be the matrix whose first row is $(\delta_0, \dots, \delta_{t_c})$ and the next t_c rows are Vandemonde rows $(j^0, \dots, j^{t_c})$ for $j \in I \cup \mathcal{C}$. We define 0^0 to be the value 1 in all of these computations. Define the event $\mathsf{Bad_M}$ as the event that at any time throughout the run, the matrix M is not invertible.

Finally, define Bad as $\mathsf{Bad_{Open}} \vee \mathsf{Bad_M}$ Let A be some PPT adversary for the protocol. Our goal is to bound the probability of the event Bad taking place. We will do so by bounding the probability of each one of the events taking place.

Bounding $\mathsf{Bad_{Open}}$. We will bound the probability of $\mathsf{Bad_{Open}}$ by constructing a reduction to Lemma C.3 that acts as follows:

1. The reduction gets as input parameters $(\mathbb{G}, p, g, g_0, \dots, g_{t_c})$.
2. The reduction runs the protocol by running the adversary and controlling all honest parties, instructing them to act honestly. It stores the polynomial S sampled by an honest dealer and all "column" messages received by honest parties. If it sees that at any point $\mathsf{Bad_{Open}}$ occurs by checking if at any point an honest party receives a verifying "column" message with an evaluation that is inconsistent with S. If that happens, some honest party received a $\langle$"column"$, C_i(j), \pi_{i,j}\rangle$ message from a corrupt party P_j such that $C_i(j) \neq S(j,i)$ and $\mathsf{PVer}^{\mathsf{H}}_{\mathsf{open}}((\mathsf{cm}_i, j, C_i(j)), \pi_{i,j}) = 1$. It then outputs $\mathsf{cm}_i, S(i,Y), (j, C_i(j), \pi_{i,j})$.

An honest dealer does compute the polynomial $S(i,Y) = \sum_{k=0}^{t_c} a_k Y^k$ and send the commitment $\mathsf{cm}_i = \prod_{k=0}^{t_c} g_k^{a_k}$. Therefore, if the event $\mathsf{Bad_{Open}}$ takes place, the reduction outputs a commitment cm_i, a corresponding polynomial $S(i,Y)$ and a verifying opening $(j, C_i(j), \pi_{i,j})$ such that $C_i(j) \neq S(i,j)$, meaning that it wins in the game described in Lemma C.3. As shown in Lemma C.3, the probability of this taking place is negligible and thus:

$$\Pr[\mathsf{Bad_{Open}}] \leq \mathsf{negl}(\lambda).$$

Bounding $\mathsf{Bad_M}$. We will bound the probability that $\mathsf{Bad_M}$ takes place by the following reduction to the DLOG assumption:

1. The reduction gets as input parameters $(\mathbb{G}, p, g, h)$ and attempts to find the discrete log of h with respect to g.
2. For every $i \in [\![t_c - 1]\!]$ the reduction samples a field element $\delta_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and defines $g_i = g^{\delta_i}$. In addition, it defines $g_{t_c} = h$.
3. The reduction runs the protocol with parameters $(\mathbb{G}, p, g, g_0, \dots, g_{t_c})$ by controlling all honest parties and running the adversary A.
4. Whenever a party is corrupted, the reduction adds its index to $\mathcal{C}$ In the beginning of the protocol and after each such update, the reduction defines $I \subseteq [\![n[\![$ to be the $t_c - |\mathcal{C}|$ minimal indices not in $\mathcal{C}$. Let V be a matrix with t_c Vandermonde rows $(j^0, \dots, j^{t_c})$ for different values $j \in I \cup \mathcal{C}$ and let its columns be $V^0, \dots, V^{t_c}$. The reduction finds the unique coefficients $a_0, \dots, a_{t_c - 1}$ such that $V^{t_c} = \sum_{k=0}^{t_c-1} a_k V^k$ by Gaussian elimination. It then computes $x = \sum_{k=0}^{t_c-1} a_k \delta_k$. If $h = g^x$, the reduction outputs x and otherwise it continues running the protocol.

The adversary's view is identical in the reduction and in a real execution, since the parameters are sampled identically. If the event $\mathsf{Bad_M}$ takes place, then after adding some index to $\mathcal{C}$, the matrix M was not invertible. The matrix has one row of the form $(\delta_0, \dots, \delta_{t_c-1}, x)$ where $h = g^x$, and the rest of the rows are the rows of V. Since the rows of V are Vandermonde rows, the matrix whose columns are $V^0, \dots, V^{t_c-1}$ is invertible. This means that the vectors $V^0, \dots, V^{t_c-1}$ are a basis for $\mathbb{Z}_p^{t_c}$, and thus there is a unique choice of coefficients $a_0, \dots, a_{t_c-1}$ such that $V^{t_c-1} = \sum_{k=0}^{t_c-1} a_k V^k$. The matrix M is not invertible if and only if this linear relationship holds for the first row as well. In other words, it is not invertible if and only if $x = \sum_{k=0}^{t_c-1} a_k \delta_k$. In this case, the reduction outputs x and succeeds in finding the discrete log of h. Therefore:

$$\Pr[\mathsf{Bad}_\mathsf{M}] \leq \mathsf{negl}(\lambda).$$

Combining these two results, we find that:

$$\Pr[\mathsf{Bad}] = \Pr[\mathsf{Bad}_\mathsf{Open} \vee \mathsf{Bad}_\mathsf{M}] \leq \Pr[\mathsf{Bad}_\mathsf{Open}] + \Pr[\mathsf{Bad}_\mathsf{M}] \leq \mathsf{negl}(\lambda).$$

□

Lemma C.9 *Assume that* $\mathsf{PS_{open}}$ *and* $\mathsf{PS_{exp}}$ *are zero-knowledge and that the* DLOG *assumption holds relative to the parameters of the protocol* $(\mathbb{G}, p, g)$*. Then, assuming secure erasures,* $\mathsf{HAVSS} = (\mathsf{HAVSS.Share}, \mathsf{HAVSS.Rec})$ *has the Oracle-Aided Simulatibility property.*

Proof. We will start by constructing a simulator Sim. The simulator receives an OMDL instance $g^{z_1}, \ldots, g^{z_{t_r+1}}$ of elements in $\mathbb{G}$. It starts by defining $\mathcal{C} = \emptyset$. It then uniformly samples discrete logs $(\delta_0, \ldots, \delta_{t_r}) \leftarrow\$ \mathbb{Z}_p^{t_r+1}$ and defines the parameters $g, g_0 = g^{\delta_0}, \ldots, g_{t_r} = g^{\delta_{t_r}}$. Sim then runs the protocol.

Honest dealer activation. When the dealer is first activated, if it is not corrupted, Sim sets $S_i = g^{z_i}$, for every $i \in [t_r + 1]$. Sim then interpolates the set $\{(i, S_i)\}_{i \in [t_r+1]}$ in the exponent and evaluates the exponentiated polynomial at 0 to get S_0 and at $t_r + 2, \ldots, n$ to get the values $S_{t_r+2}, \ldots, S_n$. Following that, Sim uniformly samples $t_r + 1$ values $(r_1, \ldots, r_{t_r+1}) \leftarrow\$ \mathbb{Z}_p^{t_r+1}$. Note that $\mathsf{ExpInterpolate}$ performs a constant linear operation in the exponent on $t_r + 1$ group elements to generate n group elements. Sim performs the same linear operation on $r_1, \ldots, r_{t_r+1}$ to generate n field elements $r_1, \ldots, r_n$ and computes $\mathsf{cm}_i = g^{r_i}$ for every $i \in [n]$. Finally, Sim simulates proofs π_i^{exp} for every $i \in [t_r+1]$ showing that $(\mathsf{cm}_i, 0, S_i)$ is in the relation $\mathcal{R}_\mathsf{exp}$. Sim adds messages to the queue as if it broadcasted a "commits" message with $\mathsf{CM} = (\mathsf{cm}_1, \ldots, \mathsf{cm}_{t_r+1})$ and $\mathsf{row}_0 = ((S_1, \pi_1^{\text{exp}}), \ldots, (S_{t_r+1}, \pi_{t_r+1}^{exp}))$, as well as "row" messages for all parties.

Honest party receiving a message. Whenever an honest party (i.e., a party not in $\mathcal{C}$) gets a message, Sim accepts it as a correct message if it was sent by a party that was honest at the time, or checks it if it was sent by a corrupt party. Sim then adds any messages that are sent as a result of processing messages. In more detail, whenever an honest party P_i receives a message from party P_j, Sim acts as follows:

- If the message is a "commits" message the dealer P_j, then if the dealer is honest, Sim considers P_i as having received a commitment CM. Otherwise, Sim checks that the message contains a commitment $\mathsf{CM}' = (\mathsf{cm}_1, \ldots, \mathsf{cm}_{t_r+1})$ and a row $\mathsf{row}'_0 = ((S_1, \pi_1^{exp}), \ldots, (S_{t_r+1}, \pi_{t_r+1}^{exp}))$ such that $\forall j \in [t_r + 1]$ $\mathsf{PVer}^\mathsf{H}_\mathsf{exp}((\mathsf{cm}_j, 0, S_j), \pi_j^{exp}) = 1$. If that is the case, Sim considers P_i as having received the commitment $\mathsf{CM} = \mathsf{ExpandCom}(\mathsf{CM}')$. Sim delays the execution of any following bullets for P_i until P_i has received a commitment.

- If the message is a "row" message from the dealer, then if the dealer is honest, Sim considers P_i as receiving a correct polynomial. If the dealer is corrupt, and the message contains a row $\mathsf{row}_i = ((C_i(1), \pi_{i,1}), \ldots, (C_i, \pi_{i,n}))$, then P_i is said to have received a correct polynomial only if for all $j \in [n]$, we have $\mathsf{PVer}^\mathsf{H}_\mathsf{open}((\mathsf{cm}_j, i, C_j(i)), \pi_{j,i}) = 1$. In both of these cases, if P_i received a correct polynomial, then Sim adds "column" and "vote" messages to the queue from i to all parties if it hasn't done so earlier.

- If the message is a "column" message with the values y_j, π_j from P_j, then if P_j is honest, Sim considers P_i as receiving a correct message from P_j. If P_j is corrupt, then P_i checks that $\mathsf{PVer}^\mathsf{H}_\mathsf{open}((\mathsf{cm}_j, i, y_j), \pi_j) = 1$ and if that is the case, it considers P_i as receiving a correct message from P_j. If the received message is the $t_c + 1$'th correct "column" message received by P_i, then Sim considers P_i as having interpolated C_i.

- If P_i received "vote" messages from $n - t_c$ different parties or "done" messages from $t_c + 1$ different parties, add "done" messages to the queue from i to all parties.

- If P_i received "done" messages from $n - t_c$ parties, and it has received a commitment from the dealer and interpolated a polynomial C_i, Sim considers it as having completed the $\mathsf{HAVSS.Share}$ protocol.

If at any time an honest party P_i is activated in the $\mathsf{HAVSS.Rec}$ protocol, Sim waits until it completes the $\mathsf{HAVSS.Share}$ protocol, and then immediately considers it as having completed the protocol.

Party corruption. Whenever a party P_i is corrupted, Sim starts by defining its polynomials C_i and R_i under the condition that C_i is also consistent with the commitment cm_i and that $C_i(j) = R_j(i)$ and $R_i(j) = C_j(i)$ for every corrupt P_j. First, it defines a polynomial C_i for P_i It does so by calling its discrete log oracle on S_i and set the response to be the value $R_0(i)$. The algebraic representations of S_i is the one defined by the linear transformation for interpolating and evaluating the S_i. Sim defines the set $I \subseteq [\![n]\!]$ to be the $t_c - |\mathcal{C}|$ minimal indices that aren't in $\mathcal{C}$. Note that 0 is always in I. For every $j \in I \setminus (\mathcal{C} \cup \{0\})$, Sim uniformly samples a value $R_j(i) \leftarrow\!\!\$\, \mathbb{Z}_p$. It then constructs a matrix M of dimensions $(t_c+1) \times (t_c+1)$ where the first row is the vector $(\delta_0, \ldots, \delta_{t_c})$ and the next t_c rows are Vandermonde rows of form $(j^0, j^1, \ldots, j^{t_c})$ for different values $j \in I \cup \mathcal{C}$. We define 0^0 to be 1 in all of these vectors. If the resulting matrix is not invertible, Sim aborts. Sim then defines the vector $v = (r_i, R_{j_1}(i), \ldots, R_{j_{t_c}}(i))^T$ with $j_1, \ldots, j_{t_c}$ being the indices of $I \cup \mathcal{C}$ in the same order as above. Finally, Sim computes the vector $(a_{i,0}, \ldots, a_{i,t_c})^T = M^{-1}v$ and defines $C_i(Y) = \sum_{k=0}^{t_c} a_{i,k} Y^k$. Note that by construction, $r_i = \sum_{k=0}^{t_c} \delta_k a_{i,k}$ and $C_i(j) = R_j(i)$ for every $j \in I$. This also means that $C_i(0) = R_0(i)$ and thus $S_i = g^{C_i(0)}$. Following that, Sim adds to i to $\mathcal{C}$ and turns to define $R_i(X)$. It does so by simply uniformly sampling a polynomial of degree t_r under the condition that for every $j \in \mathcal{C}$, $R_i(j) = C_j(i)$.

Sim then saves C_i and R_i. Note that there are at most t_c corruptions. Since I is defined before adding a newly corrupted party's index to $\mathcal{C}$, at that time $\mathcal{C}$ is of size $t_c - 1$ at most, and thus it is possible find a set I as defined above, and it always includes at least one index, namely 0. In addition, there is indeed at least one polynomial R_i of degree $t_r \geq t_c$ for which the required equalities hold.

Delivery of message to corrupt party. After sampling these polynomials, whenever Sim delivers a message to P_i from an honest party P_j, it does so in a consistent way with C_i and R_i. Similarly, before providing P_i's state to the adversary, it updates it as if it received similar messages from honest parties. In more detail whenever P_i receives a message from an honest P_j:

- If the message is a "commits" message from the dealer, then Sim delivers the message containing the commitments CM and the row row_0 defined above.
- If the message is a "row" message from the dealer, Sim delivers the message containing the row $\mathsf{row}_i = ((R_i(1), \pi_{i,1}), \ldots, (R_i(n), \pi_{i,n}))$, with $\pi_{i,j}$ being simulated proofs of the statement $(\mathsf{cm}_j, i, R_i(j))$ with respect to the relation $\mathcal{R}_{\mathsf{open}}$.
- If the message is a "column" message, then Sim delivers message $\langle$"column"$, C_i(j), \pi_{i,j}\rangle$ with $\pi_{i,j}$ being a simulated proof of the statement $(\mathsf{cm}_i, j, C_i(j))$ with respect to the relation $\mathcal{R}_{\mathsf{open}}$.
- If the message is a "vote" or a"done" message, Sim delivers it normally.

In addition, whenever such a corrupt party P_i receives a message from a corrupt P_j, Sim delivers that message to P_i.

Corrupted party's state. Before completing P_i's corruption, Sim constructs P_i's state by going through the messages P_i received, and computing the state it would have had after receiving those messages if it were honest. Note that by this we mean that Sim constructs its state as if it received the messages described above from honest parties and the actual messages sent by corrupt parties. Finally, after the first honest party completes the $\mathsf{HAVSS.Rec}$ protocol, Sim outputs $(S_0, S_1, \ldots, S_n)$ with the same algebraic representation described above for each R value.

Corruption of the dealer. If at any time the dealer is corrupted, Sim chooses a set of indices $I \subset [n]$ such that $I \cap \mathcal{C} = \emptyset$, $|I \cup \mathcal{C}| = t_r + 1$ (for example, the minimal indices for which this holds). It then calls the discrete log oracle on S_i for every $i \in I$. Following that, Sim generates polynomials C_i of degree t_c for every $i \in I$ in the same manner as the one described above. Finally, Sim defines S to be the unique bivariate polynomial of degree t_r in X and t_c in Y such that for every $i \in I \cup \mathcal{C}$ $S(i, Y) = C_i(Y)$. It then generates the state of all honest parties by simulating them receiving messages consistent with the dealer sampling the bivariate polynomial S by defining the polynomials $R_i(X) = S(X, i)$ and $C_i(Y) = S(i, Y)$. This can be done in the same way as generating the newly corrupted parties' state. Following that, whenever an honest party receives a message, it is processed as an honest party would in $\mathsf{HAVSS.Share}$, and whenever a corrupt party receives a message it is simply delivered. At the end of the simulation, Sim outputs $(S_0, \ldots, S_n)$.

Simply following the description above, it is clear that the Syntax, Dealer Corruption and Queries upon Corruption properties hold. In addition, only S_i elements are queries, which are generated by interpolating and evaluating the OMDL challenge. These are generated by by multiplying by a Vandermonde matrix and its inverse, which are both invertible. Therefore, the Query Independence property holds. As shown in Lemma C.8, the protocol has the Bad Event property. It is only left to show that the protocol has the Indistinguishability property. We proceed with that.

Indistinguishability. We will show a series of games that are indistinguishable from each other if the event Bad does not take place such that the first game is an execution of the protocol, and the last is the simulated execution of the protocol. Let A be some PPT adversary for the protocol.

Game $\mathbf{G}_0$: This game is a normal execution of the protocol given that the event Bad does not take place with the adversary A controlling the corrupt parties, and all other parties acting honestly.

Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_1$, except all parties abort if the event Bad ever takes place. As shown in the Bad Event property, the probability of the parties ever aborting is negligible. Further, this event is efficiently detectable. It is possible to check whether the event $\mathsf{Bad}_{\mathsf{Open}}$ took place by simply checking if at any point a verifying "column" message was received that is inconsistent with an honest dealer's sampled polynomial S. In addition, it is possible to check whether the event $\mathsf{Bad}_{\mathsf{M}}$ took place by computing the same Vandermonde rows described in the Bad Event property, computing the linear transformation from the first t_c columns of these rows to the final column, and then computing the same transformation in the exponent on $g_0, \ldots, g_{t_c-1}$. The matrix is not invertible if and only if the result is g_{t_c}, in which case all honest parties can abort.

Game $\mathbf{G}_2$: This game is identical to $\mathbf{G}_1$, all proofs of knowledge computed by honest parties are simulated instead of being computed honestly. $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable because of the zero-knowledge property of the proofs of knowledge.

Game $\mathbf{G}_3$: This game is identical to $\mathbf{G}_2$, except that if the dealer is honest, it sends $t_r + 1$ uniformly sampled commitments cm_i and uniform values S_i in its "commits" message. Additionally, whenever an honest party P_i gets corrupted, two polynomials C_i, R_i of degrees t_c and t_r are chosen for it uniformly under the condition that $C_i(j) = R_j(i)$ and $C_j(i) = R_i(j)$ for every corrupt P_j and under the condition that $S_i = g^{C_i(0)}$ and that $\mathsf{cm}_i = g^{\sum_{k=0}^{t_c} \delta_k a_k}$ where $C_i(Y) = \sum_{i=0}^{k} a_k Y^k$ and $g_k = g^{\delta_k}$ for every k. Then, whenever P_i receives a "row" message from the dealer, it contains the values $((R_i(j), \pi_{j,i}))_{j \in [n]}$ with $\pi_{j,i}$ being simulated proofs. Similarly, whenever it receives a "column" message from P_j, it contains the values $C_i(j), \pi_{i,j}$ with $\pi_{i,j}$ being a simulated proof. If at any point the dealer is corrupted, the same process takes place, but C_i polynomials are sampled for $t_r + 1 - |\mathcal{C}|$ additional parties as well. A polynomial S is defined to be the bivariate polynomial of degrees t_r in X and t_c in Y consistent with these C_i polynomials (i.e., $S(i, y) = C_i(Y)$ for every such i), and the dealer's view is generated as if it originally sampled S. In addition, all honest parties act as if messages that they received from the dealer are consistent with S, as well as messages from other honest parties. From this point on, honest parties continue acting as they would normally in the protocol.

First we will show that the adversary's view is identical in $\mathbf{G}_2$ and $\mathbf{G}_3$. In a both games the dealer does indeed uniformly sample a bivariate polynomial $S(X, Y)$ of degree t_r in X and t_c in Y. As long as the event Bad does not take place, all honest parties only receive points and proofs consistent with S, and send consistent messages to corrupt parties. This means that the corrupt parties' views consist of random values sent in the dealer's "commits" message, and of values on the polynomials $C_i(Y) = S(i, Y), R_i(X) = S(X, i)$. These are simply uniformly sampled polynomials which are consistent with all other C_j, R_j polynomials and with the broadcasted S_i values. That is, $C_i(j) = R_j(i)$ and $C_j(i) = R_i(j)$ and $S_i = g^{C_i(0)}$. Since the adversary only has access to the polynomials C_j, R_j held by corrupt parties P_j, all it sees are C_j and R_j polynomials consistent with each other and with the S_i values in both $\mathbf{G}_2$ and $\mathbf{G}_3$. If at any point the dealer is corrupted, then its view should be consistent with that same S, and all honest parties simply act normally under the condition that messages sent up until that point are consistent with S.

In order complete the proof, it is now left to argue that the view the adversary has in $\mathbf{G}_3$ is identical to the view it has while interacting with Sim. The "commits" message is indeed generated by using the uniform OMDL instance $g^{z_1}, \ldots, g^{z_{t_r+1}}$ as $S_1, \ldots, S_{z_{t_r+1}}$ and uniformly generating commitments. Following that, whenever a party P_i is corrupted, two polynomials C_i, R_i are defined for it. These polynomials are generated in a way consistent with S_i and with the other corrupted parties' C_j, R_j

polynomials All messages sent to P_i then contain values consistent with C_i and R_i and with simulated proofs. If the dealer is corrupted, then $t_r + 1 - |\mathcal{C}|$ polynomials C_i are defined. These polynomials are used to define S and the dealer's and honest parties' views are generated in a way consistent with S, after which they continue acting honestly. This means that it is enough to show that the distribution of the C_i and R_i polynomials in $\mathbf{G}_3$ is identical to the distribution while interacting with Sim. Note that the R_i polynomials are simply uniformly sampled by Sim under the condition that they are consistent with C_j for every $j \in \mathcal{C}$. This is the same distribution as the one in $\mathbf{G}_3$.

The proof for the distribution of the C_i polynomials is a little more intricate. Given that the event Bad does not take place, M is invertible at any point throughout the protocol. This means that M defines a bijection between coefficients $a = (a_{i,0}, \ldots, a_{i,t_c})^T$ and evaluations $v = (r_i, R_{j_1}(i), \ldots, R_{j_{t_c}})^T$ such that $Ma = v$ and $a = M^{-1}v$. By construction, r_i is the discrete log of cm_i. In addition, the set I always includes the index 0, and $R_0(i)$ is defined to be the discrete log of S_i. Finally, for every $j_k \in \mathcal{C}$, $R_{j_k}(i)$ is the evaluation of j_k's R polynomial at i. The rest of the $R_{j_k}(i)$ values are sampled uniformly. Therefore, because M^{-1} is a bijection, the coefficient vector a is sampled uniformly under the condition that the resulting polynomial $C_i(Y) = \sum_{k=0}^{t_c} a_{i,k} Y^k$ has $C_i(j) = R_j(i)$ for every $j \in \mathcal{C}$, $S_i = g^{C_i(0)}$, and $r_i = \sum_{k=0}^{t_c} \delta_k a_{i,k}$. In other words, the view generated when interacting with Sim is identical to the view in game $\mathbf{G}_3$. $\square$