Fully-Adaptive Two-Round Threshold Schnorr Signatures from DDH

Paul Gerhart, Davide Li Calsi, Luigi Russo, and Dominique Schröder

TU Wien, Vienna, Austria
{paul.gerhart, davide.li.calsi, luigi.russo, dominique.schroeder}@tuwien.ac.at

Abstract. Threshold Schnorr signatures enable t-out-of-n parties to collaboratively produce signatures that are indistinguishable from standard Schnorr signatures, ensuring compatibility with existing verification systems. While static-secure constructions are well understood and achieve optimal round complexity, obtaining *full adaptive security*—withstanding up to $t-1$ dynamic corruptions—under standard assumptions has proven elusive: Recent impossibility results (CRYPTO'25) either rule out known proof techniques for widely deployed schemes or require speculative assumptions and idealized models, while positive examples achieving full adaptivity from falsifiable assumptions incur higher round complexity (EUROCRYPT'25, CRYPTO'25).
We overcome these barriers with the first round-optimal threshold Schnorr signature scheme that, under a slightly relaxed security model, achieves full adaptive security from DDH in the random oracle model. Our model is relaxed in the sense that the adversary may adaptively corrupt parties at any time, but each signer must refresh part of their public key after a fixed number of signing queries. These updates are executed via lightweight, succinct, stateless tokens, preserving the aggregated signature format. Our construction is enabled by a new proof technique, *equivocal deterministic nonce derivation*, which may be of independent interest.

1 Introduction

Threshold signatures [Des88, DF90] are a fundamental primitive in decentralized systems, forming the backbone of many blockchain protocols and distributed applications. In a (t, n)-threshold signature scheme, any subset of at least t parties can jointly produce a valid multi-party signature, while smaller subsets learn nothing that would enable them to forge one. Originally introduced in the late 1980s [Des88, DF90, Ped92, Ped91], threshold signatures have seen renewed attention in recent years [KG20, BCK+22, CGRS23, CKM23, CKK+25a, CS25a, CKM25, BDLR25b, KRT24, DR24, BDLR25a, RRJ+22], largely due to their role in distributed trust infrastructures and cryptocurrency systems.

Among existing constructions, Schnorr-based [Sch90] threshold signatures are particularly appealing for their efficiency and compactness. An important advantage is that the resulting signatures are indistinguishable from standard Schnorr signatures, allowing verification by unmodified systems. This property ensures compatibility with platforms such as Bitcoin, which introduced Schnorr signatures in the Taproot upgrade [WNR21].

A Schnorr key pair consists of a secret key $\mathsf{sk} \in \mathbb{Z}_p$ and a public key $\mathsf{pk} = g^{\mathsf{sk}}$ in a cyclic group $\mathbb{G}$ of prime order p with generator g. To sign a message m, the signer samples a random value $r \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $R = g^r$, and computes the signature $\sigma' = (R, s)$, where

$$s = r + c \cdot \mathsf{sk}, \quad c = \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m).$$

Verification checks $g^s = R \cdot \mathsf{pk}^c$, and unforgeability follows from the discrete logarithm (DL) assumption in the random oracle model (ROM) [PS96]. Schnorr signatures are well-suited for *thresholdization*, as their linearity enables multiple parties to collaboratively compute signatures while preserving compactness and compatibility with standard formats.

Security guarantees for threshold signatures are typically categorized as *static* or *adaptive*. In the static setting, the adversary must choose which parties to corrupt before protocol execution [CKM23, CGRS23].

Adaptive security, in contrast, allows the adversary to select corruptions dynamically during execution [CKM23, BDLR25b], reflecting the needs of practical deployments. The strongest form, *fully adaptive* security, permits up to $t_c = t-1$ corruptions. Achieving this efficiently for Schnorr-based threshold signatures remains a major open challenge [CKM25, CS25a, CKK+25a].

Prominent static-secure examples of Schnorr-based schemes include FROST [KG20, BCK+22, CGRS23] and SPARKLE [CKM23], whose practical relevance has led to standardization efforts: the IETF's CFRG is formalizing FROST [CKGW24], and NIST's NIST IR 8214B [BD22] explicitly calls for threshold Schnorr signature schemes. The NIST call further stresses the importance of withstanding adaptive corruptions, crystallizing the central challenge: design threshold Schnorr protocols that

(1) remain verifiable by standard tools,
(2) achieve unforgeability against fully adaptive adversaries, and
(3) have low round-complexity.

1.1 The Challenge of Constructing Fully Adaptive, Round Optimal Schemes

One immediate question is whether known round-optimal schemes with static security can be extended to the fully adaptive setting or if fundamental limitations in current proof techniques and constructions prevent this. Despite substantial progress, recent results suggest that there are inherent barriers to achieving fully adaptive security for threshold signature schemes. Crites, Komlo, and Maller [CKM25] present a meta-reduction showing that fully adaptive security is impossible for threshold signature schemes with key-unique signing keys, meaning that each public key uniquely determines a corresponding secret key (e.g., $\mathsf{pk}_i = g^{\mathsf{sk}_i}$). Their impossibility result applies in two settings:

(1) any reduction to an underlying non-interactive assumption, and
(2) any reduction to the (algebraic) one-more discrete logarithm assumption based on rewinding, if $2 \cdot t_c \geq t$ (with t_c denoting the number of corrupted parties and t denoting the signing threshold).

As a consequence, fully adaptive security proofs using known techniques are ruled out for the most prominent constructions, including FROST [KG20, BCK+22, CGRS23] and SPARKLE [CKM23]. In a complementary line of work, Crites and Steward [CS25b], as well as Crites, Katz, Komlo, Tessaro, and Zhu [CKK+25b], show that for schemes with Shamir-secret-shared secret keys and exposing partial public keys of the form $pk_i = g^{sk_i}$, achieving full adaptive security requires the hardness of a specific low-dimensional vector representation (LDVR) problem. This problem is closely related to the so-called P-problem introduced in [CS25b]. They circumvent the impossibility of [CKM25] by doing a reduction without rewinding but instead based on the algebraic group model (AGM), an idealized model introduced by Fuchsbauer, Kiltz, and Loss [FKL18]. The authors show that if this new LDVR assumption is hard and the algebraic one-more DLog assumption holds, then FROST can be proven fully adaptively in the AGM. Furthermore, they show that the hardness of the LDVR assumption is necessary for proving fully adaptive security for the class of signature schemes whose partial public keys are formed as described earlier [CKK+25b]. Unfortunately, both the P-problem and the LDVR problem are non-natural, and their hardness is not fully understood by the community. For example, the hardness of LDVR is proven unconditionally for parameter sets $t_c < t/2$, but there remain parameters (including the fully-adaptive case) for which the hardness is unclear and must be assumed [CKK+25b].

The existence of positive examples offers hope for overcoming these limitations. In particular, there exist Schnorr threshold signature schemes whose security can be proven from falsifiable assumptions. Glacius [BDLR25b], KRT [KRT24], and Gargos [BDLR25a] avoid known impossibility results by using different key structures. They encode secret shares via Pedersen commitments [Ped92], which allow for multiple valid openings. This property renders the corresponding signing keys non-unique, thereby eliminating the key-uniqueness property exploited in existing attacks. Glacius and KRT achieve full adaptive security in five rounds, while Gargos reduces the round complexity to three.

Despite this progress, an important trade-off remains: achieving full adaptive security from falsifiable assumptions appears to require additional interaction, whereas optimal round complexity currently relies on highly speculative assumptions. Closing this gap, namely obtaining full adaptive security in the minimal

number of rounds *from falsifiable assumptions*, is not merely an incremental improvement but a key milestone for the theory and practice of Schnorr threshold signatures. We therefore pose the challenge:

> *Can we construct a fully adaptively secure threshold Schnorr signature scheme from falsifiable assumptions with minimal round complexity?*

To achieve this goal under standard assumptions and with minimal interaction, we slightly relax the standard adaptive security model. Our variant guarantees full adaptive security for up to a fixed number d of signing queries per period. Once this bound is reached, each signer performs a lightweight *update* that refreshes part of its partial public key using a succinct, stateless token that is gossiped to the other signers. The aggregated signature format remains unchanged, and the update tokens are of succinct size and can be efficiently computed. This represents a different functionality from the conventional ideal model for threshold signatures, but one that is operationally natural: periodic key updates are common practice in secure deployments [Ama, Goo, Mic, BPR22], and NIST's call for threshold cryptography [BD22] explicitly allows such adjustments, stating that *"It is acceptable that this [adaptive security] comes at the expense of some adjustment of the ideal functionality."*

1.2 Our Contributions

In this work, we answer the central open question in the affirmative. We present the first threshold Schnorr signature scheme that simultaneously attains full adaptive security from falsifiable assumptions and matches the optimal round complexity of the best-known static protocols. Our main contributions are as follows:

- **Fully adaptive security from DDH.** Our scheme achieves full adaptive security under the Decisional Diffie–Hellman (DDH) assumption for a fixed number of d signing queries.
- **Optimal two-round interaction.** The protocol requires only two rounds of interaction, matching the round complexity of the most efficient static constructions.
- **Drop-in compatibility.** The generated signatures are syntactically identical to standard Schnorr signatures, enabling seamless integration with existing verification algorithms and infrastructure.

We augment our construction with an *efficient update mechanism* that refreshes a designated portion of each signer's public key after every d signing queries, thereby extending security to fully adaptive adversaries for polynomially many messages. The update is *stateless*, requires only *logarithmic-size tokens* in d, and integrates cleanly into environments already supporting periodic key rotation. We emphasize that our mechanism for updating partial public keys is designed solely to extend adaptive security across many signing queries. It is *not* a proactive refresh protocol in the sense of [BPR24, HJJ+97, KGG24, CGG+20], and our scheme should not be interpreted as providing proactive security.

Additional Benefits. In addition to our main goal of achieving a fully adaptive Schnorr threshold signature from falsifiable assumptions in two rounds, our construction also inherits further desirable properties. These were not the focus of our design, yet our scheme attains them naturally and, to the best of our knowledge, no other fully adaptive Schnorr scheme offers the same guarantees. First, it supports *deterministic signing*, removing dependence on external randomness and making the signing process robust against both failures in randomness generation [HDWH12] and state-rewinding attacks [NRSW20]. Second, our protocol is *stateless*: beyond their secret signing key, signers maintain no secret state across or within signing sessions. Beyond their long-term keys, they need not store *any* additional secret information, which greatly simplifies deployment and eliminates common pitfalls such as synchronized counters or stateful nonce management. These properties have previously been highlighted as important in [NRSW20, GKMN21] though these Schnorr signature schemes do not achieve full adaptive security.

1.3 Related Work

There is extensive literature on threshold signatures across a variety of settings, including foundational work [DDFY94, GRJK00, Sho00], schemes for BLS-based signatures [Bol03], (EC)DSA [GGN16, GG18,

LN18, DKLs19, DOK+20, GKSŚ20, CGG+20, CCL+20, DJN+20, YCX21, ANO+22], and Schnorr signatures [GJKR96, GJKR07, GJKR03, KG20]. We already discussed (statically secure) FROST [KG20, BCK+22, CGRS23], SPARKLE [CKM23] and the recent impossibility results [CKM25, CKK+25b, CS25b] in the introduction, we restrict our focus to the most directly relevant prior work that is *not affected* by the recent impossibility results.

First, [CKK+25b] shows that FROST can be proven fully adaptively secure in the AGM under the LDVR and AOMDL assumptions. This yields the first two-round fully adaptive Schnorr threshold signature scheme. However, the proof relies on the speculative hardness of LDVR in the fully adaptive setting and requires the algebraic group model, adding another heuristic. While this is an important step in demonstrating the feasibility of two-round fully adaptive threshold Schnorr signatures, we view it primarily as a theoretical result. Our aim is to match this round complexity while relying only on standard assumptions. Second, the scheme proposed by Katsumata, Reichle, and Takemure [KRT24], uses one-time masking of signing transcripts [DKM+24] and achieves adaptive security under the discrete logarithm (DL) assumption. This approach supports full corruption by employing distinct polynomials across forking executions, thereby preventing key reuse across transcripts. However, to ensure soundness, all signers must maintain a globally consistent view, resulting in a five-round signing process. Third, Glacius relies on correlated random oracle programming and achieves adaptive security under DDH in five rounds [BDLR25b]. The subsequent Gargos [BDLR25a] uses a similar correlated random oracle programming technique, but compresses the round-complexity of Glacius to three rounds. We also build our construction based on correlated random oracle programming. To circumvent the need for a third round, we extend this technique with a novel proof strategy, which we call *equivocal deterministic nonce derivation*. Putting together both these proof strategies, we prove our two-round scheme fully-adaptively secure under DDH for up to d messages.

The deterministic Schnorr mutli-signature scheme MuSig-DN [NRSW20] also uses deterministic nonces to reduce round complexity of MuSig [MPSW19], but in an *inequivocable* way. This leads to a Schnorr multi-signature in two rounds, but it is not fully-adaptively secure. Their performance is comparable to our scheme instantiated with a signature-bound $d = 8192$ (c.f. Section 6), but their techniques are not applicable to fully-adaptive corruptions setting, since their inequivocable approach yields the commitment-problem.

2 Technical Overview

We start our overview with the *three-round*, fully adaptive threshold Schnorr signature scheme by Bacho *et al.* [BDLR25b, BDLR25a], which is based on correlated oracle programming [DR24, BDLR25b] (cf. Section 2.1). We outline the core idea of this technique to demonstrate why its commit-and-open structure poses significant challenges for a two-round adaptation. Based on these insights, we take a step back and propose a novel proof strategy based on *equivocal deterministic nonce derivation* in Section 2.2 that enables our final two-round, fully adaptive threshold Schnorr scheme, which we present in Section 2.3.

2.1 Fully Adaptive Threshold Schnorr Signatures in Three Rounds

We provide a brief overview of correlated random-oracle programming, which was first used by Das and Ren [DR24] to prove BLS signatures adaptively secure and was then used by Bacho *et al.* [BDLR25b, BDLR25a] in the context of three-round fully adaptive Schnorr. First, we recall the required modifications of the Schnorr signing protocol to enable full adaptive security using correlated random-oracle programming. Second, we discuss the key insights on why this technique inherently seems to require at least three rounds.

Switching Challenge Embedding. The core idea of correlated random-oracle programming is to relocate the embedding of the DL challenge during a security proof. Rather than embedding it in the aggregate verification key $\mathsf{pk} = g^{\mathsf{sk}}$ for a key $\mathsf{sk} = \sum \mathsf{sk}_i \in \mathbb{Z}_p$, each signer holds a triple

$$\mathsf{sk}_i = (x(i), w(i), u(i)),$$

where $x(X), w(X), u(X) \in \mathbb{Z}_p[X]$ are polynomials of degree $t+1$ with constant terms x, w, u, respectively. The aggregate key is $\mathsf{sk} = (x(X), w(X), u(X))$ with verification key

$$\mathsf{pk} = g^x \cdot h^w \cdot v^u,$$

where $w(0) = u(0) = 0$ and g, h, v are generators of $\mathbb{G}$. This looks similar to the standard construction in which $\mathsf{pk} = g^{\mathsf{sk}}$, but when doing a security proof, we allow the reduction to know the full description of $(x(X), w(X), u(X))$ while embedding the DL challenge in g and h. Since the reduction knows the full description of the polynomials, handling adaptive corruptions is rather easy: the reduction can simply forward key-shares (already known to the reduction) when the adversary corrupts a party (without relying on interactive assumptions as needed for FROST [CGRS23]). The remaining challenge is how to embed a DL instance between g and h when $w(0) = u(0) = 0$.

Rigging Keys. To encode the DL challenge into the public parameters, the reduction "rigs" the key by setting $w = 1$ (instead of $w = 0$) and sampling $u \leftarrow\!\!\$\ \mathbb{Z}_p$ (instead of $u = 0$). This transforms the verification key into

$$\mathsf{pk} = g^x \cdot h \cdot v^u = g^{x+\alpha_h+\alpha_v \cdot u},$$

where $h = g^{\alpha_h}$ and $v = g^{\alpha_v}$. The DL challenge can thus be encoded in h, while the reduction retains full knowledge of x and u. If the adversary produces a valid forgery under pk, the reduction can extract the *effective secret key*

$$\mathsf{sk} = x + \alpha_h + \alpha_v \cdot u,$$

via rewinding and recover α_h. Crucially, since x, w, and u are independently chosen, the reduction can simulate honest and corrupted shares sk_i without restriction, as the challenge lies solely in h. While this approach sidesteps the usual threshold bounds in rewinding-based proofs, it introduces a new challenge: the rigged key no longer satisfies the standard Schnorr verification equation.

Fixing the Schnorr Verification Equation. With rigged keys, a Schnorr signature $\sigma = (R, s)$ no longer satisfies the standard verification equation:

$$g^s = g^{r+x\cdot c} \quad \neq \quad g^{r+(x+\alpha_h+\alpha_v\cdot u)\cdot c} = g^{r+(x+\alpha)\cdot c},$$

where $c := \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$ and $\alpha := \alpha_h + \alpha_v \cdot u \neq 0$. To restore correctness, Bacho *et al.* [BDLR25b] modify the nonce to incorporate the extra key parts w and u setting

$$R \ \leftarrow \ g^r \cdot \mathsf{H}_0(\mathbf{d})^w \cdot \mathsf{H}_1(\mathbf{d})^u,$$

where $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$ are random oracles and $\mathbf{d}$ is a session identifier whose definition we specify later. In the non-rigged case ($w = u = 0$), this nonce construction yields to the standard $R = g^r$. In the rigged case, the reduction programs the oracles $\mathsf{H}_{\mathsf{Sig}}, \mathsf{H}_0, \mathsf{H}_1$ to enforce the verification equation. Concretely, it samples $c, \beta \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets

$$\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m) = c, \qquad \mathsf{H}_1(\mathbf{d}) = g^{\beta}, \qquad \mathsf{H}_0(\mathbf{d}) = g^{-(\beta u + c\alpha)}.$$

This correlated programming ensures that:

$$\gamma + \beta \cdot u + c \cdot \alpha = 0,$$

without requiring the discrete log of $h = g^{\alpha}$. Thus, signatures with the modified nonce verify correctly. By DDH hardness, such correlated programming is indistinguishable from uniform random oracle behavior [DR24, BDLR25b, BDLR25a]. Assuming that the $\mathbf{d}$ is chosen appropriately, this achieves full adaptive security for Schnorr signatures. However, the choice of d is central to efficiency, as will be discussed shortly.

Enabling Correlated Random-Oracle Programming. For the simulation to succeed, it is necessary that $\gamma + \beta \cdot u + c \cdot \alpha = 0$ holds in every signing session. This is enforced through a correlated programming of $\mathsf{H}_0(\mathbf{d})$, $\mathsf{H}_1(\mathbf{d})$, and $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$. A key challenge is that each triple (β, γ, c) is valid only for a single tuple (pk, R, m). Since m and pk are already known to the adversary in advance, the only remaining uncertainty lies in R. However, R is formed from the parties' shares, and a rushing adversary can choose its new shares after seeing those of the honest parties. Therefore, the reduction must know the combined R ahead of time in order to program the oracles consistently. Both Glacius [BDLR25b] and Gargos [BDLR25a] address this issue by employing a commit-and-open strategy.[1] In the first round, each party commits to their inputs. This reveals the adversary's choices to the reduction, allowing to program the random oracles while keeping the adversary unaware of the final value, R. In the second round, the parties reveal their commitments so that R can be computed. In the third round, each party publishes its partial signature on (pk, R, m) using its share sk_i.

2.2 A Two-Round Solution

The three-round protocol described above includes a commit-and-open step to ensure that the reduction learns the combined nonce R before the adversary does. This measure is necessary because a rushing adversary could otherwise select its nonce share R_i after observing those of the honest parties, thereby preventing consistent programming of the random oracles [BDLR25b].

Our goal is to remove this step and achieve round optimality. The core idea to avoid this extra round is that, if at the start of the signing process our reduction already knows each corrupted party's nonce share R_i, while the adversary still sees the combined nonce R as uniformly random, then we can program the random oracles immediately, making the commit-and-open phase unnecessary.

A straightforward solution for this idea would be an offline–online design in which each signer pre-commits to a sequence of nonces during a setup phase (e.g., via a vector commitment [CF13]) and later consumes these pre-computed nonces in the online phase. While conceptually simple, this approach has severe drawbacks for deployment: it forces the protocol to become *stateful*, which we generally want to avoid [GKMN21]. To ensure that corrupted parties use the correct pre-committed nonce, all signers would need to maintain synchronized counters and agree on which index has been consumed so far. Each signer would also need to update and persist local state reliably, which reintroduces the classic difficulties of state continuity (handling crashes, rollbacks, and synchronization across devices).

Therefore, we pursue another idea which allows us to proram the random oracles accordingly: We modify the signing protocol so that each R_i is computed deterministically (and also verifiably) from the public key pk, the message m, and the set of signers S. Once m and S are fixed (which they are at the beginning of the first signing round), the nonce shares of both honest and corrupted parties are uniquely determined, and our reduction can program the random oracle accordingly. As the deterministic nonce evaluation is also verifiable, even a rushing adversary is uniquely committed to its R_i for each signing session, enabling a two-round protocol.

The technique of having verifiable deterministic nonces is used in practice [NRSW20] for statically secure schemes. However, the main novel challenge for us is to generate these nonce shares verifiably from public inputs while retaining security against fully adaptive corruption. This is an instance of the *committing problem* [CFGN96], in which later corruptions require retroactive justification of earlier deterministic choices. While the commitment problem seems inherent for deterministic and verifiable schemes, we resolve this by deriving verifiable pseudorandom nonces in an *equivocal* manner.

Pseudorandom Nonce Derivation. Following the above intuition, we modify the signing protocol so that the combined nonce R is pseudorandom and depends on the message m and signer set S. Each signer i holds an additional secret key k_i for a PRF family $\{F_k\}$, and we set the input to the random oracles as $\mathbf{d} \leftarrow (m, S)$.

[1] Gargos uses three rounds, while Glacius has five; we describe a simplified three-round version sufficient for this overview.

Thus, while we keep the correlated-oracle-programming structure $R_i \leftarrow g^{r_i} \cdot \mathsf{H}_0(\mathbf{d})^{w_i} \cdot \mathsf{H}_1(\mathbf{d})^{u_i}$, with the crucial difference that r_i is no longer uniform but pseudorandom, i.e.,

$$r_i \leftarrow F_{k_i}(\mathbf{d}).$$

The combined nonce then takes the same form, but with pseudorandom combined value $r \leftarrow \sum_{i \in S} F_{k_i}(\mathbf{d})$. With overwhelming probability, R is unique for each pair (m, S), provided all partial nonces R_i are well-formed. Moreover, since R depends on pseudorandom outputs, the adversary can guess its value only with negligible probability. Together, these properties ensure that, except with negligible probability, the reduction knows R in advance and can program the random oracles consistently. Ensuring this well-formedness is therefore essential to eliminating the extra rounds.

Asserting Well-Formedness. We require a proof of well-formedness of the partial nonces R_i, since otherwise an adversary could simply deviate from the protocol and provide a nonce share not known to the reduction challenger. This would again be a rushing adversary and void our programming capabilities. As a natural starting point, we seek exponent-VRFs introduced by Nick *et al.* [NRSW20] and formalized by Boneh *et al.* [BHL24]. Exponent-VRFs are a bridge between PRFs and VRFs. They are linked to a public key $K = g^k$, and intuitively allow a user, on input a message $\hat{m}$, to compute

$$(\hat{R}, \hat{r}, \pi) \leftarrow \mathsf{eVRF}(k, m), \quad \text{where } \hat{r} \leftarrow \mathsf{PRF}(k, \hat{m}), \quad \text{and } (\hat{R}, \pi) \leftarrow \mathsf{VRF}(k, \hat{m}),$$

such that $\hat{R} = g^{\hat{r}}$. In addition, π verifies the correct computation of $\hat{r}$ using the key k, verifiable under K. Ideally, we would like to follow this strategy and use eVRFs to compute $(\hat{R}_i, \hat{r}_i)$ used as partial nonces for all signers, which would allow for verifiable deterministic nonces.

eVRFs and Adaptive Corruptions. Unfortunately, we cannot rely on verifiable deterministic nonces from eVRFs, since we aim for fully-adaptive security. While they enable correlated programming in two rounds—the basis of our construction—it is unclear how to deal with adaptive corruptions: At some point in the proof, we must replace the eVRF's pseudorandom outputs with uniformly random values, accompanied by simulated proofs, to incorporate the pseudorandomness of our construction into our security argument. This step is standard under static corruptions but fails in the adaptive setting. If the adversary corrupts a previously honest party *after* the PRF-to-random substitution, it can detect the inconsistency:

- Upon corruption, the challenger must reveal the party's PRF key.
- That key must retroactively justify all prior eVRF outputs.
- Since substituted values were not derived from the key, the adversary detects the mismatch—an instance of the *committing problem* [CFGN96] in fully adaptive settings.

This obstacle prevents us from applying the standard PRF-to-random substitution in the adaptive setting. To proceed, we therefore explore potential workarounds.

A Round-Optimal Solution Using Heuristics. As a starting point, we consider a workaround in the random oracle model. While this solution ultimately fails due to soundness issues, it provides useful intuition about the structure we aim to achieve and the building blocks we require. The main idea is to use the random oracle to derive the pseudorandom values r_i directly, and pair the provided nonces g^{r_i} with a proof of well-formedness. Doing so would create a useful gap between the reduction and the adversary: the adversary must output consistent elements r_j observable to the reduction, while the reduction can sample uniformly random elements and program the oracle so that for each honest party i,

$$\mathsf{RO}(k_i, m, S) = r_i$$

for a random key k_i. As long as the adversary does not know k_i, the values r_i appear uniformly random; after corruption, consistency still holds since the oracle was programmed accordingly. In this way, the reduction

avoids a PRF-to-random substitution entirely, as the values r_i are already uniform. Furthermore, the approach is round-optimal and circumvents the committing issue. However, it introduces a critical theoretical drawback since proof systems lose soundness when instantiated over a programmable random oracle. In particular, soundness would require knowledge of the oracle's full description, which is inherently exponential in the security parameter.[2]

Since our goal is a two-round, fully adaptive Schnorr threshold signature scheme from falsifiable assumptions, we cannot adopt this heuristic. Nevertheless, it suggests a path forward: if we can construct a random function with a succinct, polynomial-size description that can be made retroactively consistent with some key, we can resolve this tension and achieve our desired security guarantees.

Substituting the RO. To build our fully adaptive two-round threshold Schnorr scheme, we require a function that produces uniform-looking outputs on input $\mathbf{d}$, yet admits a succinct description and can be explained with some key k, after a party has seen a couple of evaluations of this function. A natural candidate is to use a random polynomial: A polynomial $f \in \mathbb{Z}_p[X]_{(d)}$ of degree d, with coefficients $a_0, \ldots, a_d$ sampled uniformly from $\mathbb{Z}_p$, has a succinct description of size $d+1$, and its outputs are fully deterministic. Yet the first $d+1$ evaluations appear indistinguishable from uniform: up to d input–output pairs, the constraints that have been employed by choosing d evaluation points can be satisfied by $|\mathbb{Z}_p|$ different polynomials of degree d, so the actual coefficients remain hidden. Given $d' \leq d+1$ elements $y \in \mathbb{Z}_p^{d'}$ representing f's evaluations at d' distinct points, finding f's coefficients $x \in \mathbb{Z}_p^{d'}$ amounts to solving a linear system $Ax = y$, where A is a suitable Vandermonde matrix. The said system is guaranteed to be underdetermined, nullifying the risk of inconsistencies. Moreover, by treating the coefficients as the key $k = (a_0, \ldots, a_d)$, the function provides exactly the properties we sought from the random oracle:

- Outputs are indistinguishable from uniform as long as the key remains hidden (up to $d+1$ evaluations).
- For a fixed key, outputs are fully determined.
- With simulated proofs and programmable commitments, the challenger can sample (up to $d+1$) uniform outputs and later find a consistent key upon corruption.
- The description of the function is of polynomial size, and we can build (efficient) proof systems that guarantee well-formed outputs.

Furthermore, the very nature of polynomials enables active signers to derive a random exponent in a *stateless* fashion. Specifically, leveraging polynomials for nonce derivation means that knowing only m and S is sufficient to evaluate f. This removes the need to maintain synchronized counters, as required by the aforementioned solution based on precomputed nonces. Using polynomials for randomness derivation, we can now put things together and describe our construction of a fully adaptive round-optimal Schnorr threshold signature scheme for up to d many adaptively-chosen messages. After describing this construction, we explain how to enhance the scheme to support a polynomial number of messages to sign.

2.3 A Fully-Adaptive Two-Round Schnorr Threshold Signature

We start the overview of our construction with key generation. As outlined above, each signer receives key shares $\mathsf{sk}_i = (x(i), w(i), u(i))$. In addition, each signer samples a uniformly random polynomial f_i of degree d and commits to this polynomial using a polynomial-commitment scheme resulting in $\mathsf{C}_{\mathsf{f}_i}$. We then enhance the partial public key of signer i by the polynomial commitment, setting it to $\mathsf{pk}_i = (g^{x(i)} h^{w(i)} v^{u(i)}, \mathsf{C}_{\mathsf{f}_i})$.

In signing round one, each signer sets $\mathbf{d} \leftarrow (m, S)$, where m is the message to sign and S consists of all partial public keys of the t signers participating in the signing request. Then, each signer computes:

$$R_i \leftarrow g^{r_i} \cdot \mathsf{H}_0(\mathbf{d})^{w_i} \cdot \mathsf{H}_1(\mathbf{d})^{u_i}, \quad \text{where } r_i \leftarrow \mathsf{f}_i(\mathbf{d})$$

[2] Also, it is worth noticing that any such system would incur a direct random oracle instantiation, roughly due to the *arithmetization* procedure of the relation, which additionally leads to practical issues as well as known vulnerabilities [CGH98].

Each signer then forwards R_i alongside a proof of well-formedness to all other signers. In signing round two, each signer verifies the provided shares R_j of all other signers using the partial public keys contained in S and computes the combined nonce:

$$R \leftarrow \prod_{j \in S} R_j,$$

The i-th signer then computes the challenge $c \leftarrow \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$, and outputs the partial signature:

$$s_i \leftarrow r_i + c\overline{x}_i \cdot \Lambda_i$$

where Λ_j is a Lagrange interpolation factor. Upon receiving all partial signatures, the final signature is computed as:

$$\sigma \leftarrow \left(R, \sum_{j \in S} s_j \right) = \left(g^r, x \cdot c + r \right),$$

which verifies against the public key pk using the standard Schnorr verification equation.

Extending the Signing Limitation. Using this protocol, we can support adaptive corruptions up to d signing queries. However, once an honest signer reaches the evaluation bound of d queries, it must refresh its randomness source. Although d can, in principle, be set arbitrarily large to suit practical needs, we observe that the efficiency of the resulting proofs–such as the prover's runtime, and in many cases the proof size or the verifier's runtime–is affected by d. Therefore, it is advantageous to keep d reasonably small to maintain practical efficiency.

To do so, we introduce an efficient update procedure. In this update procedure, each signer samples a fresh polynomial $\mathsf{f}'_i \in \mathbb{Z}_p[X]$ of degree d, commits to this fresh polynomial using a polynomial commitment resulting in $\mathsf{C}_{\mathsf{f}'_i}$, and gossips this fresh commitment to the other sigers. All other signers update the public key of signer i and switch the old for the fresh commitment. Having a fresh polynomial commitment distributed, the signer can now answer another d signing queries. We stress that the rest of the secret key (and consequently the public key) does not need to be changed.

Proving our Scheme Secure. To prove our scheme secure, we combine correlated random oracle programming with equivocable and verifiable nonce derivation (a.k.a, retroactively interpolating polynomials). During setup, the challenger samples the polynomials (x, w, u) as described above. For each signer, it additionally samples a uniformly random, perfectly hiding polynomial commitment and outputs the corresponding partial public key:

$$\mathsf{pk}_i = (g^{x(i)} h^{w(i)} v^{u(i)}, \mathsf{C}_{\mathsf{f}_i}).$$

When a signer j is corrupted, the challenger interpolates a uniformly random polynomial f_j consistent with the input-output pairs recorded for that signer so far and adjusts the commitment to match this polynomial. It then reveals the signing key $\mathsf{sk}_j = (x(j), w(j), u(j))$ and a description of the polynomial f_j. This ensures that the signer's internal state is consistent with all its previously answered signing queries. To simulate signing queries, the challenger samples uniformly random elements as the polynomial outputs, simulates the corresponding proof of nonce well-formedness, and stores the resulting input-output pair for the yet-to-interpolate polynomial f_i. Importantly, the challenger knows the full nonce R_j of each corrupted party at the beginning of round one, since these nonces are deterministically derived from the known polynomial f_j and accompanied by simulated proofs of well-formedness. This allows the challenger to apply the correlated oracle programming strategy described earlier. Our strategy so far works for up to d signing queries. When this bound is reached, the challenger refreshes each honest signer's commitment by generating a new perfectly hiding uniformly random polynomial commitment. When the adversary performs a commitment update, the challenger uses a knowledge extractor to obtain the new polynomial description.[3] This enables the challenger

[3] One may think that, in order to be extractable, the proof should be at least as large as the witness itself, which would make the construction inefficient; however, we only need this extractability feature in the security reduction, where we can leverage rewinding-based extractors, and thus these proofs can be instantiated efficiently.

to compute the adversary's future nonces at signing round one even after commitment updates. Finally, note that this knowledge extraction does not interfere with signing query simulation: our protocol runs in two rounds, and the second round aborts if the random nonces cannot be verified against the provided public keys. However, in principle, nothing prevents an adversary from mounting *cross-epoch* attacks by reusing proofs and commitments of previous epochs. To address this subtle issue, we require the knowledge-extractor to successfully extract the witness even in presence of simulations that, looking ahead, are provided by the reduction itself; this is why we rely on a stronger property, known as *simulation extractability* (see Section 4.1 for a detailed discussion).

Eventually, when the adversary outputs a valid forgery, the reduction rewinds it to extract two valid signatures for the same random commitment. This enables recovery of the secret exponent and solves the discrete logarithm challenge. Additionally, under the DDH assumption, the adversary cannot distinguish our programming of the three random oracles used in the protocol, following the simulation techniques from [BDLR25b, BDLR25a].

Applicability. Importantly, our verifiable nonce approach modifies only the structure of partial signatures, without affecting the final signature. Once the partial signatures are aggregated, verification proceeds as in the standard Schnorr scheme. The overall verification key also remains unchanged, since the added elements in the partial public keys are used solely for verifying nonce well-formedness and do not impact the combined key.

Each partial public key now includes a constant-size polynomial commitment, increasing its size from one to two group elements. On the secret side, each signer's key also grows slightly, as it must include the coefficients of the randomness-deriving polynomial in addition to the usual shares $(x(i), w(i), u(i))$. However, rather than storing all coefficients explicitly, each signer can derive them on the fly from the random oracle via $a_{i,j} = \mathsf{RO}(k_{\mathsf{RO},i}, j)$, so that the secret key needs to be extended only by a single scalar $k_{\mathsf{RO},i} \in \mathbb{Z}_p$, as similarly done in [BDK+18]. This optimized derivation is fully compatible with our equivocability strategy, since we can program the random oracle as required, and it preserves verifiability, as proofs only concern the values $a_{i,j}$ rather than their method of generation. We benchmark this method of on-the-fly polynomial evaluation in Section 6.

The main computation and communication overhead comes from computing and sending a proof of well-formedness for each partial nonce. To keep the proof system transparent and avoid additional assumptions, we can leverage a (commit-and-prove) simulation-extractable scheme based on Bulletproofs [BBB+18], which yields logarithmic proof size in the polynomial degree d under the discrete logarithm assumption. We implemented and evaluated the Bulletproof-based nonce derivation, and observe that our scheme has comparable performance as the deterministic (non-adaptive) Schnorr multisignature MuSig-DN [NRSW20] if we use a equivocability bound of $d = 8192$ (c.f. Section 6). For smaller bounds, our protocol is much more efficient.

Alternatively, if constant-size communication is desired, one could use proof systems such as Groth16 [Gro16], Plonk [GWC19], Marlin [CHM+20], and Lunar [CFF+21] among the others, at the expense of introducing stronger assumptions. We discuss the choice of a suitable proof system in Section 4.1.

3 Preliminaries

Notation and Group Description. We denote by $x \leftarrow y$ the assignment of value y to variable x, and we denote by $x \leftarrow\!\!\$\ X$ the uniform sampling of x from the set X. We utilize the symbol $\mathbb{G} := (\mathbb{G}, p, g)$ to denote a group description, where $\mathbb{G}$ is a cyclic group of order p, where p is a λ-bit prime, and g is a generator of $\mathbb{G}$. Given an element $X \in \mathbb{G}$, we let $\log_g(X)$ denote the discrete logarithm of X with base g, i.e., $\log_g(X)$ is the value $x \in \mathbb{Z}_p$, such that $X = g^x$. We use $\mathbb{Z}_p[Z]_{(t)}$ to denote the set of all polynomials in $\mathbb{Z}_p[Z]$ of degree t, and $\mathbb{Z}_p[Z]_{\leq t}$ to denote all polynomials of degree *at most* t. Both notations are straightforwardly extended to an arbitrary finite field $\mathbb{F}$ by replacing $\mathbb{Z}_p$ with the latter.

3.1 Non-interactive Arguments

A *non-interactive argument system* (NARG) for relation $\mathcal{R}$ in the random oracle model, denoted by Π, consists of a tuple of algorithms $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ having black-box access to a random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, with the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: takes as input the security parameter 1^λ and outputs public parameters pp.
- $\pi \leftarrow \mathcal{P}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$: takes as input parameters pp, a statement $\mathbb{x}$ and witness $\mathbb{w}$, and outputs a proof π if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi)$: takes as input parameters pp, a statement $\mathbb{x}$ and proof π, and it accepts ($b = 1$) or rejects ($b = 0$).

In this work, we consider NARGs with *transparent* setup, i.e. pp can be generated with a call to the random oracle. For this reason, we may omit pp in the description of the prover and verifier algorithms as it is implicit.

Definition 1 (Completeness). *A NARG Π satisfies* completeness *if for every $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, it holds that*

$$\Pr\left[b = 1 : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathcal{P}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \mathbb{w}); b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi)\right] = 1.$$

Definition 2 (Soundness). *A NARG Π satisfies* soundness *if for every PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[b = 1 \wedge \mathbb{x} \notin \mathcal{L}_{\mathcal{R}} : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); (\mathbb{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{pp}); b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathsf{pp}, \mathbb{x}, \pi)\right]$$

where $\mathcal{L}_{\mathcal{R}} \coloneqq \{\mathbb{x} \mid \exists \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$ is the set of true-statements.

If soundness holds with respect to a computationally unbounded adversary, we say that the argument has *statistical* (or *information-theoretic*) soundness, and we call the argument a *proof* system.

Zero-knowledge. Informally, an argument is zero-knowledge if a proof reveals no information about the witness [GMR85]. We formalize this property by following the syntax of [FKMV12]. A zero-knowledge simulator $\mathcal{S}$ is defined as a stateful algorithm, whose initial state $\mathsf{st} = \mathsf{pp}$, that operates in two modes. The first mode, $(y, \mathsf{st}') \leftarrow \mathcal{S}(1, \mathsf{st}, a)$ takes care of handling calls to the oracle H on input a query a. The second mode, $(\pi, \mathsf{st}') \leftarrow \mathcal{S}(2, \mathsf{st}, \mathbb{x})$ simulates a proof for the input statement $\mathbb{x}$. We define the following "wrapper" oracles, that are stateful and share their internal state:

- $\mathcal{S}_1(a)$ returns the first output of $\mathcal{S}(1, \mathsf{st}, a)$;
- $\mathcal{S}_2(\mathbb{x}, \mathbb{w})$ returns the first output of $\mathcal{S}(2, \mathsf{st}, \mathbb{x})$ if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and $\perp$ otherwise;
- $\mathcal{S}'_2(\mathbb{x})$ returns the first output of $\mathcal{S}(2, \mathsf{st}, \mathbb{x})$.

Definition 3 (Zero-knowledge). *A NARG Π is zero-knowledge if there exists a simulator $\mathcal{S}$, with wrapper oracles $\mathcal{S}_1, \mathcal{S}_2$, such that for all PPT adversaries $\mathcal{A}$ it holds that:*

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathcal{A}^{\mathcal{P},\mathsf{H}}(\mathsf{pp}) = 1\end{matrix}\right] \approx \Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathcal{A}^{\mathcal{S}_1,\mathcal{S}_2}(\mathsf{pp}) = 1\end{matrix}\right]$$

Notice that zero-knowledge is a security property that is only guaranteed for valid statements in the language, hence the above definition uses $\mathcal{S}_2$ as a proof simulation oracle. Also, a zero-knowledge NARG with statistical soundness is simply referred to as a non-interactive zero-knowledge proof (NIZK) [BFM88].

Simulation exctractability. A strong security property of NARGs is *simulation extractability* [DDO+01, GM17]. Informally, from a prover producing a valid proof after seeing polynomially many proofs provided by a simulator (even for *false* statements), it is possible to extract a valid witness. This property can be formalized in many different ways, depending on the model and the constraints of the adversary and the extractor. We refer to [BCC+25] for a recent work that summarizes the main differences.

Definition 4 (Simulation extractability). *A NARG Π is simulation-extractable with respect to a zero-knowledge simulator $\mathcal{S}$, with wrapper oracles $\mathcal{S}_1, \mathcal{S}_2'$, if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that:*

$$\Pr\left[\begin{array}{l|l} \mathcal{V}^{\mathcal{S}_1}(\mathbb{x}, \pi) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge\ (\mathbb{x}, \pi) \notin \mathcal{Q} & (\mathbb{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}_2'}(\mathsf{pp}) \\ \wedge\ (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \mathbb{w} \leftarrow \mathcal{E}(\mathsf{st}, \mathbb{x}, \pi) \end{array}\right] \in \mathsf{negl}(\lambda)$$

where st is the final internal state of $\mathcal{S}$, and $\mathcal{Q}$ is the set of statement-proof pairs obtained by $\mathcal{A}$ when interacting with $\mathcal{S}_2'$.

The previous definition implies the weaker notion of *simulation soundness* [Sah99]. Informally, a NARG is simulation-sound if it is infeasible to generate a valid proof π for a false statement $\mathbb{x}$, even after seeing polynomially many simulated proofs. The implication holds because Definition 4 imposes a negligible probability of extraction failure, and extraction is by definition impossible for a false statement.

Discrete Logarithm and Decisional Diffie-Hellmann Assumptions. The Discrete Logarithm (abbreviated as DL) assumption is the conjectured computational hardness of evaluating discrete logarithms in some groups. For the sake of rigour, we say that the DL assumption holds in group $\mathbb{G}$ defined as above, if any PPT adversary $\mathcal{A}$ has a negligible advantage $\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{A}}(\lambda)$ in the security game $\mathsf{DL}(\mathbb{G}, p, g)$ of Figure 1.
The Decisional Diffie-Hellmann (abbreviated as DDH) assumption is a related computational assumption. From a high-level viewpoint, it states that given uniformly random $(X, Y) \sim \mathcal{U}(\mathbb{Z}_p^2)$ and $Z \sim \mathcal{U}(\mathbb{Z}_p)$ ($\mathcal{U}(\cdot)$ denotes the uniform distribution over its argument), the distributions (g, g^X, g^X, g^{XY}) and (g, g^X, g^Y, Z) are computationally indistinguishable. The statement can be made rigorous by defining a security game in Figure 1 and assuming that any PPT adversary $\mathcal{A}$ has negligible advantage $\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{A}}(\lambda)$ when playing it.
The DDH assumption implies the hardness of DL, since an adversary computing DL in polynomial time can distinguish the two DDH distributions with few additional operations (compute $\log_g$ of the last element, check if it equals XY modulo p). The converse is not known to hold, and there exist groups in which DDH does not hold, but DL is believed to. Hence, DDH is, to the best of our knowledge, a stronger assumption than DL.

Game $\mathsf{DL}(\mathbb{G}, p, g)$		Game $\mathsf{DDH}(\mathbb{G}, p, g)$	
1 :	$X \leftarrow\!\!\$\ \mathbb{Z}_p$	1 :	$X, Y, Z \leftarrow\!\!\$\ \mathbb{Z}_p$
2 :	$x \leftarrow \mathcal{A}(g^X)$	2 :	$b \leftarrow\!\!\$\ \{0, 1\}$
3 :	**return** $g^x == g^X$	3 :	$b' \leftarrow \mathcal{A}(g, g^X, g^Y, g^{(1-b)Z + bXY})$
		4 :	**return** $b == b'$

Fig. 1. Security games for DL and DDH.

Commitment Schemes. A commitment scheme with message space $\mathcal{M}$ is a tuple of algorithms $\mathsf{CS} := (\mathsf{KGen}, \mathsf{Com}, \mathsf{VerCom})$ with the following syntax:

- $\mathsf{ck} \leftarrow \mathsf{KGen}(1^\lambda)$: takes as input the security parameter (and possibly group parameters) and outputs a commitment key ck.
- $(\mathsf{C}, o) \leftarrow \mathsf{Com}(\mathsf{ck}, m; \omega)$: takes as input the commitment key ck, a message $m \in \mathcal{M}$, and randomness ω, and outputs a commitment C and an opening o.
- $b \leftarrow \mathsf{VerCom}(\mathsf{ck}, \mathsf{C}, m, o)$: takes as input the commitment key ck, a commitment C, a message $m \in \mathcal{M}$, an opening o, and it accepts ($b = 1$) or rejects ($b = 0$).

We say that a commitment scheme whose message space is $\mathbb{F}[X]_{\leq d}$, for a degree parameter d given as additional input to KGen, is a *polynomial commitment scheme*. Besides correctness , a commitment scheme CS satisfies two more properties.

Definition 5 (Binding). *A commitment scheme* CS *is (computationally) binding if no PPT adversary can find, unless with negligible probability, a commitment* C*, two messages* $m \neq m'$ *and two openings* o, o'*:*

$$\mathsf{VerCom}(\mathsf{ck}, \mathsf{C}, m, o) = \mathsf{VerCom}(\mathsf{ck}, \mathsf{C}, m', o') = 1$$

Definition 6 (Hiding). *A commitment scheme* CS *is (computationally) hiding if* $\forall m, m'$*,* $\forall \mathsf{ck}$*:*

$$\{\mathsf{C} : (\mathsf{C}, o) \leftarrow \mathsf{Com}(\mathsf{ck}, m)\} \approx \{\mathsf{C}' : (\mathsf{C}', o') \leftarrow \mathsf{Com}(\mathsf{ck}, m')\}$$

In contrast, perfect hiding/binding strengthens this guarantee by requiring security even against unbounded adversaries.

Trapdoor Commitment Schemes. In some applications, one further requires the trapdoor property for commitment schemes [GKS24]. That is, we require the existence of three additional PPT algorithms ($\mathsf{tdKeyGen}$, $\mathsf{tdComm}, \mathsf{tdOpen}$). The first outputs $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{tdKeyGen}(1^\lambda)$, where ck is a commitment key indistinguishable from an honestly generated key, and td is a commitment trapdoor. The second algorithm outputs $(\mathsf{C}, \mathsf{st}) \leftarrow \mathsf{tdComm}(\mathsf{td})$, where C is a commitment and st is a piece of side information. The third outputs a commitment opening $o \leftarrow \mathsf{tdOpen}(\mathsf{C}, m, \mathsf{st}, \mathsf{td})$. The requirement is that o can be used to open C to value m without triggering a rejection, for any message m of our choice. A trapdoor commitment is said to be ϵ-equivocable if the statistical distance between a honestly generated commitment key, commitment, and opening, and an analogous triple generated by $(\mathsf{tdKeyGen}, \mathsf{tdComm}, \mathsf{tdOpen})$ is at most ϵ. We refer to [GKS24] for a formal definition.

Pedersen Commitment Scheme. We mostly rely on the Pedersen commitment scheme [Ped92] with message space $\mathbb{Z}_p^n$, for some $n \in \mathbb{N}$, over some group $\mathbb{G}$, that is computationally binding under the DL assumption and perfectly hiding, and works as follows:

- $\mathsf{KGen}(1^\lambda)$ outputs $n+1$ random generators $g_1, \ldots, g_n, h$ of $\mathbb{G}$.
- $\mathsf{Com}(\mathsf{ck}, \boldsymbol{a}; \omega)$ parses ck as $(g_1, \ldots, g_n, h)$ and outputs the commitment $\mathsf{C} := \prod_{i \in [n]} g_i^{a_i} h^\omega$ and the opening ω.
- $\mathsf{VerCom}(\mathsf{ck}, \mathsf{C}, \boldsymbol{a}, o)$ outputs 1 if and only if $\mathsf{Com}(\mathsf{ck}, \boldsymbol{a}; o) = \mathsf{C}$.

The Pedersen commitment can be effectively used as a polynomial commitment scheme. In particular, we define $\mathsf{Com}(\mathsf{ck}, f(X)) := \mathsf{Com}(\mathsf{ck}, (f_i)_{i \in [0,d]})$ as the commitment to (the coefficients of) a univariate polynomial $f(X) := \sum_{i=0}^{d} f_i X^i$ of degree d. In addition, the Pedersen commitment scheme is a is perfectly equivocable (i.e., with factor $\epsilon = 0$) trapdoor commitment scheme if endowed with the following extra algorithms:

- $\mathsf{tdKeyGen}(1^\lambda)$: sample $e_1, \ldots, e_n, \rho \leftarrow\!\!\$\ \mathbb{Z}_p$. Set $g_i = g^{e_i}$ for all $i \in [n]$, and $h = g^\rho$. Output $\mathsf{ck} = g_1, \ldots, g_n, h$ and $\mathsf{td} = e_1, \ldots, e_n, \rho$.
- $\mathsf{tdComm}(\mathsf{td})$: sample $r \leftarrow\!\!\$\ \mathbb{Z}_p$. Output (g^r, r).
- $\mathsf{tdOpen}(\mathsf{C}, \boldsymbol{a}, \mathsf{td})$: solve the linear equation $\rho\omega + \sum_{i=1}^{n} e_i a_i = r$. Return ω.

3.2 Threshold Signatures

A threshold signature scheme enables a group of n signers to collaboratively generate a signature on a given message m. The scheme is parameterized by a threshold $t < n$ that dictates the minimum number of signers required to jointly produce a valid signature. Another relevant parameter is the number of interaction rounds k, which directly impacts the scheme's efficiency. We follow the definition from [BDLR25b] but for the case $k = 2$, as it aligns with our scheme, though the definition extends naturally to arbitrary k.

Formally, a 2-round threshold signature scheme is a tuple of algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SR}_1, \mathsf{SR}_2, \mathsf{SignAgg}, \mathsf{Verify})$ such that:

- $\mathsf{pp} \leftarrow \mathsf{Setup}$ reads a pair (t, n) and outputs public parameters pp. The latter are used throughout the rest of the protocol and, unless otherwise specified, should be implicitly assumed to be an input to all subsequent algorithms.
- $(\mathsf{pk}, \{(\mathsf{sk}_i, \mathsf{pk}_i)\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ is the key generation algorithm. It outputs a global public key pk, n public key shares $\{\mathsf{pk}_i\}_{i\in[n]}$ and n secret signing key shares $\{\mathsf{sk}_i\}_{i\in[n]}$.
- $\mathsf{pm}_{1,i} \leftarrow \mathsf{SR}_1(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}, S, i, m)$ is the first signing round. It reads a signing key share sk_i, public key share pk_i, public key pk, signing set S, index $i \in [n]$, and message m. The output is a protocol message for the i-th signer $\mathsf{pm}_{1,i}$.
- $s_i \leftarrow \mathsf{SR}_2(\mathsf{sk}_i, \mathsf{pk}, S, i, \{\mathsf{pm}_{1,i}\}_{i\in S}, m)$ is the second signing round. It reads a signing key share sk_i, public key pk, signing set S, index $i \in [n]$, the set of all previous-round messages $\{\mathsf{pm}_{1,i}\}_{i\in S}$, and message m. The output is a signature share s_i.
- $\sigma \leftarrow \mathsf{SignAgg}(\{s_i\}_{i\in S})$ is the aggregation phase, whose goal is to combine partial signatures into a unique global signature. It reads partial signatures $\{s_i\}_{i\in S}$ and outputs global signature σ for message m.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}, m, \sigma)$ is the verification procedure. It reads a public key pk, a message m, and a signature σ. It then outputs a binary flag $b \in \{0, 1\}$, indicating the signature is invalid or valid, respectively.

A threshold signature scheme provides *correctness* if signatures generated by an honest signing coalition of size at least t are accepted by Verify, as described in the following game:

1. Generate pp and all keys honestly.
2. Read a signing set S and a message m.
3. Execute all phases honestly, obtaining signature σ.
4. Output $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$.

We require that for any n, $t < n$, set $S \subseteq [n]$ with $|S| \geq t$, the output is 1 with overwhelming probability (in the security parameter).

Unforgeability. Several threat models for threshold signatures have been considered in the literature. In this work, we assume an *adaptive* adversary $\mathcal{A}$ who can dynamically choose to corrupt new parties [CKM23, BDLR25b]. In addition, $\mathcal{A}$ may start multiple signing sessions in parallel. As usual when dealing with signatures, $\mathcal{A}$'s goal is (informally) to generate a new pair (m', σ') such that m' is a fresh message and on input (m', σ') the output of Verify is 1. In this context, "fresh" should be interpreted as not in the set of signing queries submitted by $\mathcal{A}$. The formal definition of unforgeability against adaptive adversaries is more cumbersome and requires a dedicated security game. We adapt the definition of [CKK+25b] to our purposes. Figure 2 provides a full description of all the involved oracles and procedures. In our security game, $\mathcal{A}$ is given access to three oracles:

- A corruption oracle OCorr. On input $i \in [n]$, the oracle corrupts the i-th signer. This instantly lets $\mathcal{A}$ read its private information and control its future actions. To comply with the threshold requirement, $\mathcal{A}$ is allowed to submit at most $t - 1$ corruption requests.
- A signing round oracle OSR_1. Upon receiving a call from $\mathcal{A}$, it executes the first signature round. $\mathcal{A}$ must specify some core parameters, such as the desired session, round, and *honest* signer to interact with. The oracles returns the corresponding first round message $\mathsf{pm}_{1,i}$ to $\mathcal{A}$.
- Another signing round oracle OSR_2, which executes the second signing round upon receiving a request from $\mathcal{A}$. The oracle returns a partial signature s_i to $\mathcal{A}$.

$\mathcal{A}$ submits as many queries as it wants to its oracles, and finally outputs a pair (m', σ'). The game models existential unforgeability under a chosen-message attack, so its outcome is defined as $\mathsf{Verify}(\mathsf{pk}, m', \sigma') \wedge (m' \notin Q)$ with $\wedge$ denoting boolean AND, and $(m' \notin Q)$ denoting a boolean predicate for set membership. As customary, we call a scheme unforgeable if for any PPT adversary $\mathcal{A}$ and any $(n, t) \in \mathbb{N}$ with $t < n$

$$\mathsf{Adv}^{\mathsf{tsauf}}_{\mathsf{TS},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda).$$

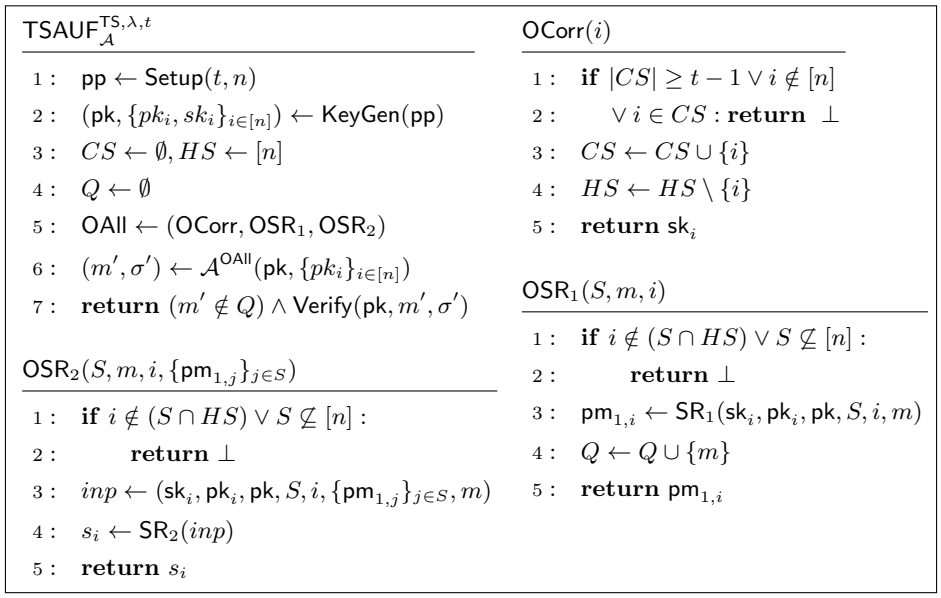

Fig. 2. The adaptive unforgeability security game.

Proactively Refreshable Threshold Signatures. The previous unforgeability definition is arguably the most traditional one. In this work, we put forth a slightly different notion of unforgeability for threshold signatures, in which periodic key refreshments are possible. Intuitively, we perform period key refreshments that update partial public and secret keys, and ensure that the scheme remains unforgeable. We follow the approach of [BPR22], but adapt it to our setting. We say a threshold signature scheme supports proactive key refresh with d messages if (in addition to the algorithms already provided by the threshold signature scheme, and an updated Setup that takes d as additional input) there exist two algorithms, $(\mathsf{Update}_{\mathsf{sk}}, \mathsf{Update}_{\mathsf{pk}})$, such that:

- $(\mathsf{sk}'_i, \tau_i) \leftarrow \mathsf{Update}_{\mathsf{sk}}(\mathsf{sk}_i)$ reads the i-th partial secret key sk_i, and outputs an updated secret key sk'_i as well as an update token τ_i. sk'_i replaces sk_i as the secret key of signer i. τ_i is then shared with all the other signers. Each signer must run it once at the end of each epoch.
- $\mathsf{pk}'_j \leftarrow \mathsf{Update}_{\mathsf{pk}}(\mathsf{pk}_j, \tau_j)$ reads the j-the partial public key pk_j and an update token τ_j. It verifies its validity, and eventually updates pk_j to a new pk'_j. Each signer i must run $\mathsf{Update}_{\mathsf{pk}}$ for each other signer index j at the end of each epoch.

In light of this, we slightly modify the adversarial model of Figure 2 to encompass schemes with updates. We provide the formal security game for unforgeability with proactively refreshable keys in Figure 7 in Appendix B. In the unforgeability game with proactive key refresh, each signer i holds a local view of all the public keys owned by all the other signers, represented by the data structure $\mathsf{pkArr}_i[\cdot]$. This change is due to the fact that signers independently verify the update tokens and possibly reject them. We now provide $\mathcal{A}$ access to an update oracle $\mathsf{OUpdate}$, which triggers a key update. When querying $\mathsf{OUpdate}$, $\mathcal{A}$ must input a set of malicious update tokens, which are submitted to other signers for verification. Honest signers must also refresh local keys, generate honest update tokens, and update their view of public keys based on the incoming tokens. Each call to $\mathsf{OUpdate}$ triggers a transition from one *epoch* to the next. The security game uses a counter ctr to track the number of signature queries per epoch, and resets it after each update. To win the game, $\mathcal{A}$ should still output a forgery, but must do so by making no more than d queries per epoch.

Our model is not intended to address (nor should it be confused with) traditional definitions of refreshable signatures [HJJ+97, KGG24, CGG+20]. The latter often deal with a mobile adversary that can periodically change the corruption set and do not impose restrictions on the number of signatures per epoch. Our model,

on the other hand, aims to upgrade the standard TSAUF by allowing updates to parts of the secret and public keys and by replacing exhausted entropy sources with fresh ones.

4 Protocol Description

In this section, we describe our two-round threshold Schnorr signature scheme in greater detail. For a high-level overview, we refer to Section 2.3.

Building Blocks. Our scheme is based on the interplay between several building blocks, namely:

- Random oracles $\mathsf{H}_{\mathsf{Sig}}, \mathsf{H}_{\mathsf{f}}, \mathsf{H}_0, \mathsf{H}_1$, of which $\mathsf{H}_{\mathsf{Sig}}$ and H_{f} map from $\{0,1\}^*$ into $\mathbb{Z}_p$, and $\mathsf{H}_0, \mathsf{H}_1$ map from $\{0,1\}^*$ into $\mathbb{G}$.
- A polynomial commitment scheme $\mathsf{CS} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{VerCom})$ for polynomials in $\mathbb{Z}_p[X]$.
- A NIZK $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ for the *join*[4] relation $\mathcal{R}$ defined as

$$\mathcal{R} := \{(\mathsf{opn}, \mathbb{x}), \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{opn}}\} \wedge \{(\mathsf{evl}, \mathbb{x}), \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{evl}}\}.$$

  The relation $\mathcal{R}_{\mathsf{opn}}$ is defined as

$$\mathcal{R}_{\mathsf{opn}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{ck}, g, h, v, \mathsf{C}_{\mathsf{f}}, \mathsf{C}, i, \mathbf{d}) \\ \mathbb{w} = (\mathsf{f}, \rho_{\mathsf{f}}) \end{matrix} : \mathsf{C}_{\mathsf{f}} = \mathsf{Com}(\mathsf{ck}, \mathsf{f}; \rho_{\mathsf{f}}) \right\}.$$

  Looking ahead to our proof, proofs over the relation $\mathcal{R}_{\mathsf{opn}}$ will allow us to extract the updated signing key (aka fresh polynomials) from the adversary after a key update. The relation $\mathcal{R}_{\mathsf{evl}}$ is parametrized by random oracles $\mathsf{H}_0, \mathsf{H}_1$ and is defined as

$$\mathcal{R}_{\mathsf{evl}} := \left\{ \begin{matrix} \mathbb{x} = (\mathsf{ck}, g, h, v, \mathsf{C}_{\mathsf{f}}, \mathsf{C}, \mathbf{d}, R, \Lambda) \\ \mathbb{w} = (\mathsf{f}, \rho_{\mathsf{f}}, x, u, w) \end{matrix} : \begin{matrix} \mathsf{C}_{\mathsf{f}} = \mathsf{Com}(\mathsf{ck}, \mathsf{f}; \rho_{\mathsf{f}}), \\ \mathsf{C} = g^x h^w v^u, \\ R = (g^r \mathsf{H}_0(\mathbf{d})^w \mathsf{H}_1(\mathbf{d})^u)^\Lambda \\ r = f(\mathsf{H}_{\mathsf{f}}(\mathbf{d})) \end{matrix} \right\}.$$

  Looking ahead to our proof, proofs over the relation $\mathcal{R}_{\mathsf{evl}}$ will guarantee that the random commitments R sent by the adversary are well-formed w.r.t. our protocol description, i.e., the random element r was obtained by evaluating a polynomial f committed into the commitment C_{f}, and R was computed using r, public bases, and the secret key parts w and u, that are committed in the commitment C (aka, the partial public key). We say that a statement of the form $\mathbb{x} = (\mathsf{evl}, \mathbb{x}')$ (resp. $(\mathsf{opn}, \mathbb{x}')$) is an evl-instance (resp. opn-instance) of the join relation $\mathcal{R}$.

Setup. To set up our scheme for parameters (t, n, d), we select a public cyclic group $\mathbb{G}$ of prime order p and generator g. We further sample two group elements $h, v \leftarrow\!\!\$\, \mathbb{G}$, whose discrete logarithm in base g should be unknown. We select four hash functions $\mathsf{H}_{\mathsf{Sig}}$, H_{f}, H_0, and H_1. The first two map arbitrary bitstrings into $\mathbb{Z}_p$ scalars, while the latter two map arbitrary bitstrings into $\mathbb{G}$. We model all of them as random oracles. Then, we sample a commitment key ck for the polynomial commitment CS. This setup allows a t-out-of-n threshold signature scheme with up to d messages signed before a partial public key refresh (c.f. Section 2.3).

[4] The relation $\mathcal{R}_{\mathsf{opn}}$ is only needed to manage key updates, hence could be safely ignored for scenarios with a bounded number of signatures. Moreover, the reason why we opt for a single NIZK instead of one for each relation is rather technical and has to do with the security guarantees we require from the composition, as we elaborate in Section 4.1.

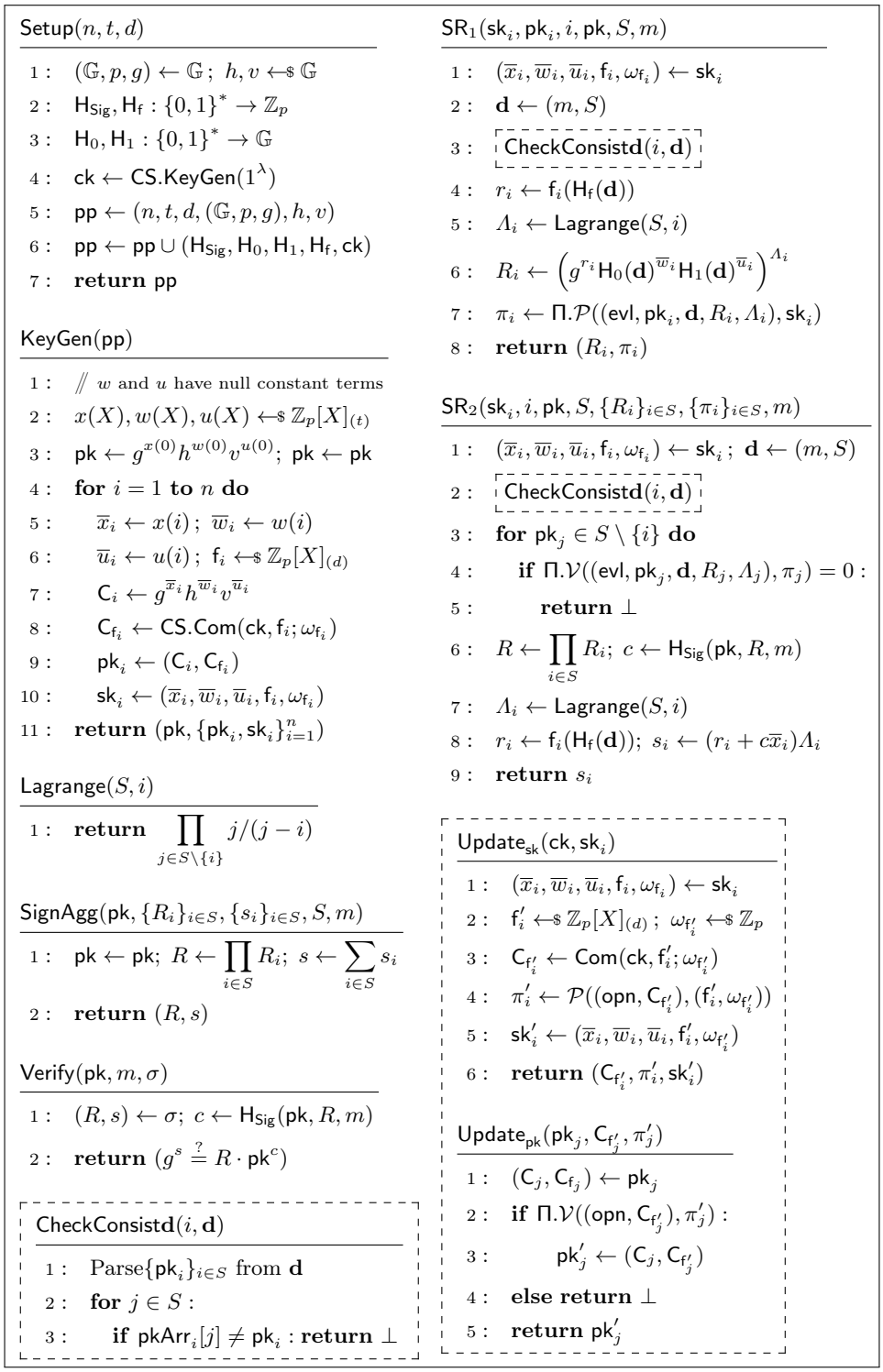

Fig. 3. Our two-round threshold Schnorr signature scheme. The signer set S contains the partial public keys of all signers involved in a signing operation. We abuse the notation of this set and also just select the indices of these signers whenever needed. We mark the optional key-update algorithms in a dashed box.

Key Generation. Key generation begins by constructing the joint public key pk and then distributing the corresponding signing key among the signers. We sample three random polynomials $x, u, w \leftarrow\!\!\$\ \mathbb{Z}_p[X]_{(t)}$ of degree t, with u and w chosen uniformly from the set of polynomials with zero constant term. The global public key is set to $\mathsf{pk} = g^{x(0)}h^{w(0)}v^{u(0)} = g^{x(0)}$.

For each signer $i \in [n]$, we evaluate the polynomials at i to obtain $\overline{x}_i = x(i)$, $\overline{w}_i = w(i)$, and $\overline{u}_i = u(i)$. We then sample a uniformly random polynomial $\mathsf{f}_i \in \mathbb{Z}_p[X]_{(d)}$ of degree d and commit to it using the polynomial commitment scheme CS, obtaining $\mathsf{C}_{\mathsf{f}_i} \leftarrow \mathsf{CS.Com}(\mathsf{ck}, \mathsf{f}_i; \rho_{\mathsf{f}_i})$ for uniformly random ρ_{f_i}. The partial public key of signer i is $\mathsf{pk}_i = (\mathsf{C}_i, \mathsf{C}_{\mathsf{f}_i})$ with $\mathsf{C}_i = g^{\overline{x}_i}h^{\overline{w}_i}v^{\overline{u}_i}$, and the partial signing key is $\mathsf{sk}_i = (\overline{x}_i, \overline{w}_i, \overline{u}_i, \mathsf{f}_i, \rho_{\mathsf{f}_i})$. For the ease of readability, we denote sampling of the polynomial via $\mathsf{f}_i \leftarrow\!\!\$\ \mathbb{Z}_p[X]_{(d)}$, and assume that the signer stores all polynomial coefficients. However, we can replace this directly with an on-the-fly sampling method [BDK+18] in which a signer would just store a uniformly random key $k_{\mathsf{RO},i}$ and sample the polynomial coefficients via $a_{i,j} = \mathsf{RO}(k_{\mathsf{RO},i}, j)$ whenever the polynomial needs to be evaluated (c.f. Section 2.3).

First Round. In the first round, we deterministically derive the signing nonces R_i. As a basis for the signing rounds serves the session identifier $\mathbf{d} \leftarrow (m, S)$. Here, m denotes the message to be signed and S contains the combined public key pk, as well as all partial public keys of all signers (at least t) involved in the signing operation. We want to emphasize that by this choice, all signing queries on the same $\mathbf{d}$ result in the same signature. In addition, when the partial public keys are updated, the signer set S is also updated, thus leading to a fresh session identifier. While S contains the partial public keys, we will abuse notation and also assume that we can simply iterate over the indices of participating signers when iterating over S. Each active signer evaluates the polynomial f_i over input $\mathsf{H}_{\mathsf{f}}(\mathbf{d}) \in \mathbb{Z}_p$, obtaining a scalar $r_i \in \mathbb{Z}_p$. It then computes the Lagrange coefficient Λ_i corresponding to the own index i and the signer set S, and the partial nonce

$$R_i = \left(g^{r_i}\mathsf{H}_0(\mathbf{d})^{\overline{w}_i}\mathsf{H}_1(\mathbf{d})^{\overline{u}_i}\right)^{\Lambda_i},$$

where $r_i = \mathsf{f}_i(\mathsf{H}_{\mathsf{f}}(\mathbf{d}))$, and $\overline{w}_i, \overline{u}_i$ are parts of the partial signing key. Furthermore, each signer i computes a non-interactive zero-knowledge proof π_i, showing the well-formedness of the nonce R_i w.r.t. the evl-statement consisting of the partial public key pk_i, the session $\mathbf{d}$, the random nonce R_i, and the Lagrange coefficient Λ_i, and the witness being signer i's signing key sk_i. Note that this statement-witness pair matches the description of the relation $\mathcal{R}_{\mathsf{evl}}$, since the partial keys sk_i and pk_i contain all the required elements. We provide details on how we instantiate this NIZK in Section 4.1. Finally, each signer outputs the pair (R_i, π_i).

Second Round. Upon receiving the nonce-proof pairs $\{(R_j, \pi_j)\}_{j \in S}$ of all other involved signers, signer i verifies all the well-formedness proofs by computing $\Pi.\mathcal{V}((\mathsf{evl}, \mathsf{pk}_j, \mathbf{d}, R_j, \Lambda_j), \pi_j)$, and aborts if any nonce is not well-formed. Upon successful verification of all nonces, signer i aggregates the partial nonces into the combined nonce $R = \prod_{i \in S} R_i$ and computes the random challenge $c \leftarrow \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$. Finally, it outputs its partial signature $s_i = (r_i + c\overline{x}_i)\Lambda_i$. We want to emphasize that the partial signing randomness r_i does not need to be stored between rounds, but can be recomputed by signer i through evaluating f_i on input $\mathsf{H}_{\mathsf{f}}(\mathbf{d})$. This holds, since r_i is deterministically derived. From an implementation viewpoint, this allows our protocol to be stateless across rounds.

Signature Aggregation and Verification. Once a signer receives all partial signatures, the final signature consists of a pair $\sigma = (R, s)$, with R defined as above, and s being the sum of all partial signatures $s = \sum_{i \in S} s_i$. The verification procedure is identical to a standard Schnorr signature verification. That is, one computes $c = \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$ and checks that $\mathsf{pk}^c \cdot R = g^s$.

(Optional) Key Update. To update the polynomial commitment included in the partial public keys, each signer i samples a fresh uniformly random polynomial $\mathsf{f}'_i \leftarrow\!\!\$\ \mathbb{Z}_p[X]_{(d)}$ of degree d, and commits to this polynomial via $\mathsf{C}_{\mathsf{f}'_i} \leftarrow \mathsf{Com}(\mathsf{ck}, \mathsf{f}'_i; \rho_{\mathsf{f}'_i})$ using a fresh randomness $\rho_{\mathsf{f}'_i}$. Then, each signer proves knowledge of f'_i via $\pi'_i \leftarrow \mathcal{P}((\mathsf{opn}, \mathsf{C}_{\mathsf{f}'_i}), (\mathsf{f}'_i, \rho_{\mathsf{f}'_i}))$, updates it's signing key $\mathsf{sk}'_i \leftarrow (\overline{x}_i, \overline{w}_i, \overline{u}_i, \mathsf{f}'_i, \rho_{\mathsf{f}'_i})$ to contain the fresh polynomial, and the fresh commitment randomness, and sends the pair $(\mathsf{C}_{\mathsf{f}'_i}, \pi'_i)$ as update token to each other signer j. Upon receiving an update token from another signer, each signer verifies the proof π'_j w.r.t.

the fresh commitment $\mathsf{C}_{\mathsf{f}'_j}$ and updates the corresponding partial public key $\mathsf{pk}_j = (\mathsf{C}_j, \mathsf{C}_{\mathsf{f}_j})$ to now contain the fresh polynomial commitment, i.e., $\mathsf{pk}_j = (\mathsf{C}_j, \mathsf{C}_{\mathsf{f}'_j})$.

We stress that the update mechanism does not require global synchronization or long-term state. Each signer can decide independently when to refresh its token, and we make no assumption that corrupted parties ever perform an update. In particular, an honest party can rotate its polynomial by simply broadcasting a fresh token; verification by others is one-sided and does not require a round-trip exchange. This makes the update process closer to a lightweight gossip mechanism than a coordinated protocol step. In practice, the parameter d can be chosen by each signer individually, and even large enough that it suffices for honest parties to refresh their polynomial on a fixed schedule, e.g., daily or weekly, while still ensuring adaptive security for all signatures within the epoch. This approach allows having full adaptive security even without tracking how many signatures have been signed so far.

Argument of Correctness. Intuitively, correctness follows from the fact that the polynomials w and u have zero constant term, and thus vanish at the origin. As a result, their contributions in the aggregated nonce cancel out, leaving only the g^r term. The aggregated partial signatures then reduce to $(g^r, r + c\,x(0))$, which satisfies the standard Schnorr verification equation for the public key $\mathsf{pk} = g^{x(0)}$. We provide a detailed argument of correctness in Appendix D.1.

Fully-Adaptive Unforgeability. We now state the security of our construction, discuss practical instantiations for the proof systems in Section 4.1, and provide a formal proof of Theorem 1 in Section 5.

Theorem 1 (Main). *Let $\mathbb{G}$ be a cyclic group, CS be a perfectly hiding, computationally binding trapdoor polynomial commitment scheme, Π be a simulation extractable zero-knowledge proof system for the join relation $\mathcal{R}$, and $\mathsf{H}_{\mathsf{Sig}}, \mathsf{H}_{\mathsf{f}}, \mathsf{H}_0, \mathsf{H}_1$ be hash functions modeled as random oracles. If DDH is hard in $\mathbb{G}$, our two-round threshold Schnorr signature scheme over $\mathbb{G}$ (Figure 3) is unforgeable w.r.t. a fully-adaptive adversary.*

4.1 Instantiating the Proof System

In Theorem 1, we claim that our threshold signature scheme is fully-adaptively secure if there exists a perfectly hiding, computationally binding trapdoor polynomial commitment scheme, together with a simulation-extractable zero-knowledge proof system for the join relation $\mathcal{R}$. Before proving the security of this theorem, we first provide concrete instantiations of these primitives and show that, based on these choices, our protocol is both efficient and provably secure under the Decisional Diffie-Hellman (DDH) assumption in the random oracle model. We begin by examining the join relation $\mathcal{R}$ in detail, as it determines the structure of the proof statements required in our protocol. We then discuss the selection of a suitable proof system that satisfies our security requirements while enabling efficient implementation under reasonable assumptions. Finally, we describe explicit instantiations for proving $\mathcal{R}$.

The join relation $\mathcal{R}$. We use a single NIZK Π for the join relation $\mathcal{R}$, rather than two different schemes Π_{opn} and Π_{evl}. The main reason for this choice has to do with the security guarantees we require from the composition of proofs. Roughly, we would need Π_{opn} to be simulation-extractable even in presence of simulated proofs for Π_{evl} since these proofs possibly provide additional information that may interfere with the security guarantees for the former scheme: hence, even if we instantiate Π_{opn} with a simulation-extractable scheme, this may be insecure when used in combination with other proof systems (see [FFK+23] for a similar issue).

It is worth noticing that it is possible to efficiently and naturally instantiate Π from two distinct NIZKs, Π_{opn} and Π_{evl}, as long as both these schemes are simulation-extractable with respect to a *trapdoorless* zero-knowledge simulator, i.e., the zero-knowledge simulator does not rely on any trapdoor but only leverages the ability to control/program the random oracle. Intuitively, this property allows us to reduce the security of the join scheme to the security of either scheme by *locally* simulating the proofs for the other one, without any concrete interference, due to the domain separation offered by the random oracle. We refer to [CFR25] for an in-depth overview of these techniques and for a general composition framework in the context of simulation-extractable schemes.

A suitable proof system. A line of works [GM17, FFK+23, KPT23, DG23, FFR24, CFR25] has shown that many general-purpose zkSNARKS, including (possibly slight variants of) Plonk [GWC19], Marlin [CHM+20], Lunar [CFF+21], Basilisk [RZ21], and Spartan [Set20], are simulation-extractable, and thus would be sufficient for our protocol. However, they usually require representing the relation in some intermediate format such as R1CS or Plonkish. Moreover, for some of these schemes, we only know they are secure under stronger security assumptions, such as the AGM [FKL18] or some knowledge assumptions, or their most efficient constructions require an additional trusted setup, which we would like to avoid.

Luckily for us, when we use the Pedersen (polynomial) commitment scheme, both $\mathcal{R}_{\mathsf{opn}}$ and $\mathcal{R}_{\mathsf{evl}}$ can be instantiated efficiently with transparent schemes, in particular relying on the zero-knowledge commit-and-prove inner-product argument (IPA) at the core of Bulletproofs [BBB+18] and Hyrax [WTs+18], which is logarithmic in the size of the witness $\mathbb{w}$ (the committed polynomial). Moreover, this scheme is simulation-extractable with respect to a trapdoor-less zero-knowledge simulator, in the ROM, under the discrete logarithm (DLog) assumption [DG23, CFR25]. Since DLog is implied by DDH, this choice of a proof system does not add additional assumptions to our protocol.

Instantiations. We now give a detailed overview of how the inner-product argument is used to prove the join relation $\mathcal{R}$. For clarity, we first describe the (sub)protocols in this section as *interactive* arguments between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where the verifier's messages are sampled uniformly at random from a prescribed domain. By applying the Fiat–Shamir transform, we can compile these arguments into non-interactive proofs in the random oracle model, with $\mathcal{P}$ and $\mathcal{V}$ given black-box access to a random oracle H.

At the core of our instantiation is a zero-knowledge commit-and-prove inner-product argument (IPA) for the following dot-product relation:

$$\mathcal{R}_{\mathsf{LogDotPf}} = \left\{ (n, \boldsymbol{g}, g_\rho, \mathsf{P}, \boldsymbol{a}), (\boldsymbol{x}, y, \rho_p) : \mathsf{P} = g_0^y \cdot \prod_{i \in [n]} g_i^{x_i} \cdot g_\rho^{\rho_p}, y = \langle \boldsymbol{x}, \boldsymbol{a} \rangle \right\}$$

Here, $(\boldsymbol{g}, g_\rho)$ are Pedersen commitment generators, P is a Pedersen commitment to $(y, \boldsymbol{x})$ with opening ρ_p, and $\boldsymbol{a}$ is a public vector. We use this relation to prove the correct evaluation of a polynomial given only (hiding) Pedersen commitments to the polynomial coefficients $\boldsymbol{x}$ and to the evaluation output y.[5] In the public statement of $\mathcal{R}_{\mathsf{LogDotPf}}$, we include the dimension n (corresponding to being the polynomial degree, i.e., $n = d + 1$), the Pedersen generators $(\boldsymbol{g}, g_\rho)$, the commitment P, and the Vandermonde vector $\boldsymbol{a}$ of the evaluation point z. The witness consists of the polynomial coefficients $\boldsymbol{x}$, the evaluation result y, and the commitment opening ρ_p. The relation asserts that y is indeed the correct polynomial evaluation and that P is a valid commitment to $(y, \boldsymbol{x})$ under opening ρ_p. We choose this dot-product formulation because it admits an efficient, logarithmic-size proof in the random oracle model based on the discrete logarithm assumption, as we formalize hereafter.

Lemma 1 ([CFR25]). *There exists a logarithmic-size dot-product argument* $\mathsf{LogDotPf}$ *for the relation* $\mathcal{R}_{\mathsf{LogDotPf}}$ *that is simulation-extractable in the ROM under the discrete logarithm assumption.*

Instantiating $\mathcal{R}_{\mathsf{opn}}$. To prove knowledge of a polynomial f, i.e., to prove the relation $\mathcal{R}_{\mathsf{opn}}$, the verifier chooses a random challenge point $z \leftarrow\!\!\$\ \mathbb{Z}_q$, the prover evaluates $y = \mathsf{f}(z)$, and commits to y via $\mathsf{C}_y = g^y g_\rho^{\rho_y}$. The parties then invoke $\mathsf{LogDotPf}$ on public input $(d+1, g, g_\rho, \mathsf{C}_\mathsf{f} \cdot \mathsf{C}_y, z)$, with the prover's witness $(\mathsf{f}, y, \rho_\mathsf{f} + \rho_y)$, where ρ_f is the opening for the polynomial commitment C_f.

Instantiating $\mathcal{R}_{\mathsf{evl}}$. To prove the well-formedness of a nonce R with respect to a partial public key $\mathsf{pk}_i = (\mathsf{C}, \mathsf{C}_\mathsf{f})$ and a session identifier $\mathbf{d}$, the prover $\mathcal{P}$ first evaluates $r = \mathsf{f}(z)$ with $z = \mathsf{H}_\mathsf{f}(\mathbf{d})$ and commits to it as $\mathsf{C}_r = g^r g_\rho^{\rho_r}$. It then proves correct polynomial evaluation using $\mathsf{LogDotPf}$ for $\mathcal{R}_{\mathsf{LogDotPf}}$ on the commitment

[5] The connection between the inner product and polynomial evaluation is immediate: for $\boldsymbol{a} = (1, z, z^2, \ldots, z^n)$, the equality $y = \langle \boldsymbol{x}, \boldsymbol{a} \rangle$ corresponds to $y = \mathsf{f}(z)$ for the polynomial $\mathsf{f}(X) = \sum_{i=0}^{n} x_i X^i$ with coefficients $\boldsymbol{x}$.

$\mathsf{C}_r \cdot \mathsf{C}_\mathsf{f}$. Finally, the prover and verifier run a Σ-protocol for the relation

$$\mathcal{R}_\Sigma = \{(\mathsf{C}_r, R, \mathsf{C}, g, h, v, g_\rho, h_1, h_2,), (r, \rho_r, u, w, x) : \\ R = g^r h_1^w h_2^u,\ \mathsf{C}_r = g^r g_\rho^{\rho_r},\ \mathsf{C} = g^x h^w v^u\},$$

which shows that the opening of C_r and the committed (x, w, u) in C correctly explain R as $R = (g^r h_1^w h_2^u)^{\Lambda_i}$ for the Lagrange coefficient Λ_i. This proof has logarithmic size in the polynomial degree d, due to the proof for $\mathcal{R}_{\mathsf{LogDotPf}}$. In addition, the sigma protocol adds three $\mathbb{G}$ and five $\mathbb{Z}_p$ elements to this proof (c.f. Figure 8). We give full protocol details in Figure 8.

5 Security Analysis

In this section, we prove the fully adaptive security of our two-round Schnorr threshold signature scheme under the DDH assumption. We first present Theorem 2, a simplified form of the main theorem (Theorem 1) in which the adversary's number of signing queries is bounded, so no key updates occur. This initial proof is meant to clearly showcase our new strategy—combining equivocal deterministic nonce derivation with correlated oracle programming—without the additional complications of key updates. We then move to Theorem 3, which lifts the restriction on signing queries and allows the adversary to make a polynomial number of them, leveraging partial-key updates. Here, in addition to the techniques from the simplified setting, we employ a knowledge extractor to recover the adversary's updated keys and demonstrate how this extraction integrates with our rewinding strategy. We defer the outline and proof of Theorem 3 to Appendix B.

Theorem 2 (2-round Adaptive Security for Bounded Signature Queries). *Fix $n > t \geq 1$ and a group description $(\mathbb{G}, p, g)$ such that p is a λ-bit prime. Consider an adversary $\mathcal{A}$ making at most $q_s, q_{\mathsf{Sig}}, q_0$ and q_1 queries to $\mathsf{OSR}, \mathsf{H}_{\mathsf{Sig}}, \mathsf{H}_0$ and H_1, respectively. Furthermore, suppose that the adversary makes no more than q_f calls to H_f, and no more than $d+1$ calls of the form $\mathsf{OSR}(1, \cdot, \cdot, \cdot, i)$ for each $i \in [n]$.*

If $\mathcal{A}$ has an advantage of $\epsilon_{\mathsf{TSAUF}} = \mathsf{Adv}^{\mathsf{tsauf}}_{\mathsf{2RTSch},\mathcal{A}}(\lambda)$ there exists a PPT algorithm $\mathcal{C}_{\mathsf{DL}}$ having an advantage of $\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}$ such that

$$\epsilon_{\mathsf{TSAUF}} \leq \frac{q_{\mathsf{Sig}}}{p} + \sqrt{q_{\mathsf{Sig}} \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda)} + \frac{q_\mathsf{f}^2 + q_0^2 + q_1^2}{p} + \mathsf{Adv}^{\mathsf{sfzkp}}_{\Pi,\mathcal{B}_{\mathsf{sfzkp}}}(\lambda) \\ + q_s \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda) + \frac{q_s \cdot q_{\mathsf{Sig}}}{p} + \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda) + \frac{1}{p},$$

for some PPT algorithms $\mathcal{B}_{\mathsf{sfzkp}}, \mathcal{B}_{\mathsf{ss}}$, and $\mathcal{B}_{\mathsf{DDH}}$. The first two are PPT adversaries against the zero-knowledge and soundness of Π respectively, and the last one is a PPT adversary against the DDH game.

Proof Overview. The goal of our proof is to reduce the fully adaptive unforgeability of our two-round Schnorr threshold signature scheme to the hardness of the discrete logarithm problem (DLog). At a high level, our approach is to solve DLog by exploiting the special-soundness property of the Σ-protocol underlying Schnorr signatures [Sch90, CDS94]. Concretely, given two Schnorr signatures $s = c \cdot \mathsf{sk} + r$ and $s' = c' \cdot \mathsf{sk} + r$ that use the same nonce $R = g^r$ but have distinct challenges $c \neq c'$, we can extract the secret key as $\mathsf{sk} = (s - s') \cdot (c - c')^{-1}$.

To recreate this extraction in the security proof, we rely on the popular forking lemma of Bellare and Neven [BN06]. Intuitively, in our proof, we consider an adversary that produces a single valid signature forgery on a random nonce R. We then rewind this adversary and execute it again with a slight change in its sequence of random oracle responses. The forking lemma guarantees that, with "good" probability, the adversary will forge again, and the two forgeries will share the same nonce R while differing in their challenges (since we changed the random oracle responses at some point). Moreover, it provides a quantitative bound relating this probability to the original success probability of the forger. With some additional postprocessing–discussed later–this allows us to construct a DLog adversary $\mathcal{C}_{\mathsf{DL}}$.

In our reduction, we do not directly plug $\mathcal{A}$ into the forking construction. Instead, we first define a sequence of security games $\mathsf{Game}_0, \mathsf{Game}_1, \ldots$ where $\mathsf{Game}_0 = \mathsf{TSAUF}$ and each Game_i introduces only minimal changes compared to its predecessor Game_{i-1}, while remaining provably indistinguishable from it from $\mathcal{A}$'s perspective. We establish this indistinguishability under our stated assumptions, all of which follow from the DDH assumption. At the start of this sequence, we instantiate the fully adaptive unforgeability game with our signature construction. As we progress through the game hops, we incrementally "rig" the signing keys (cf. Section 2.1), derive nonces in a deterministic yet equivocable manner, and ensure that the adversary's interaction follows our intended protocol. In the final game, we construct a wrapper adversary $\mathcal{B}$ that internally runs $\mathcal{A}$ and perfectly simulates the game's oracles until $\mathcal{A}$ outputs a forgery. This wrapper $\mathcal{B}$ is precisely the forger required by the forking construction. By running and rewinding $\mathcal{B}$ (and thereby $\mathcal{A}$) with a slight change in randomness, we obtain two valid forgeries on the same nonce with "good" probability, enabling us to solve DLog. Since the discrete logarithm problem is assumed to be hard, no such adversary can exist, and our scheme is therefore fully adaptively secure.

We conclude the overview by explaining how the DLog challenge is embedded into our threshold protocol. Let $(\mathbb{G}, p, g)$ be the target group description. Given a challenge $X \leftarrow\!\$\ \mathbb{G}$, the reduction $\mathcal{C}_{\mathsf{DL}}$ embeds it into the protocol by using X as the second generator h. Specifically, upon receiving X, $\mathcal{C}_{\mathsf{DL}}$ samples $\alpha_v \leftarrow\!\$\ \mathbb{Z}_p$ and sets the public key generators to $(g, h, v) = \left(g, X, g^{\alpha_v}\right)$ in both internal executions of $\mathcal{B}$. This embedding requires a slight adjustment to the post-processing to extract the DLog of X. I.e., $\mathcal{C}_{\mathsf{DL}}$ first rewinds $\mathcal{B}$ to learn the signing key sk, then exploits the fact that the signing key sk is "rigged" in the final game (i.e., $\mathsf{sk} = x + \alpha_h + \alpha_v \cdot u$ for known α_v, x, u), to learn the DLog of X (i.e. α_h) when sk is known. A rigorous and extended development of the aforementioned high-level idea can be found in Appendix A.

6 Practical Deployment

To demonstrate deployability, we implemented and benchmarked our equivocable deterministic nonce derivation from polynomials and the accompanying proofs of well-formedness. These operations account for the main computational effort in our two-round threshold Schnorr protocol. Our prototype builds on standard primitives: group operations are instantiated over Ristretto255 and well-formedness is ensured using Bulletproofs. For the latter, we rely on the existing inner-product argument implementation in the `bulletproofs` crate, already optimized for Ristretto255, which allows us to reuse well-established components rather than engineering a new proof system from scratch. Our prototype implementation is available at `github`.

Benchmarks. We benchmarked both proving and verification for nonce derivation (i.e., round one of our signature scheme), as well as the computation and verification of update tokens, on an Apple M3 Pro with 36 GB RAM. For small instances, proving and verification are fast, taking about 1.6 ms and 0.6 ms, respectively, at $d = 16$. At medium size, $d = 1024$, runtimes grow to 66 ms for proving and 13 ms for verification, while at the largest tested scale, $d = 16{,}384$, proving completes in about 984 ms and verification in 186 ms. Verification is consistently about 5–7× faster than proving, and can also be batched using standard techniques, further reducing amortized cost [BBB+18]. The remaining computation costs of our scheme (i.e., computing the partial signature and combining partial signatures) are on the order of microseconds, and are therefore negligible.

Standards compliance. Our benchmark confirms that our construction meets the efficiency requirements emphasized in NIST IR 8214B [BD22]: we achieve "low latency" (due to two rounds), "small communication" (succinct protocol messages), and "sustainable throughput" (millisecond computation and verification on commodity hardware), demonstrating practicality for threshold Schnorr deployment.

Comparison with musig-DN. We further compared our protocol to the deterministic two-round (non-adaptive) Schnorr multisignature musig-DN [NRSW20], explicitly referenced in the NIST call, running its public implementation [Nic20] on the same machine. The verifiable deterministic nonce computation in musig-DN takes about 545 ms for proving and ≈ 23 ms for verification, which is comparable to our scheme with an

update after each 8192 signatures (requiring 488 ms for proving and 93 ms for verification). Our verification is slower than musig-DN because we use a pure-Rust Ristretto implementation without the highly optimized multi-scalar multiplication and batch-verification routines available in secp256k1-zkp. Both protocols rely on Bulletproofs, leading to comparable message sizes.

On-the-fly evaluation. As we discussed in Section 2.3, the secret key size of each signer can be optimized by deriving the nonce polynomial coefficients on the fly rather than storing them. We implemented this derivation using SHA-512 and ChaCha. Compared to plain table lookups ($\approx 1.4\,\mu$s at $d = 16$ and ≈ 0.30 ms at $d = 16{,}384$), the on-the-fly method adds only a small overhead: $\approx$ 3–6 μs at $d = 16$, $\approx$ 0.6–0.9 ms at $d = 16{,}384$, and 2–3 ms even at $d = 65{,}536$. In all cases, the evaluation cost is negligible relative to proof generation, while eliminating the need to store coefficient tables and thereby reducing secret key size to contain just five $\mathbb{Z}_p$ elements.

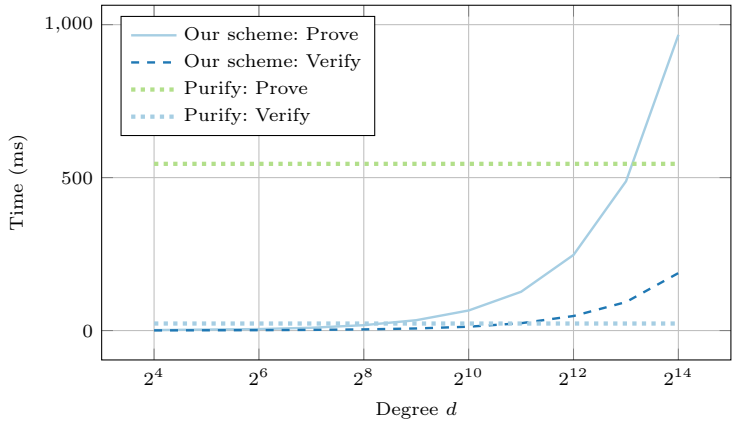

Fig. 4. Prove and verify time vs. polynomial degree d (log-scaled x). Purify shown as constant horizontal baselines.

References

Ama. Amazon Web Services. Aws kms best practices — data protection key rotation. https://docs.aws.amazon.com/prescriptive-guidance/latest/aws-kms-best-practices/data-protection-key-rotation.html. Accessed 2025-08-11.

ANO+22. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572. IEEE Computer Society Press, May 2022.

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.

BCC+25. Christian Badertscher, Matteo Campanelli, Michele Ciampi, Luigi Russo, and Luisa Siniscalchi. Universally composable SNARKs with transparent setup without programmable random oracle. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 225–258. Springer, Cham, August 2025.

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, Cham, August 2022.

BD22. Luís T. A. N. Brandão and Michael Davidson. Notes on threshold eddsa/schnorr signatures. Technical Report NIST IR 8214B (Draft), National Institute of Standards and Technology (NIST), August 2022. Initial Public Draft.

BDK+18. Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals - kyber: A cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, 2018.

BDLR25a. Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Adaptively secure three-round threshold schnorr signatures from DDH. Cryptology ePrint Archive, Paper 2025/1009, 2025.

BDLR25b. Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part II*, volume 15602 of *LNCS*, pages 304–334. Springer, Cham, May 2025.

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.

BHL24. Dan Boneh, Iftach Haitner, and Yehuda Lindell. Exponent-VRFs and their applications. Cryptology ePrint Archive, Report 2024/397, 2024.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

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, Berlin, Heidelberg, January 2003.

BPR22. Dan Boneh, Aditi Partap, and Lior Rotem. Proactive refresh for accountable threshold signatures. Cryptology ePrint Archive, Paper 2022/1656, 2022.

BPR24. Dan Boneh, Aditi Partap, and Lior Rotem. Proactive refresh for accountable threshold signatures. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 140–159. Springer, Cham, March 2024.

CCL+20. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Cham, May 2020.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Berlin, Heidelberg, August 1994.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, February / March 2013.

CFF+21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021.

CFGN96. Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

CFR25. Matteo Campanelli, Antonio Faonio, and Luigi Russo. SNARKs for virtual machines are non-malleable. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 153–183. Springer, Cham, May 2025.

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.

CGH98. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

CGRS23. Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.

CKGW24. Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures. RFC 9591, June 2024.

CKK+25a. Elizabeth Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. On the adaptive security of FROST. Cryptology ePrint Archive, Paper 2025/1061, 2025.

CKK+25b. Elizabeth C. Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. On the adaptive security of FROST. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 480–511. Springer, Cham, August 2025.

CKM23. Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023.

CKM25. Elizabeth Crites, Chelsea Komlo, and Mary Maller. On the adaptive security of key-unique threshold signatures. Cryptology ePrint Archive, Paper 2025/943, 2025.

CS25a. Elizabeth Crites and Alistair Stewart. A plausible attack on the adaptive security of threshold schnorr signatures. Cryptology ePrint Archive, Paper 2025/1001, 2025.

CS25b. Elizabeth C. Crites and Alistair Stewart. A plausible attack on the adaptive security of threshold schnorr signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 457–479. Springer, Cham, August 2025.

DDFY94. Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.

DDO+01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001.

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, Berlin, Heidelberg, August 1988.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990.

DG23. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 531–562. Springer, Cham, April 2023.

DJN+20. Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 382–400. Springer, Cham, September 2020.

DKLs19. Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

DKM+24. Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.

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, Cham, September 2020.

DR24. Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 251–284. Springer, Cham, August 2024.

FFK+23. Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 455–485. Springer, Cham, November / December 2023.

FFR24. Antonio Faonio, Dario Fiore, and Luigi Russo. Real-world universal zkSNARKs are non-malleable. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3138–3151. ACM Press, October 2024.

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, Berlin, Heidelberg, August 2005.

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, Cham, August 2018.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012.

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.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 2016*, volume 9696 of *LNCS*, pages 156–174. Springer, Cham, June 2016.

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Berlin, Heidelberg, May 1996.

GJKR03. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 373–390. Springer, Berlin, Heidelberg, April 2003.

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.

GKMN21. François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 2021. Springer, Cham.

GKS24. Kamil Doruk Gür, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*, pages 266–300. Springer, Cham, June 2024.

GKSŚ20. Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020.

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

Goo. Google Cloud. Cloud key management service — rotate a key. https://cloud.google.com/kms/docs/rotate-key. Accessed 2025-08-11.

GRJK00. Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, March 2000.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

HDWH12. Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, August 2012.

HJJ+97. Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97*, pages 100–110. ACM Press, April 1997.

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, Cham, October 2020.

KGG24. Brian Koziel, S. Dov Gordon, and Craig Gentry. Fast two-party threshold ECDSA with proactive security. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1567–1580. ACM Press, October 2024.

KPT23. Markulf Kohlweiss, Mahak Pancholi, and Akira Takahashi. How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In Guy N. Rothblum and Hoeteck Wee, editors,

TCC 2023, Part III, volume 14371 of *LNCS*, pages 486–512. Springer, Cham, November / December 2023.

KRT24. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.

LN18. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

Mic. Microsoft. Configure key rotation for azure key vault keys. https://learn.microsoft.com/en-us/azure/key-vault/keys/how-to-configure-key-rotation. Accessed 2025-08-11.

MPSW19. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptography*, 87(9):2139–2164, September 2019.

Nic20. Jonas Nick. Purify benchmarks in the bulletproofs reference code. https://github.com/jonasnick/secp256k1-zkp/tree/bulletproof-musig-dn-benches, 2020.

NR97. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 458–467, 1997.

NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020.

Pat09. Jacques Patarin. The "coefficients H" technique (invited talk). In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 328–345. Springer, Berlin, Heidelberg, August 2009.

Ped91. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Berlin, Heidelberg, April 1991.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Berlin, Heidelberg, May 1996.

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.

RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Cham.

Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020.

Sho00. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Berlin, Heidelberg, May 2000.

WNR21. Pieter Wuille, Jonas Nick, and Tim Ruffing. Bip 340, 341, 342: Schnorr signatures, taproot, and tapscript. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki, 2021. Accessed: 2025-08-14.

WTs+18. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

YCX21. Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 481–511. Springer, Cham, May 2021.

A Proof of Fully Adaptive Unforgeability without Key Updates

We now describe each of the steps introduced in Section 5 in detail, and in the following order: first, we outline the sequence of games, and prove that each pair of consecutive games is indistinguishable; next, we cover the $\mathcal{B}$ wrapper; finally, we describe the forking construction, its application to obtain $\mathcal{C}_{\mathsf{DL}}$, and analyze its advantage in calculating DLog.

Proof (Proof of Theorem 2). Throughout the following proof, we use the notation $\mathsf{Adv}^{\mathsf{game}}_{\mathcal{A}}(\lambda)$ to denote the advantage of interactive algorithm $\mathcal{A}$ (not necessarily equal to the main adversary) in game Game as a function of the security parameter λ. If Game pertains to the security property of a specific protocol/scheme Π, we include the latter as a suffix $\mathsf{Adv}^{\mathsf{game}}_{\Pi,\mathcal{A}}(\lambda)$. All the games that we refer to are outlined in the preliminaries, though sometimes implicitly. For the sake of clarity, we provide a concise summary of the relevant security games in the following:

- DDH: the game defining the DDH computational assumption, as defined in figure Figure 1.
- DL: the game defining the DL computational assumption, as defined in figure Figure 1.
- sfzkp: the game defining the zero-knowledge property for a proof system Π, implicit in Definition 3. $\mathcal{A}$ wins sfzkp if it distinguishes the real world, where it has oracle access to $\mathcal{P}$ and H, from the real world, where it only interacts with $\mathcal{S}$.
- sfext: the game defining the simulation extractability property for a proof system Π, implicit in Definition 4. $\mathcal{A}$ wins sfext if it outputs a statement-proof pair $(\mathbb{x}, \pi)$ that is accepted by $\mathcal{V}$, not in $\mathcal{Q}$, and for which $\mathcal{E}$ does not successfully extract a witness.
- ss: the game defining the simulation soundness property for a proof system Π. $\mathcal{A}$ wins ss if it outputs a statement-proof pair $(\mathbb{x}, \pi)$ that is accepted by $\mathcal{V}$, not in $\mathcal{Q}$, and for which $\mathbb{x}$ is a no-statement.

Whenever the games are parameterized but the parameters are made explicit, we omit them from the previous notation. For instance, although DDH and DL are parametrized by $(\mathbb{G}, g, p)$ we omit them when obvious.

Game_0. This is the original security game for our signature scheme, following the honest protocol. Let $h := g^{\alpha_h}$ and $v := g^{\alpha_v}$ for some $\alpha_h, \alpha_v \leftarrow\!\!\$\ \mathbb{Z}_p^*$. Then h, v, and g can serve as uniformly random generators sampled by the setup algorithm Setup. In this game, adversary $\mathcal{A}$ is granted access to any random oracle through the standard lazy simulation technique.

We additionally introduce two purely conceptual modifications to the game. First, let (m^*, σ^*) denote the forgery. We assume that $\mathcal{A}$ always queries $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m^*)$ prior to producing the forgery. Second, we assume that $\mathcal{A}$ makes exactly t distinct corruption queries. These changes are made without loss of generality and do not affect the advantage of $\mathcal{A}$ [BDLR25b]. To see this, we can always construct a wrapper adversary that internally runs $\mathcal{A}$.[6] The wrapper ensures that the query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m^*)$ is made and it corrupts exactly t signers before terminating. Then we have

$$\mathsf{Adv}^{\mathsf{tsauf}}_{\mathsf{2RTSch},\mathcal{A}}(\lambda) = \Pr[1 \leftarrow \mathsf{Game}_0] = \epsilon_{\mathsf{TSAUF}}.$$

Game_1. In this game, we abort if a collision for H_{f} occurs. More precisely, the game aborts if there exist two distinct signing sessions identifiers $\mathbf{d}_1 \neq \mathbf{d}_2$ submitted by $\mathcal{A}$ (at any point during its execution) to H_{f} such that $\mathsf{H}_{\mathsf{f}}(\mathbf{d}_1) = \mathsf{H}_{\mathsf{f}}(\mathbf{d}_2)$. This condition is easy to check by using a table T_{f} such that $T_{\mathsf{f}}[\mathbf{d}]$ contains $\mathsf{H}_{\mathsf{f}}(\mathbf{d})$ if $\mathbf{d}$ is the identifier of a session triggered by $\mathcal{A}$, and $\perp$ otherwise. With a similar strategy, using tables T_0 and T_1, we abort if a collision for either H_0 or H_1 occurs. This game is statistically indistinguishable from the previous one, and a precise bound can be found by upper-bounding the collision probability. Assuming no more than q_{f} evaluations of the random oracle, the collision probability for H_{f} is at most $\frac{q_{\mathsf{f}}^2}{p}$. With a similar analysis, followed by a union bound, the collision probability for either of H_0 and H_1 is no more than $\frac{q_0^2+q_1^2}{p}$,

$$|\Pr[1 \leftarrow \mathsf{Game}_0] - \Pr[1 \leftarrow \mathsf{Game}_1]| \leq \frac{q_{\mathsf{f}}^2 + q_0^2 + q_1^2}{p}.$$

[6] This can be done without a loss of generality, since the reduction knows all signing-key shares independent of how many corruptions are made.

Game_2. In this game, we replace the NIZK proofs generated by honest signers in the first round with simulated ones. That is, we generate π_i in the first round by making $\mathcal{A}$ interact with $\Pi.\mathcal{S}_1$ and $\Pi.\mathcal{S}_2$ for each $i \in HS$. Indistinguishability between Game_1 and Game_2 can be shown by constructing an adversary $\mathcal{B}_{\mathsf{sfzkp}}$ for game sfzkp that internally runs $\mathcal{A}$. $\mathcal{B}_{\mathsf{sfzkp}}$ receives public parameters from $\Pi.\mathsf{Setup}$ and simulates the rest of the game towards $\mathcal{A}$. Notice that Game_1 and Game_2 are identical except for the way in which NIZKs are generated, and thus the expression "rest of the game" (without specifying an index) is well-defined. In particular, $\mathcal{B}_{\mathsf{sfzkp}}$ runs KeyGen honestly, generates all the due keys, and simulates the random oracles and signing oracles towards $\mathcal{A}$. $\mathcal{B}_{\mathsf{sfzkp}}$ queries its pair of oracles, which either equals $(\Pi.\mathcal{P}, \mathsf{H})$ or $(\Pi.\mathcal{S}_1, \Pi.\mathcal{S}_2)$, when producing a proof π_i in the first round, and forwards it to $\mathcal{A}$. Finally, it outputs whatever the game outcome is. Since $\mathcal{B}_{\mathsf{sfzkp}}$ exactly simulates Game_1 or Game_2 depending on the random bit of the challenger of the sfzkp game, we obtain

$$|\Pr[1 \leftarrow \mathsf{Game}_1] - \Pr[1 \leftarrow \mathsf{Game}_2]| \leq \mathsf{Adv}^{\mathsf{sfzkp}}_{\Pi,\mathcal{B}_{\mathsf{sfzkp}}}(\lambda).$$

Game_3. Game_3 is identical to Game_2 except that the secret nonce r_i from honest signer i, which is computed by evaluating $\mathsf{f}_i(\mathsf{H}_{\mathsf{f}}(\mathbf{d}))$ in Game_3, is drawn uniformly at random in $\mathbb{Z}_p$ and stored in a table $T_r(i,\cdot)$, where $T_r(i,\mathbf{d})$ returns the corresponding r_i value (this way, the same integer r_i is used if signing on $\mathbf{d} = (m, S)$ is called again to signer i). We leverage Lagrangian interpolation to manage adaptive corruptions. That is, upon corruption of signer i we inspect the content of $T_r(i,\cdot)$ and run Lagrangian interpolation over it to reconstruct the f_i polynomial. If $T_r(i,\cdot)$ does not contain enough input-output pairs we append random ones until we match the desired degree of f_i (that is, we must collect $d+1$ pairs). Notice that, assuming every f_i polynomial is evaluated less or equal than $\deg(\mathsf{f}_i)+1$ times, Lagrangian interpolation is always feasible, and yields a consistent polynomial with probability 1. Furthermore, it is well known that the family of d-degree polynomials over $\mathbb{Z}_p$ is $(d+1)$-wise independent — that is, the first $(d+1)$ evaluations of a random polynomial look uniformly random and independent.

We introduce further changes concerning the key generation and the polynomial commitment, to enable opening $\mathsf{C}_{\mathsf{f}_i}$ to the desired value. We replace the call to $\mathsf{CS.KeyGen}$ at the beginning of the game with one to $\mathsf{CS.tdKeyGen}$. This provides knowledge of the trapdoor td to the challenger before interacting with $\mathcal{A}$. Furthermore, the polynomial commitments of honest users are generated by honestly committing to 0 (which for our choice of Pedersen commitments, is equivalent to calling $\mathsf{CS.tdOpen}$). After performing Lagrangian interpolation as previously described, the challenger leverages td to open $\mathsf{C}_{\mathsf{f}_i}$ to f_i. Assuming a perfectly hiding commitment with perfect equivocability, this implies perfect indistinguishability of the two games, that is

$$\Pr[1 \leftarrow \mathsf{Game}_2] = \Pr[1 \leftarrow \mathsf{Game}_3].$$

Game_4. In this game, we compute on our own all the nonce values R_i''s that we expect to receive from the adversary. That is, we compute for each signing session $\mathbf{d} = (m, S)$ all the R_i such that $i \in CS \cap S$. This allows us to deduce the aggregate nonce R as $\prod_{i \in CS \cap S} R_i' \cdot \prod_{i \in HS \cap S} R_i$, and the value $c \coloneqq \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$. Furthermore, both are unique to the signing session identified by (m, S). We check that the nonces R_i's sent by the adversary are equal to the expected value R_i'. If for some $i \in CS \cap S$ the proof π_i was valid but $R_i \neq R_i'$, the game aborts. Notice that the NIZK's simulation soundness aims at preventing exactly this event. More precisely, one can construct an adversary $\mathcal{B}_{\mathsf{ss}}$ that aims at breaking the simulation soundness of Π. $\mathcal{B}_{\mathsf{ss}}$ runs the honest Setup and KeyGen algortihms, thus generating all the signatures public parameters and secret/public keys honestly. It then simulates Game_4 towards $\mathcal{A}$, taking care of the random oracles and signing oracles as outlined in previous games. $\mathcal{B}_{\mathsf{ss}}$ waits until $\mathcal{A}$ produces an R_i value for some $i \in CS$, and checks if R_i is a yes-statement. If no, it submits R_i and the corresponding adversarially generated NIZK π_i to $\Pi.\mathcal{V}(\mathsf{evl}, \cdot)$. The probability that the latter accepts is at most $\mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda)$. Let q_s be an upper bound to the number of signing queries. Then, by a union bound

$$|\Pr[1 \leftarrow \mathsf{Game}_3] - \Pr[1 \leftarrow \mathsf{Game}_4]| \leq q_s \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda).$$

Game_5. In this game, we ensure that we can program all random oracles as required by our proof idea. I.e., when the adversary queries a signature on a message–signing-set pair (m, S), then this pair can either be

used as a fresh input to the random oracles, or all random oracles are already programmed consistently, such that simulating signing does work. This game is particularly required, such that we can perform correlated random-oracle programming in the games to follow. For this purpose, we introduce two maps χ and ρ, and number signing sessions using a generic index ℓ. Initially, we assume $\rho[\mathbf{d}] = 0$ by default for all possible $\mathbf{d}$. For every random oracle query to H_b with input $\mathbf{d}$ and $b \in \{0,1\}$, we do:

- if $\mathsf{H}_b(\mathbf{d}) = \bot$, then we sample random values $\beta_\ell, \gamma_\ell, c_\ell \leftarrow\!\!\$\ \mathbb{Z}_p$ and program the random oracles as

$$\mathsf{H}_0(\mathbf{d}) \coloneqq g^{\gamma_\ell}, \mathsf{H}_1(\mathbf{d}) \coloneqq g^{\beta_\ell},$$

  We also update the map χ as $\chi[\mathbf{d}] \leftarrow c_\ell$.
- if $\mathsf{H}_b(\mathbf{d}) \neq \bot$, we return $\mathsf{H}_b(\mathbf{d})$.

Note that we always do simultaneous programming of both the random oracles H_0 and H_1 upon receiving a query $\mathsf{H}_b(\mathbf{d})$. The c_ℓ value is used to program $\mathsf{H}_{\mathsf{Sig}}$. Recall that thanks to our modifications in the previous game, we can compute the adversarially generated nonces $\{R_j\}_{j \in CS \cap S}$ before receiving them from $\mathcal{A}$. After determining all the $\{R_j\}_{j \in HS \cap S}$ terms, we can determine the aggregate

$$R = \prod_{j \in CS \cap S} R_j \cdot \prod_{j \in HS \cap S} R_j,$$

which allows us to program $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m) \leftarrow \chi[\mathbf{d}]$. We set $\rho[\mathbf{d}] = 1$ right after. If $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m) \neq \bot$ before the latter programming attempt, and $\rho[\mathbf{d}] = 0$, we abort. Conceptually, ρ tracks the session identifiers $\mathbf{d}$ for which we explicitly programmed $\mathsf{H}_{\mathsf{Sig}}$. This is meant to avoid unnecessary aborts if the adversary invokes the same signing session twice.

Recall the value $\chi[\mathbf{d}] = c_\ell$ is sampled uniformly at random and independently for each $\mathbf{d}$. Additionally, since we know that there is at most one $\{R_i\}_{i \in S}$ set for honest signers after the first round, it implies that (i) we extract at most one aggregated nonce R, and (ii) we program $\mathsf{H}_{\mathsf{Sig}}$ at most once with a uniformly random $\chi[\mathbf{d}]$.

With this observation, we can now easily bound the probability of aborting. As mentioned above, we sample the R_i for honest signers after we have computed the nonces R_j from corrupt signers $j \in (CS \cap S)$. Since the nonces R_i for honest signers are sampled uniformly at random, we know that the aggregated nonce R will also be uniformly random and hidden from $\mathcal{A}$ at the time when we program $\mathsf{H}_{\mathsf{Sig}}$. More precisely, because we program before the beginning of round two $\mathcal{A}$ has not received the $\{R_j\}_{j \in HS \cap S}$ terms yet. Therefore, the condition $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m) \neq \bot$ can only hold if $\mathcal{A}$ correctly guessed R beforehand. The guessing probability for a single query is $1/p$, and by a union bound we get that the abort probability is no more than q_{Sig}/p (where q_{Sig} is an upper bound on the number of queries to $\mathsf{H}_{\mathsf{Sig}}$) for a single signing session. Having at most q_s signing sessions, we get through yet another union bound

$$|\Pr[1 \leftarrow \mathsf{Game}_5] - \Pr[1 \leftarrow \mathsf{Game}_4]| \leq \frac{q_s \cdot q_{\mathsf{Sig}}}{p}.$$

Game_6. In this game, we change how we program the random oracles H_0 and H_1. Namely, we first sample a uniformly random $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$ at the beginning of the game. Then, for every new query $\mathsf{H}_b(\mathbf{d}_\ell)$ for either $b \in \{0,1\}$, we sample three random values $\beta_\ell, \gamma_\ell, c_\ell \leftarrow\!\!\$\ \mathbb{Z}_p$, and then program the random oracles as

$$\mathsf{H}_0(\mathbf{d}_\ell) \coloneqq g^{(-\gamma_\ell + \alpha_h \cdot \beta_\ell) \cdot \alpha_v^{-1} - \alpha \cdot c_\ell}, \mathsf{H}_1(\mathbf{d}_\ell) \coloneqq g^{\beta_\ell}.$$

Recall that $h \coloneqq g^{\alpha_h}$ and $v \coloneqq g^{\alpha_v}$ by definition (see Game_0). Again, we update the map χ as $\chi[\mathbf{d}_\ell] \coloneqq c_\ell$, and program $\mathsf{H}_{\mathsf{Sig}}$ as in the previous game. Regarding our game change here, observe that each γ_ℓ is uniformly random and independently sampled of α, β_ℓ and c_ℓ. Further, $\alpha_v \neq 0$, so that $(-\gamma_\ell + \alpha_h \cdot \beta_\ell) \cdot \alpha_v^{-1} - \alpha \cdot c_\ell$ is also uniformly random and independent of α, β_ℓ and c_ℓ. Therefore, $\mathcal{A}$'s view in game Game_6 is identically distributed as its view in game Game_5, and we get $\Pr[1 \leftarrow \mathsf{Game}_5] = \Pr[1 \leftarrow \mathsf{Game}_6]$.

Game_7. In this game, we change the programmed outputs of H_0 and H_1 by correlated values. More specifically, for each query $\mathsf{H}_b(\mathbf{d}_\ell)$ for either $b \in \{0,1\}$ (if not already defined), we program both random oracles H_0 and H_1 as before except that we set $\gamma_\ell := \alpha \cdot \beta_\ell$ instead of a uniformly random γ_ℓ. That is, we sample $\beta_\ell, c_\ell \leftarrow\!\!\$\ \mathbb{Z}_p$ and then program

$$\mathsf{H}_0(\mathbf{d}_\ell) := g^{(-\alpha\cdot\beta_\ell+\alpha_h\cdot\beta_\ell)\cdot\alpha_v^{-1}-\alpha\cdot c_\ell}, \mathsf{H}_1(\mathbf{d}_\ell) := g^{\beta_\ell}.$$

The indistinguishability between games Game_6 and Game_7 is proven assuming the hardness of DDH in the group $\mathbb{G}$. We also rely on the following lemma.

Lemma 2. *Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order p in wihch the* DDH *assumption holds. Then, the distributions*

$$D_0 = (g, \alpha_h, \alpha_v, \{(g^{\beta_i}, g^{-\gamma_i}, c_i)\}_{i\in[q]}) \text{ with } \begin{cases} \alpha \leftarrow\!\!\$\ \mathbb{Z}_p \\ (\beta_i)_{i\in[q]}, (\gamma'_i)_{i\in[q]}, (c_i)_{i\in[q]} \leftarrow\!\!\$\ \mathbb{Z}_p^q \\ \gamma_i = (\gamma'_i - \alpha_h\beta_i)\alpha_v^{-1} + \alpha c_i \end{cases}$$

$$D_1 = (g, \alpha_h, \alpha_v, \{(g^{\beta_i}, g^{-\gamma_i}, c_i)\}_{i\in[q]}) \text{ with } \begin{cases} \alpha \leftarrow\!\!\$\ \mathbb{Z}_p \\ (\beta_i)_{i\in[q]}, (c_i)_{i\in[q]} \leftarrow\!\!\$\ \mathbb{Z}_p^q \\ \gamma_i = (\alpha \cdot \beta_i - \alpha_h\beta_i)\alpha_v^{-1} + \alpha c_i \end{cases}$$

are indistinguishable for any $\alpha_h, \alpha_v \in \mathbb{Z}_p^*$. More precisely, any PPT adversary can distinguish the two distributions with probability at most $\epsilon_{\mathsf{DDH}} + 1/p$, where DDH *is its advantage in the* DDH *security game.*

We defer a formal proof of Lemma 2 to Appendix D. In Game_6 we use a sample from the distribution D_0 to program the random oracles H_0 and H_1, whereas in Game_7 we use a sample from the distribution D_1. Therefore, the advantage of $\mathcal{A}$ in distinguishing between Game_6 and Game_7 is, informally speaking, upper-bounded by the advantage of winning the DDH security game. Formally, we can construct an adversary $\mathcal{B}_{\mathsf{DDH}}$ that runs $\mathcal{A}$ internally, and uses it to beat the DDH game. If we call its advantage $\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda)$ then

$$|\Pr[1 \leftarrow \mathsf{Game}_6] - \Pr[1 \leftarrow \mathsf{Game}_7]| \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda) + \frac{1}{p}.$$

Game_8. This game is identical to its predecessor, except for the way in which we sample α (that we first introduced in Game_2). More precisely, we use $\alpha := \alpha_h + \alpha_v u$ for some $u \leftarrow\!\!\$\ \mathbb{Z}_p$. Since $\alpha_v \neq 0$ and u are uniformly random and independent, α in Game_7 is also uniformly random and independent. Therefore, $\Pr[1 \leftarrow \mathsf{Game}_7] = \Pr[1 \leftarrow \mathsf{Game}_8]$.

Game_9. In this game, we change how we sample the signing keys. We do so by evolving the signing key polynomials from games Game_8 to Game_9. More precisely, let $(x_8(Z), w_8(Z), u_8(Z))$ and $(x_9(Z), w_9(Z), u_9(Z))$ be the signing key polynomials in game Game_8 and Game_9, respectively. Then, in Game_9 we sample the signing key polynomial

$$x_9(Z) := x_8(Z) + \alpha,$$

with α defined in the preceding game. The other two signing key polynomials remain unchanged, i.e.,

$$w_9(Z) := w_8(Z) \quad \text{and} \quad u_9(Z) := u_8(Z).$$

Observe that for any fixed α, since $x_8(Z)$ is a random t-degree polynomial, $x_9(Z) := x_8(Z) + \alpha$ is a random t-degree polynomial too. Hence, $\mathcal{A}$'s view in Game_9 is identically distributed to its view in game Game_8, and thus,

$$\Pr[1 \leftarrow \mathsf{Game}_8] = \Pr[1 \leftarrow \mathsf{Game}_9].$$

Game_{10}. In this game, we change how we sample the signing keys. More precisely, we sample signing key polynomials such that

$$x_{10}(Z) \coloneqq x_8(Z), \quad w_{10}(Z) \coloneqq w_8(Z) + 1, \quad u_{10}(Z) \coloneqq u_8(Z) + u,$$

for the uniformly random $u \in \mathbb{Z}_p$ (introduced in Game_8) we used to define $\alpha \coloneqq \alpha_h + \alpha_v u$.

A key component of our proof is to prove the indistinguishability between $\mathcal{A}$'s view in Game_{10} and Game_9. Proving this follows the strategies and observations of [BDLR25b, BDLR25a]. We rely on the following auxiliary lemma on the statistical indistinguishability of independent random variables and their functions.

Lemma 3 ([DR24]). *Let (X_0, Y_0) and (X_1, Y_1) denote two pairs of random variables. Suppose that for all $\theta \in \{0,1\}$, X_θ and Y_θ are independent. Furthermore, suppose that $X_0 \equiv X_1$ and $Y_0 \equiv Y_1$, where $\equiv$ denotes perfect indistinguishability. Then for any function f*

$$(X_0, Y_0, f(X_0, Y_0)) \equiv (X_1, Y_1, f(X_1, Y_1)).$$

Based on Lemma 3, we can show, that the gaps between game Game_9 and Game_{10} vanishes.

Lemma 4. $\Pr[1 \leftarrow \mathsf{Game}_9] = \Pr[1 \leftarrow \mathsf{Game}_{10}]$.

The proof of this lemma follows [BDLR25b, BDLR25a] and we defer it to Appendix D. Combining the bounds of the above games through a simple triangle inequality, we get

$$\begin{aligned}|\Pr[1 \leftarrow \mathsf{Game}_{10}] - \Pr[1 \leftarrow \mathsf{Game}_0]| \leq{}& \frac{q_{\mathsf{f}}^2 + q_0^2 + q_1^2}{p} + \mathsf{Adv}^{\mathsf{sfzkp}}_{\Pi, \mathcal{B}_{\mathsf{sfzkp}}}(\lambda) \\ &+ q_s \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi, \mathcal{B}_{\mathsf{ss}}}(\lambda) + \frac{q_s \cdot q_{\mathsf{Sig}}}{p} + \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda) + \frac{1}{p}.\end{aligned}$$

Constructing a wrapper $\mathcal{B}$. At this point, we construct a wrapper algorithm $\mathcal{B}$ whose purpose is to run $\mathcal{A}$ inside an environment mimicking the last game. To this end, $\mathcal{B}$ must program the random oracles, simulate corruption and signing queries, and possibly abort exactly as specified in the previous games. As these details are already implicit in their description, we skip them for the sake of readability, and focus on $\mathcal{B}$'s input.

Formally, $\mathcal{B}$ takes as input the public group generators (g, h, v) as well as the three polynomials $(x(X), w(X), u(X))$. We further supply $\mathcal{B}$ with a stream of independent and identically distributed uniformly random hashes $(h_1, \ldots, h_q) \in \mathbb{Z}_p^q$. Said stream is used to program H_{Sig}, and is included to perform forking in the next step of our proof. $\mathcal{B}$ runs $\mathcal{A}$ until the latter outputs a forgery (m^*, σ^*), with $\sigma^* = (R^*, s^*)$, and checks that (m^*, σ^*) is indeed a real forgery. That is, it checks that verification succeeds and m^* is not in the set of previous signing queries. Then, $\mathcal{B}$ identifies the element h_i of the hash stream such that h_i was used to answer the query $\mathsf{H}_{\mathsf{Sig}}(R^*, \mathsf{pk}, m^*)$. It finally returns (i, s^*) if such an index i exists, or $(0, \bot)$ if no i was found or if the forgery is not valid. This output's postprocessing, although perhaps seemingly unmotivated at first sight, is once again justified by the need to apply the forking lemma.

Constructing a DL adversary $\mathcal{C}_{\mathsf{DL}}$. We now leverage the well-known forking lemma by Bellare and Neven [BN06] to turn $\mathcal{B}$ into a DL solver. We first recall the lemma's statement, and then outline its application for our purposes.

Lemma 5 (BN Forking Lemma [BN06]). *Let $q \geq 1$ be an integer. Let $\mathcal{A}$ be a probabilistic algorithm that takes as input a main input inp generated by some probabilistic algorithm $\mathsf{InpGen}()$, elements $h_1, \ldots, h_q$ from some sampleable set H, and random coins from some samplable set $R_{\mathcal{A}}$, and returns either a distinguished failure symbol $\bot$, or a tuple (f, ϕ), where $f \in \{1, \ldots, q\}$ and ϕ is some side output. The accepting probability of $\mathcal{A}$, denoted acc, is defined as the probability (over inp $\leftarrow \mathsf{InpGen}()$, $h_1, \ldots, h_q \leftarrow\!\!\$\ H$, and the random coins of $\mathcal{A}$) that $\mathcal{A}$ returns a non-$\bot$ output. Consider algorithm $\mathsf{Fork}^{\mathcal{A}}_H$ as defined in Figure 5, and let frk be the probability (over inp $\leftarrow \mathsf{InpGen}()$ and the random coins of $\mathsf{Fork}^{\mathcal{A}}_H$) that $\mathsf{Fork}^{\mathcal{A}}_H$ returns a non-$\bot$ output. Then*

$$frk \geq acc \left(\frac{acc}{q} - \frac{1}{|H|} \right).$$

Algorithm $\mathsf{Fork}^{\mathcal{A}}_{H}(inp)$

$\rho \leftarrow\!\$\ R_{\mathcal{A}}\ ;\ h_1, \ldots, h_q \leftarrow\!\$\ H$
$\omega \leftarrow \mathcal{A}(inp, (h_1, \ldots, h_q); \rho)$
if $\omega = \bot$ **then return** $\bot$
$(f, \phi) \leftarrow \omega$
$h'_1, \ldots, h'_q \leftarrow\!\$\ H$
$\omega' \leftarrow \mathcal{A}(inp, (h_1, \ldots, h_{f-1}, h'_f, \ldots, h'_q); \rho)$
if $\omega' = \bot$ **then return** $\bot$
$(f', \phi') \leftarrow \omega'$
if $f \neq f' \vee h_f = h'_f$ **then return** $\bot$
$out \leftarrow (h_f, \phi)\ ;\ out' \leftarrow (h'_f, \phi')$
return (f, out, out')

Fig. 5. Forking algorithm $\mathsf{Fork}^{\mathcal{A}}_{H,R}$ from Lemma 5.

$\mathcal{C}_{\mathsf{DL}}$ leverages the above lemma to fork $\mathcal{B}$, as detailed in Figure 6. Formally speaking, $\mathcal{C}_{\mathsf{DL}}$ is a DLog adversary, and as such it receives a challenge $X_{\mathsf{DL}} \in \mathbb{G}$ whose discrete logarithm in base g it should find. The core idea is to embed X into the public parameters provided to $\mathcal{B}$. More precisely, $\mathcal{C}_{\mathsf{DL}}$ samples a random $\alpha_v \leftarrow\!\$\ \mathbb{Z}_p^*$ and sets $v \leftarrow g^{\alpha_v}$. It then embeds X_{DL} by setting it to be the h generator (i.e., $h \leftarrow X_{\mathsf{DL}}$). $\mathcal{C}_{\mathsf{DL}}$ also generates the three secret polynomials x, w, u at random, but ensuring that $w(0) = 1$ and $u(0) = u$ for a randomly sampled $u \in \mathbb{Z}_p$. Once all the secret and public parameters and keys are available, $\mathcal{C}_{\mathsf{DL}}$ runs the forking construction. This aims at collecting two forgeries from $\mathcal{B}$, to be parsed as $(h_{f_{\text{sig}}}, s)$ and $(h'_{f_{\text{sig}}}, s')$, respectively. At this point, $\mathcal{C}_{\mathsf{DL}}$ combines them into a DL solution through

$$\alpha_h \leftarrow (s - s')/(h_{f_{\text{sig}}} - h'_{f_{\text{sig}}}) - x(0) - u(0) \cdot \alpha_v.$$

We formally define $\mathcal{C}_{\mathsf{DL}}$ in Figure 6.

Algorithm $\mathcal{C}_{\mathsf{DL}}(\mathbb{G}, p, g, X_{\mathsf{DL}})$

$\alpha_v \leftarrow\!\$\ \mathbb{Z}_p^*; (g, h, v) \coloneqq (g, X_{\mathsf{DL}}, g^{\alpha_v})$
$x(X), w(X), u(X) \leftarrow\!\$\ \mathbb{Z}_p[X]_{(t)}$ s.t. $w(0) = 1 \wedge u(0) \leftarrow\!\$\ \mathbb{Z}_p$
$inp_{\mathcal{B}} \leftarrow ((\mathbb{G}, p, g), h, v, x(\cdot), w(\cdot), u(\cdot))$
$\omega \leftarrow \mathsf{Fork}^{\mathcal{B}}_{H}(inp_{\mathcal{B}})$
if $\omega = \bot$ **then return** $\bot$
$(f_{\text{sig}}, out, out') \leftarrow \omega$
$(h_{f_{\text{sig}}}, s) \leftarrow out$
$(h'_{f_{\text{sig}}}, s') \leftarrow out'$
$\alpha_h \leftarrow (s - s')/(h_{f_{\text{sig}}} - h'_{f_{\text{sig}}}) - s(0) - u(0) \cdot \alpha_v$
return α_h

Fig. 6. Algorithm $\mathcal{C}_{\mathsf{DL}}$.

To convince ourselves that this works, we recall that by construction of $\mathcal{B}$ the two outputs $out = (f_{\text{sig}}, s)$ and $out' = (f'_{\text{sig}}, s')$ satisfy

$$g^s = R \cdot \mathsf{pk}^{h_{f_{\text{sig}}}} \quad \text{and} \quad g^{s'} = R' \cdot \mathsf{pk}^{h'_{f_{\text{sig}}}},$$

where the non-primed values are from the primary execution of $\mathcal{B}$ and the primed values are from $\mathcal{B}$'s second execution. As the two executions of $\mathcal{B}$ are identical before the assignments $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m) = h_{f_{\text{sig}}}$ and $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R', m') = h'_{f_{\text{sig}}}$, we have $\mathsf{pk} = \mathsf{pk}'$, $R = R'$ and $m = m'$. Also we have $h'_{f_{\text{sig}}} \neq h_{f_{\text{sig}}}$ by construction of Fork.

Therefore, $(s - s')/(h_{f_{\text{sig}}} - h'_{f_{\text{sig}}})$ is the discrete logarithm of pk. The latter equals $g^{x(0)}h^{w(0)}v^{u(0)} = g^{x(0)}hv^{u(0)}$, which implies that $\alpha_h = (s - s')/(h_{f_{\text{sig}}} - h'_{f_{\text{sig}}}) - x(0) - u(0) \cdot \alpha_v$ defined above is the discrete logarithm of h. This computation is successful as long as the forking construction is successful itself. By Lemma 5, if we call ϵ_{10} the adversary's winning probability in the last game, and $\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda)$ the DLog advantage of $\mathcal{C}_{\mathsf{DL}}$ we get a lower bound

$$\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda) \geq \epsilon_{10} \left(\frac{\epsilon_{10}}{q_{\mathsf{Sig}}} - \frac{1}{p} \right).$$

By reversing the inequality, we derive

$$\epsilon_{\mathsf{Game}_{10}} \leq \frac{q_{\mathsf{Sig}}}{p} + \sqrt{q_{\mathsf{Sig}} \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda)},$$

which yields the claimed bound when combined with the indistinguishability bound between Game_0 and Game_{10}, as well as the fact that $\mathcal{B}$ simulates Game_{10} towards $\mathcal{A}$ perfectly.

Finally, we argue that $\mathcal{C}_{\mathsf{DL}}$ is efficient, as claimed. We start by analyzing $\mathcal{B}$. The latter is efficient by construction, since all the programming and simulation strategies introduced in our game hops are efficiently computable. $\mathcal{C}_{\mathsf{DL}}$ runs two instances of $\mathcal{B}$ according to the forking construction, and performs minimal post-processing over their outputs. Therefore, it is efficient too. This concludes our proof. □

B Fully Adaptive Unforgeability with Key Updates

In this section, we complete the security definition for fully-adaptive threshold signature unforgeability with key updates by providing the extended game description of Figure 7, and then prove in Theorem 3 that our scheme with key updates satisfies it. As hinted before, we upgrade Theorem 2 to the following version, which encompassess the security of our scheme in presence of updates.

Theorem 3 (2-round Adaptive Security with Updates). *Fix $n > t \geq 1$ and a group description $(\mathbb{G}, p, g)$ such that p is a λ-bit prime. Consider an adversary $\mathcal{A}$ making at most $E, q_s, q_{\mathsf{Sig}}, q_0$ and q_1 queries to $\mathsf{OUpdate}, \mathsf{OSR}, \mathsf{H}_{\mathsf{Sig}}, \mathsf{H}_0$ and H_1, respectively. Furthermore, suppose that the adversary makes no more than q_{f} calls to H_{f}, and no more than $d + 1$ calls of the form $\mathsf{OSR}(1, \cdot, \cdot, \cdot, i)$ for each $i \in [n]$ per epoch.*

If $\mathcal{A}$ has an advantage of $\epsilon_{\mathsf{TSAUF\text{-}UP}} = \mathsf{Adv}^{\mathsf{tsauf\text{-}up}}_{\mathsf{2RTSch},\mathcal{A}}(\lambda)$ there exists a PPT algorithm $\mathcal{C}_{\mathsf{DL}}$ having an advantage of $\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda)$ such that

$$\epsilon_{\mathsf{TSAUF\text{-}UP}} \leq \frac{q_{\mathsf{Sig}}}{p} + \sqrt{q_{\mathsf{Sig}} \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda)} + \Delta,$$

for

$$\Delta = \frac{q_{\mathsf{f}}^2 + q_0^2 + q_1^2}{p} + \mathsf{Adv}^{\mathsf{sfzkp}}_{\Pi,\mathcal{B}_{\mathsf{sfzkp}}}(\lambda) + q_s \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda) + \frac{n^2 E}{4} \cdot \mathsf{Adv}^{\mathsf{sfext}}_{\Pi,\mathcal{B}_{\mathsf{sfext}}}(\lambda) + \frac{nE^2}{p} + \frac{q_s \cdot q_{\mathsf{Sig}}}{p} + \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_{\mathsf{DDH}}}(\lambda) + \frac{1}{p}$$

and some PPT algorithms $\mathcal{B}_{\mathsf{sfzkp}}, \mathcal{B}_{\mathsf{ss}}$, and $\mathcal{B}_{\mathsf{DDH}}$. The first two are PPT adversaries against the zero-knowledge and soundness of Π, respectively, and the last one is a PPT adversary against the DDH game.

From a conceptual viewpoint, the proof of Theorem 3 follows rather naturally from the one of Theorem 2. Intuitively, the main gap lies in the fact that our update-free reduction knows the secret f_i polynomials, but $\mathcal{A}$ could update the f_i of corrupted signers to an unknown value. Fortunately, using extractable NIZKs enables an extraction of the committed polynomial by our reduction, and the proof essentially falls back to the update-free case. We modify the previous proofs to cover the case of updates. Most of the modifications take place in the hybrid argument, and they are in turn reflected in the construction of the $\mathcal{B}$ wrapper.

$\mathsf{TSAUF\text{-}UP}^{\mathsf{TS},\lambda,t}_{\mathcal{A}}$

$\mathsf{pp} \leftarrow \mathsf{Setup}(n,t)$
$(\mathsf{pk}, \{pk_i, sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}()$
for $i \in [n], j \in [n]$
  $\mathsf{pkArr}_i[j] \leftarrow \mathsf{pk}_j$
$CS \leftarrow \emptyset, HS \leftarrow [n]$
$Q \leftarrow \emptyset, \mathsf{SessSet} \leftarrow \emptyset$
$\mathsf{ctr} \leftarrow 0$
$\mathsf{OAll} \leftarrow (\mathsf{OCorr}, \mathsf{OSR}_1$
  $\mathsf{OSR}_2, \mathsf{OUpdate})$
$(m', \sigma') \leftarrow \mathcal{A}^{\mathsf{OAll}}(\mathsf{pk}, \{pk_i\}_{i\in[n]})$
return $(m' \notin Q) \wedge \mathsf{Verify}(\mathsf{pk}, m', \sigma')$
  $\wedge\, \mathsf{ctr} < d+1$

$\mathsf{OSR}_1(S, m, i)$

if $i \notin (S \cap HS) \vee S \not\subseteq [n]$:
  return $\perp$
$\mathsf{pm}_{1,i} \leftarrow \mathsf{SR}_1(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}, S, i, m)$
$Q \leftarrow Q \cup \{m\}$
$\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
return $\mathsf{pm}_{1,i}$

$\mathsf{OSR}_2(S, m, i, \{\mathsf{pm}_{1,j}\}_{j\in S})$

if $i \notin (S \cap HS) \vee S \not\subseteq [n]$:
  return $\perp$
$s_i \leftarrow \mathsf{SR}_2(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}, S, i, \{\mathsf{pm}_{1,j}\}_{j\in S}, m)$
return s_i

$\mathsf{OCorr}(i)$

if $|CS| \geq t \vee i \notin [n]$
$\vee\, i \in CS$: **return** $\perp$
$CS \leftarrow CS \cup \{i\}$
$HS \leftarrow HS \setminus \{i\}$
return sk_i

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

$\mathsf{pk} \leftarrow \mathsf{pk}$
$(R, s) \leftarrow \sigma$
$c \leftarrow \mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R, m)$
return $(g^s = R\mathsf{pk}^c)$

$\mathsf{OUpdate}()$

// All honest signers run $\mathsf{Update}_{\mathsf{sk}}$
for $i \in HS$:
  $(\mathsf{sk}_i, \tau_i) \leftarrow \mathsf{Update}_{\mathsf{sk}}(\mathsf{sk}_i)$
// Get malicious update tokens
// $\mathcal{A}$ sends a set of tokens to each signer
for $i \in HS$:
  $\{\tau_j\}_{j\in CS} \leftarrow \mathcal{A}(\{\tau_j\}_{j\in HS})$
// Check update tokens from other users
  for $j \in [n]$:
    $\mathsf{pkArr}_i[j] \leftarrow \mathsf{Update}_{\mathsf{pk}}(\mathsf{pkArr}_i[j], \tau_j)$
// If $\mathsf{Update}_{\mathsf{pk}}$ returns $\perp$, we abort
// We skip the formal description
// for the sake of readability
$\mathsf{ctr} \leftarrow 0$

Fig. 7. The adaptive unforgeability security game with updates. The differences w.r.t. Figure 2 are highlighted in cyan.

Proof (Proof of Theorem 3). We start with a sequence of indistinguishable games akin to those provided in the beginning of the proof of Theorem 2.

$\overline{\mathsf{Game}}_0$. This is once again the original security game for our 2RTSch scheme, where the challenger answers all update and signing queries by following the protocol specification honestly. By definition

$$\mathsf{Adv}^{\mathsf{tsauf\text{-}up}}_{\mathcal{A},\mathsf{2RTSch},\lambda,t}(\lambda) = \Pr\big[1 \leftarrow \overline{\mathsf{Game}}_0\big] = \epsilon_{\mathsf{TSAUF\text{-}UP}}.$$

$\overline{\mathsf{Game}}_1$. We again abort if there is a collision for H_f, or for either of H_0 and H_1, as detailed in Game_1. The gap between this game and its predecessor is identical to that between Game_1 and Game_0, i.e.

$$\big|\Pr\big[1 \leftarrow \overline{\mathsf{Game}}_0\big] - \Pr\big[1 \leftarrow \overline{\mathsf{Game}}_1\big]\big| \leq \frac{q_\mathsf{f}^2 + q_0^2 + q_1^2}{p}.$$

$\overline{\mathsf{Game}}_2$. As previously done, we replace the NIZK proofs generated by honest signers in the first round with simulated ones, by properly interacting with $\Pi.\mathcal{S}_1$ and $\Pi.\mathcal{S}_2$ when generating proofs for each $i \in HS$. Furthermore, we change how the NIZK proofs of honest users are generated at update time, by replacing them with simulated ones too.

The gap between $\overline{\mathsf{Game}}_2$ and $\overline{\mathsf{Game}}_1$ is bounded above by the zero-knowledge of Π. More precisely, we can again construct an adversary $\mathcal{B}_{\mathsf{sfzkp}}$ that distinguishes simulated Π-proofs from real ones by running $\mathcal{A}$ internally. The details are identical to those provided in Game_2.

$$\big|\Pr\big[1 \leftarrow \overline{\mathsf{Game}}_1\big] - \Pr\big[1 \leftarrow \overline{\mathsf{Game}}_2\big]\big| \leq \mathsf{Adv}^{\mathsf{sfzkp}}_{\Pi,\mathcal{B}_{\mathsf{sfzkp}}}(\lambda).$$

$\overline{\mathsf{Game}}_3$. $\overline{\mathsf{Game}}_3$ is identical to $\overline{\mathsf{Game}}_2$ except that the secret nonce r_i from honest signer i, which is computed by evaluating $\mathsf{f}_i(\mathsf{H}_\mathsf{f}(\mathbf{d}))$ in $\overline{\mathsf{Game}}_3$, is drawn uniformly at random in $\mathbb{Z}_p$ and stored in a table $T_r(i,\cdot,\cdot)$. Unlike the previous reduction, this time $T_r(i,\mathbf{d},e)$ returns the r_i value that was used in epoch $e \in [E]$ with session id $\mathbf{d}$ (this way, the same integer r_i is used if signing on $\mathbf{d} = (m,S)$ is called again to signer i in epoch e). We still leverage lagrangian interpolation to manage adaptive corruptions: upon corruption of signer i in epoch e we inspect the content of $T_r(i,\cdot,e)$ and run lagrangian interpolation over it to reconstruct the f_i polynomial of the corresponding epoch. If not enough input-output tuples are provided we append random ones until we match the desired degree of f_i. Once again, assuming every f_i polynomial is evaluated less than $\deg(\mathsf{f}_i)+1$ times *per epoch*, and each epoch has a distinct f_i, lagrangian interpolation is always feasible, and yields a consistent polynomial with probability 1. Furthermore, we also modify the key generation as done in the previous proof, to endow the challenger with the trapdoor td and enable opening the polynomial commitments of honest signers to arbitrary values (including, the interpolated polynomial). We also generate $\mathsf{C}_{\mathsf{f}_i}$ of every honest user by committing to 0, in both KeyGen and Update. As argued before for Game_3 and Game_2

$$\Pr\big[1 \leftarrow \overline{\mathsf{Game}}_2\big] = \Pr\big[1 \leftarrow \overline{\mathsf{Game}}_3\big].$$

$\overline{\mathsf{Game}}_4$. As done in Game_4, we design this game to compute on our own all the nonce values R_i''s that we expect to receive from the adversary. For the first epoch, this can be done as outlined in Game_4. To deal with multiple epochs, we leverage the simulation extractability of the NIZK that we used to generate updates. In particular, the proof of knowledge π_i' (see $\mathsf{Update}_{\mathsf{sk}}$ in Figure 3) is generated from an extractable NIZK, and we leverage the corresponding extractor to obtain the committed polynomial f_i. If the extraction fails, we abort. As we did before, we check that the maliciously generated R_j terms in round two correspond to the ones we pre-computed. We note that since $\mathcal{A}$ sends the update tokens to honest users independently (line 7 of Figure 7), it could in principle send distinct and inconsistent commitments to all of them. Because this would lead to different partial public keys, and the latter are incorporated in the $\mathbf{d}$ identifier, we keep a table T_{ext} indexed by $\mathbf{d}$, such that

$$T_{ext}[\mathbf{d}] = \{\mathsf{f}_i\}_{i \in CS}$$

contains all the extracted polynomials f_i corresponding to $\mathbf{d}$. Then, whenever $\mathcal{A}$ triggers the first round of a signing session identified by $\mathbf{d}$, we use the content of $T_{ext}[\mathbf{d}]$ to compute all the $\{R_i\}_{i \in S \cap CS}$ nonces, and in turn the aggregate R corresponding to that identifier.

To bound the gap between the two games, we analyze the probability of abortion. At first, consider the probability of aborting due to an extraction failure. In a single epoch $e \in [E]$, assuming $t_c(e) < n$ corrupted users, we extract at most $t_c(e)$ times for each honest signer, for a total of $t_c(e)(n - t_c(e))$ times per epoch. The latter is upper bounded by $n^2/4$[7], and we have at most E epochs, leading to a probability of at most $n^2E/4 \cdot \mathsf{Adv}^{\mathsf{sfext}}_{\Pi,\mathcal{B}_{\mathsf{sfext}}}(\lambda)$. Crucially, notice that our definition of $\mathcal{R}_{\mathsf{opn}}$ in Section 4 incorporates $\mathbf{d}$ and the signer index i into the public statement $\mathbb{x}$. This is meant to bind a proof to an epoch (through $\mathbf{d}$, which carries epoch-specific information through the partial public keys), and signer within that epoch (through i). In particular, this prevents $\mathcal{A}$ from recycling proofs issued by other signers in previous epochs, as they pertain to a different statement. Because of this countermeasure, we can safely reduce to simulation extractability here. As argued for Game_4, this happens with probability no more than $q_{\mathsf{Sig}} \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda)$. By a union bound

$$\left|\Pr\left[1 \leftarrow \overline{\mathsf{Game}}_3\right] - \Pr\left[1 \leftarrow \overline{\mathsf{Game}}_4\right]\right| \leq q_{\mathsf{Sig}} \cdot \mathsf{Adv}^{\mathsf{ss}}_{\Pi,\mathcal{B}_{\mathsf{ss}}}(\lambda) + \frac{n^2E}{4} \cdot \mathsf{Adv}^{\mathsf{sfext}}_{\Pi,\mathcal{B}_{\mathsf{sfext}}}(\lambda).$$

$\overline{\mathsf{Game}}_5$. At this stage, one is tempted to faithfully reproduce Game_5. Unfortunately, this is unfeasible due to a core gap between the refreshable and update-free constructions. In the latter, the f_i factors are fixed at the beginning, and never updated. It follows that given an identifier $\mathbf{d}$, there is at most one corresponding valid nonce R. In the refreshable case, things are not as straightforward. In principle, the updates could generate a session identifier $\mathbf{d}'$ that equals a previously used $\mathbf{d}$, but corresponds to distinct polynomials f_i. This is due to the perfectly hiding Pedersen polynomial commitment that honest signers employ to generate $\mathsf{C}_{\mathsf{f}_i}$. If such an event occurs, blindly adopting the programming strategy of Game_5 would yield inconsistent behavior. This is true because if the f_i polynomials change, then with overwhelming probability the aggregate nonce R' differs from R too, and in turn $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R', m)$ should not equal $\chi[\mathbf{d}]$.

Therefore, we rule out this possibility by introducing ad-hoc changes in $\overline{\mathsf{Game}}_5$. We start from the following observations:

- because of the calls $\mathsf{CheckConsist}\mathbf{d}$ in the first round, each honest signer generates the R_i term corresponding to identifier $\mathbf{d}$ iff the latter is consistent with its array $\mathsf{pkArr}_i[\cdot]$ at that point in time.
- the programming $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R', m) \leftarrow \chi[\mathbf{d}]$ only occurs if all the honest signers terminate the first round of session identified by $\mathbf{d}$.
- suppose that the following condition is guaranteed: as long as a signer i is honest, its commitments f_i are all distinct.
- because the partial public keys incorporate the $\mathsf{C}_{\mathsf{f}_i}$ commitments, this would imply distinct partial public keys (across different epochs) for each honest signer.
- then, because $\mathbf{d}$ incorporates partial public keys, there would be no later attempt to program $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R', m) \leftarrow \chi[\mathbf{d}]$.

Therefore, in this game we enforce exactly the aforementioned condition, i.e., the absence of collisions in Pedersen commitments. Notice that each Pedersen commitment yields a uniformly random element of $\mathbb{G}$. The collision probability for a single user is at most E^2/p, and by union bound over the n users

$$\left|\Pr\left[1 \leftarrow \overline{\mathsf{Game}}_4\right] - \Pr\left[1 \leftarrow \overline{\mathsf{Game}}_5\right]\right| \leq \frac{nE^2}{p}$$

$\overline{\mathsf{Game}}_6$ *to* $\overline{\mathsf{Game}}_{11}$. These games introduce changes that are identical to those introduced by their update-free counterparts (i.e., games Game_5 to Game_{10}). The indistinguishability bounds and the corresponding arguments are essentially the same, and we omit them for the sake of brevity.

By triangular inequality, we get the final bound

$$\left|\Pr\left[1 \leftarrow \overline{\mathsf{Game}}_{10}\right] - \Pr\left[1 \leftarrow \overline{\mathsf{Game}}_0\right]\right| \leq \Delta,$$

with Δ defined exactly as in the main body of Theorem 3.

[7] That is a consequence of the GM-AM inequality.

Wrapper $\mathcal{B}$. We can again construct a wrapper $\mathcal{B}$ that runs $\mathcal{A}$ internally, and simulates its interaction with the challenger in $\overline{\mathsf{Game}}_{10}$.

Once again, $\mathcal{B}$ takes as input the public group generators (g, h, v), the three polynomials $x(X), w(X), u(X)$, and a stream of uniformly random hashes $(h_1, \ldots, h_q) \in \mathbb{Z}_p^q$ to program H_{Sig}. Akin to the previous proof, $\mathcal{B}$ runs $\mathcal{A}$ until the latter outputs a forgery (m^*, σ^*), with $\sigma^* = (R^*, s^*)$. Then, it identifies the element h_i of the hash stream such that h_i was used to answer the query $\mathsf{H}_{\mathsf{Sig}}(R^*, \mathsf{pk}, m^*)$. It finally returns (i, s^*) if such an index i exists, or $(0, \perp)$ if no i was found or if the forgery is not valid.

Forking. Finally, we apply Lemma 5 to $\mathcal{B}$, and construct a DLog adversary $\mathcal{C}_{\mathsf{DL}}$ akin to the previous one. $\mathcal{C}_{\mathsf{DL}}$ collects the output of the forking construction, and postprocess it to solve the DLog challenge. The specification is identical to the previous one, and can be found in Figure 6. The forking lemma relates $\overline{\epsilon}_{10} = \Pr\left[1 \leftarrow \overline{\mathsf{Game}}_{10}\right]$ to the probability of successful forking. It once again holds

$$\mathsf{Adv}^{\mathsf{dl}}_{\mathcal{C}_{\mathsf{DL}}}(\lambda) \geq \overline{\epsilon}_{10} \left(\frac{\overline{\epsilon}_{10}}{q_{\mathsf{Sig}}} - \frac{1}{p} \right).$$

Reversing the inequality, and combining it with $\left|\Pr\left[1 \leftarrow \overline{\mathsf{Game}}_{10}\right] - \Pr\left[1 \leftarrow \overline{\mathsf{Game}}_{0}\right]\right| \leq \Delta$ yields the claimed bound.

While deriving $\mathcal{C}_{\mathsf{DL}}$'s success probability does not seem conceptually harder than in the update-free case, the combination of forking and rewinding-based extraction could introduce non-trivial problems which call for an extended discussion. Before diving into their description and mitigation, we point out that said problems can be avoided entirely by making the NIZK *straightline extractable*, e.g. by leveraging Fischlin's transform [Fis05].

The main concern is due to $\mathcal{B}$ using a rewinding witness extractor to extract the adversarially generated f_i polynomials at update time. To extract a polynomial of degree d, we must rewind $d+1$ times in total, each time supplying a fresh random challenge. Conceptually, we can visualize this as splitting the execution flow of $\mathcal{A}$ (inside $\mathcal{B}$) into $d+1$ parallel branches, each corresponding to a random challenge. After collecting all of $\mathcal{A}$'s responses, the extractor can do its magic and recover f_i. Noticeably, most of the $d+1$ branches become superfluous at this point. In truth, *all but one* of them can be safely pruned, and we just keep one (say, the first) as the main execution of $\mathcal{A}$ within $\mathcal{B}$. This spares us the load of performing nested branching, which could otherwise lead to an exponentially large number of branched executions. Furthermore, when used in combination with forking, there are some pathological cases that should be addressed. Our application of Lemma 5 forks the execution of $\mathcal{B}$ exactly at the point in time where $\mathcal{A}$ makes the query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$ that it uses to generate the forgery. It could happen that said query occurs exactly when we need to rewind, i.e., during the call $\mathcal{A}(\{\tau_j\}_{j \in HS})$ in line 7 of Figure 7. To ensure that the forking lemma applies, we combine it with the rewinding-based extractor by adopting the following strategy:

- For $i \in [d+1]$, let $X_1(i), \ldots, X_L(i)$ denote the inputs that $\mathcal{A}$ supplies to $\mathsf{H}_{\mathsf{Sig}}$ in the i-th branch *after* making the query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$. In principle, L is also parametrized by i, but one can safely ignore it by suitably upper bounding $\max_i L_i$, e.g. by setting $L = q_{\mathsf{Sig}}$.
- More generally, if no $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$ query took place in the sequence to be rewound, let $X_1(i), \ldots, X_L(i)$ just denote all of the inputs that $\mathcal{A}$ supplies to $\mathsf{H}_{\mathsf{Sig}}$ in the i-th branch.
- **Case A:** We rewind before forking, and no query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$ is made. Then, whenever $\mathcal{B}$ receives query $\mathsf{H}_{\mathsf{Sig}}(X_j(i))$ such that $T_{\mathsf{Sig}}[X_j(i)] = \perp$, it simply moves forward in its hash stream. .
- **Case B:** The query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$ is made during a rewinding sequence. Then, sticking to the notation of the first bulletpoint, suppose that $f - 1 \in [q_{\mathsf{Sig}}]$ is the index of the last hash stream element h_{f-1} that was used by $\mathcal{B}$ to answer a $\mathsf{H}_{\mathsf{Sig}}$ query before $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$. As prescribed by the forking lemma, $\mathcal{B}$ answers the latter call with h_f. For any call $\mathsf{H}_{\mathsf{Sig}}(X_j(i))$ received by $\mathcal{B}$ after that, $\mathcal{B}$ checks if $T_{\mathsf{Sig}}[X_j(i)] = \perp$, and if so it programs with the next element of the hash stream. That is, consider the sequence

  $$X_1(1), \ldots, X_L(1), X_1(2), \ldots, X_L(2), \ldots, X_1(d+1), \ldots, X_1(d+1),$$

  , and remove all duplicate entries from it, obtaining some sequence $Y_1, \ldots, Y_{L'}$. Then, $\mathcal{B}$ uses the hash stream element h_{f+i} to program $\mathsf{H}_{\mathsf{Sig}}(Y_i)$ for all $i \in [L']$.

– **Case C:** We rewind after forking, and no query $\mathsf{H}_{\mathsf{Sig}}(\mathsf{pk}, R^*, m^*)$ is made. The case is managed analogously to case A.

Finally, to avoid confusion, we remark that the randomness used to program the random oracle in the $d+1$ rewound executions is independent of the main hash stream, and is encompassed by the common randomness ρ, if we follow the notation of Lemma 5.

C Additional Figures

1. $\mathcal{P}$ chooses random $k_r, k_{\rho_r}, k_u, k_w, k_x \in \mathbb{Z}_q$ and computes:

$$T_1 = g^{k_r} g_\rho^{k_{\rho_r}}$$
$$T_2 = g^{k_r} h_1^{k_w} h_2^{k_u}$$
$$T_3 = g^{k_x} h^{k_w} v^{k_u}$$

   $\mathcal{P}$ sends to $\mathcal{V}$: (T_1, T_2, T_3)
2. $\mathcal{V}$ sends to $\mathcal{P}$ a uniform random challenge $c \in \mathbb{Z}_p$.
3. $\mathcal{P}$ computes:

$$z_r = k_r + c \cdot r$$
$$z_{\rho_r} = k_{\rho_r} + c \cdot r_r$$
$$z_u = k_u + c \cdot u$$
$$z_w = k_w + c \cdot w$$
$$z_x = k_x + c \cdot x$$

   $\mathcal{P}$ sends to $\mathcal{V}$: $(z_r, z_{\rho_r}, z_u, z_w, z_x)$
4. $\mathcal{V}$ checks that:

$$g^{z_r} g_\rho^{z_{\rho_r}} = T_1 \cdot \mathsf{C}_r^c$$
$$g^{z_r} h_1^{z_w} h_2^{z_u} = T_2 \cdot R^c$$
$$g^{z_x} h^{z_w} v^{z_u} = T_3 \cdot \mathsf{C}^c$$

   If all checks pass, $\mathcal{V}$ accepts. Otherwise, $\mathcal{V}$ rejects.

Fig. 8. An instantiation of our Sigma protocol to check that the prover knows r, ρ_r, u, w, x such that: $R = g^r h_1^w h_2^u$, $\mathsf{C}_r = g^r g_\rho^{\rho_r}$ and $\mathsf{C} = g^x h^w v^u$.

D Proofs of Lemmas

Proof (Proof of Lemma 2). We borrow the original argument of [BDLR25b] and summarize it here for the sake of self-containment. Let us introduce auxiliary distributions

$$C_0 = (g, g^\alpha, \{(g^{\beta_i}, g^{\gamma_i})\}_{i\in[q]}) \text{ with } \begin{cases} \alpha \leftarrow\!\$\, \mathbb{Z}_p \\ (\beta_i)_{i\in[q]}, (\gamma_i)_{i\in[q]} \leftarrow\!\$\, \mathbb{Z}_p^q \end{cases}$$
$$C_1 = (g, g^\alpha, \{(g^{\beta_i}, g^{\alpha\beta_i})\}_{i\in[q]}) \text{ with } \begin{cases} \alpha \leftarrow\!\$\, \mathbb{Z}_p \\ (\beta_i)_{i\in[q]} \leftarrow\!\$\, \mathbb{Z}_p^q \end{cases}.$$

Suppose $\mathcal{A}$ can distinguish them with advantage ϵ. Then, as shown by Naor and Reingold [NR97], one can construct a PPT adversary $\mathcal{A}'$ whose advantage in the DDH game is at least $\epsilon - 1/p$. Hence, under the DDH assumption C_0 and C_1 are distinguishable with probability at most $\epsilon_{\mathsf{DDH}} + 1/p$.
Next, we show that the indistinguishability of C_1 and C_1 implies that of D_0 and D_1. Assume you receive a sample $(g, w, \{(u_i, v_i)\}_{i\in[q]})$ from either C_0 or C_1. Then, consider the transformation

$$\begin{aligned}(c_1, \ldots, c_q) &\leftarrow\$ \mathbb{Z}_p\\ z_i \leftarrow (v_i \cdot u_i^{-\alpha_h})^{\alpha_v^{-1}} &\cdot w^{c_i} \text{ for all } i \in [q]\\ (g, w, \{(u_i, v_i)\}_{i\in[q]}) &\rightarrow (g, \alpha_h, \alpha_v, \{(u_i, z_i^{-1}), c_i\}_{i\in[q]})).\end{aligned}$$

You can verify that this maps a sample of C_b to one of D_b for any $b \in \{0, 1\}$. Hence, discriminating D_0 and D_1 allows to discriminate C_0 and C_1 with the same advantage. □

Proof (Proof of Lemma 4). We leverage Lemma 3, combined with Patarin's H-coefficient technique [Pat09], to prove indistinguishability. We briefly recap the latter, and then proceed to its application. Let τ denote a possible transcript of an adversary $\mathcal{A}$, and let $p_7(\tau)$ and $p_6(\tau)$ denote the probabilities of obtaining transcript τ *when $\mathcal{A}$'s corruption queries, random oracle queries, and signing queries are fixed in advance* in Game_7 and Game_6, respectively. Then, to prove indistinguishability of the two games it suffices to show that $p_7(\tau) = p_6(\tau)$ for all possible values of τ.

Following the syntax of Lemma 3, which we leverage to achieve the previous goal, we introduce the random variables

$$X_0 = (\alpha_h, \alpha_v, x, u, (x_6(i), w_6(i), u_6(i))_{i\in CS}, (c_j, \beta_j, (R_{i,j}, s_{i,j})_{i\in HS\cap S})_{j\in[q_s]})$$

$$X_1 = (\alpha_h, \alpha_v, x, u, (x_7(i), w_7(i), u_7(i))_{i\in CS}, (c_j, \beta_j, (R_{i,j}, s_{i,j})_{i\in HS\cap S})_{j\in[q_s]}),$$

both denoting matching parts of $\mathcal{A}$'s transcript in the two games. Y_0 (respectively Y_1) denotes the random variable for:

1. the outputs of $\mathsf{H}_{\mathsf{Sig}}$ on all inputs except those where the game programs $\mathsf{H}_{\mathsf{Sig}}$ by extracting the combined nonce R during any signing session;
2. all $(r_{i,j})_{i,j}$ values for $i \in CS \cap S_j$ that the game samples for the j-th signing session (where the signer set is S_j) before $\mathcal{A}$ corrupts the signer i;
3. the simulated NIZK proofs of all honest signers;
4. the secret polynomials of corrupted signers $\{\mathsf{f}_i\}_{i\in CS}$.

It is easy to see that Y_0 (respectively Y_1) is independent of X_0 (respectively X_1). Moreover, due to the description of games Game_7 and Game_6, Y_0 is identically distributed as Y_1. We now show that for any fixed queries of $\mathcal{A}$, given either (X_0, Y_0) (or (X_1, Y_1)), the remainder of the transcript is determined by a deterministic function of (X_0, Y_0) (or (X_1, Y_1)). We also show that both games apply a common function $f(\cdot, \cdot)$, which we describe below for $\theta \in \{0, 1\}$:

- Define $\alpha := \alpha_h + u \cdot \alpha_v$. The H_0 outputs on the session identifier $\mathbf{d} = (m, S)$ of all signing sessions are a deterministic function of $(\alpha_h, \alpha_v, \alpha, (\beta_j, c_j)_{j\in[qs]}, Y_\theta)$.
- The discrete logarithm of the public key is identical in both games, and is given by $x_5(0) + \alpha$. More precisely, $\mathsf{pk}_{\mathsf{Game}_6} = g^{x_5(0)+\alpha}$ by definition. Also, since $w_7(0) = 1$ and $u_7(0) = u$

$$\mathsf{pk}_{\mathsf{Game}_7} = g^{x_7(0)} h^{w_7(0)} v^{u_7(0)} = g^{x_5(0)} h v^u = g^{x_5(0)+\alpha_h+\alpha_v u} = g^{x_5(0)+\alpha}.$$

  Since $|CS| = t$, the threshold public keys of all signers are a deterministic function of $x_5(0) + \alpha$ and the signing keys of signers in CS from X_θ.
- The combined nonces and final signatures are a deterministic function of $(c_j, (R_{i,j}, s_{i,j})_{i\in S_j\cap HS})_{j\in[q_s]}$, the signing keys of the corrupt signers, $\mathcal{A}$'s internal state, and Y_θ.

It now remains to show that X_0 and X_1 are identically distributed. Let

$$\tau = (\alpha'_h, \alpha'_v, x', u', (x'_i, w'_i, u'_i)_{i\in CS}, (c'_j, \beta'_j, (R'_{i,j}, s'_{i,j})_{i\in HS\cap S_j})_{j\in[q_s]})$$

denote a generic value that X_0 and X_1 can take. Let us analyze $\Pr X_b = \tau$ for $b = 0$ and $b = 1$ independently, starting with the first case. At first, consider the event

$$(\alpha_h, \alpha_v, x, u, (x_6(i), w_6(i), u_6(i))_{i\in CS}) = (\alpha'_h, \alpha'_v, x', u', (x'_i, w'_i, u'_i)_{i\in CS}).$$

Taking into account the definition of polynomials given in Game_6, we can rewrite this as

$$(\alpha_h, \alpha_v, x, u, (x_5(i), w_5(i), u_5(i))_{i\in CS}) = (\alpha'_h, \alpha'_v, x' - \alpha, u', (x'_i - \alpha, w'_i, u'_i)_{i\in CS}).$$

Let us break down this tuple into independent subtuples, and estimate their distributions one by one. Because α_h, α_v, and u are uniformly random and independent, each corresponding triple of possible realizations $(\alpha'_h, \alpha'_v, u')$ occurs with probability $p^{-1}(p-1)^{-2}$.

Then, because $\mathcal{A}$ makes exactly t corruptions (see Game 1) and x_5 has degree t, the $t+1$ points $(x' - \alpha, \{x'_i - \alpha\}_{i\in CS})$ uniquely determine it. More precisely, there exists a unique polynomial $P(X) \in \mathbb{Z}_p[X]$ such that $P(0) = x' - \alpha$ and $P(i) = x'_i - \alpha$ for all $i \in CS$. Thus, x_5 must equal $P(X)$, which happens with probability $\left|\mathbb{Z}_p[x]_{(t)}\right|^{-(t+1)} = p^{-(t+1)}$. Because w_5 and u_5 both have null constant term and degree t, the t values $(w'_i, u'_i)_{i\in CS}$ suffice to uniquely determine them. Hence, by a similar argument, the tuple $(w'_i, u'_i)_{i\in CS}$ occurs with probability p^{-2t}.

The c_j and β_j values are also uniform and independent, thus each tuple $(c'_j, \beta'_j)_{i\in HS\cap S_j}$ is equiprobable with probability density p^{-2q_s}.

Now, consider the distribution of $(R_{i,j}, s_{i,j})_{i\in HS\cap S_j, j\in[q_s]}$ conditioned on the event that all other components of X_0 are fixed and equal to the generic values reported in τ. By taking into account all the modification to the original game, one can verify that

$$R_{i,j} = g^{(r_{i,j} - \beta_j u\cdot w_5(i) - \alpha c_j w_5(i) + \beta_j u_5(i))\Lambda_i(S_j)},$$
$$s_{i,j} = (r_{i,j} + c_j \cdot (x_5(i) + \alpha))\Lambda_i(S_j).$$

Note that in this scenario, the $R_{i,j}$ and $s_{i,j}$ terms are almost deterministic, and the only source of stochasticity is the dependence on the random variables $r_{i,j}$. Such variables are sampled by honest signers, and they are all uniformly random over $\mathbb{Z}_p$ and independent. Let us focus on fixed indices (i, j). With other components fixed as assumed, not all $(R'_{i,j}, s'_{i,j})$ pairs are admissible. Specifically, if we set $R'_{i,j} = g^{e_{i,j}}$ then the following linear system

$$L_0^{i,j} : \begin{cases} (r_{i,j} - \beta_j u \cdot w_5(i) - \alpha c_j w_5(i) + \beta_j u_5(i))\Lambda_i(S_j) = e_{i,j} \\ (r_{i,j} + c_j \cdot (x_5(i) + \alpha))\Lambda_i(S_j) = s'_{i,j} \end{cases}.$$

in the unknown $r_{i,j}$ must admit a solution. Thus, each $(R'_{i,j}, s'_{i,j})_{i\in HS\cap S_j, j\in[q_s]}$ for which $L_0^{i,j}$ holds for all $i \in HS \cap S_j, j \in [q_s]$ occurs with the same probability

$$\prod_{j\in[q_s]} p^{-|S_j|} = p^{-k},$$

where $k = \sum_{j\in[q_s]} |S_j|$. Combining everything, each generic τ has probability $(p-1)^{-2} \cdot p^{-(2+3t+2q_s+k)}$ of being generated.

Now, let us focus on X_1. As before, the values $(\alpha'_h, \alpha'_v, u')$ occur with probability $p^{-1}(p-1)^{-2}$. Furthermore, by recycling the previous arguments on polynomials x_7, w_7, u_7 and samples $(c'_j, \beta'_j)_{i\in HS\cap S_j}$, and combining it with the aforementioned density, we get that each tuple $(\alpha'_h, \alpha'_v, x', u', (x'_i, w'_i, u'_i)_{i\in CS}, (c'_j, \beta'_j)_{j\in[q_s]})$ occurs with probability $p^{-(2+3t+2q_s)}$. Next, we again evaluate the distribution of $(R_{i,j}, s_{i,j})_{i\in HS\cap S_j, j\in[q_s]}$ conditioned on the event that all other components of X_1 are fixed. This time the coefficients equal

$$R_{i,j} = g^{(r_{i,j} + \beta_j u(w_5(i)+1) - \alpha c_j(w_5(i)+1) + \beta_j(u_5(i)+u))\Lambda_i(S_j)},$$
$$s_{i,j} = (r_{i,j} + c_j \cdot x_5(i))\Lambda_i(S_j).$$

We see that, once again, the only source of randomness is the set of coefficients $r_{i,j}$. By recycling the previous argument we derive an analogous admissibility condition

$$L_1^{i,j} : \begin{cases} (r_{i,j} + \beta_j u(w_5(i)+1) - \alpha c_j (w_5(i)+1) + \beta_j (u_5(i)+u))\Lambda_i(S_j) = e_{i,j} \\ (r_{i,j} + c_j \cdot x_5(i))\Lambda_i(S_j) = s'_{i,j} \end{cases} .$$

Since the latter are uniformly random, each admissible tuple $(R'_{i,j}, s'_{i,j})_{i \in HS \cap S_j, j \in [q_s]}$ still occurs with probability p^{-k}, with k defined as in the previous case. Finally, we claim that conditions $L_0^{(i,j)}$ and $L_1^{(i,j)}$ are equivalent for all i, j. Indeed, one can verify that the latter is obtained from the first by subtracting term αc_j to both equations. Thus, by combining everything we conclude that any admissible τ has density $(p-1)^{-2} \cdot p^{-(2+3t+2q_s+k)}$, and that the sets of admissible τ values under X_0 and X_1 coincide. □

D.1 Correctness of our Scheme

Fix a signing set $S \subseteq [n]$ with $|S| \geq t$ and let Λ_i denote the Lagrange coefficient at 0 for index $i \in S$. Each signer i publishes

$$R_i \;=\; \big(g^{r_i}\, \mathsf{H}_0(\mathbf{d})^{\overline{w}_i}\, \mathsf{H}_1(\mathbf{d})^{\overline{u}_i}\big)^{\Lambda_i}, \qquad s_i \;=\; (r_i + c\,\overline{x}_i)\,\Lambda_i,$$

where $r_i = \mathsf{f}_i(\mathsf{H}_\mathsf{f}(\mathbf{d}))$ and $c = \mathsf{H}_\mathsf{Sig}(\mathsf{pk}, R, m)$ with $R = \prod_{j \in S} R_j$.

First, aggregate the nonces:

$$R \;=\; \prod_{i \in S} R_i \;=\; \prod_{i \in S} \big(g^{r_i}\, \mathsf{H}_0(\mathbf{d})^{\overline{w}_i}\, \mathsf{H}_1(\mathbf{d})^{\overline{u}_i}\big)^{\Lambda_i} \;=\; g^{\sum_i \Lambda_i r_i}\, \mathsf{H}_0(\mathbf{d})^{\sum_i \Lambda_i \overline{w}_i}\, \mathsf{H}_1(\mathbf{d})^{\sum_i \Lambda_i \overline{u}_i}.$$

Because w and u are degree-t polynomials with zero constant term, Lagrange interpolation at 0 over any set of size at least t gives

$$\sum_{i \in S} \Lambda_i\, \overline{w}_i \;=\; w(0) \;=\; 0 \qquad \text{and} \qquad \sum_{i \in S} \Lambda_i\, \overline{u}_i \;=\; u(0) \;=\; 0.$$

Hence the H_0- and H_1-factors vanish, and we obtain

$$R \;=\; g^r \quad \text{with} \quad r \leftarrow \sum_{i \in S} \Lambda_i\, \mathsf{f}_i(\mathbf{d}).$$

Next, aggregate the partial responses:

$$s \;=\; \sum_{i \in S} s_i \;=\; \sum_{i \in S} \Lambda_i\,(r_i + c\,\overline{x}_i) \;=\; \underbrace{\sum_i \Lambda_i\, r_i}_{=\,r} \;+\; c\, \underbrace{\sum_i \Lambda_i\, \overline{x}_i}_{=\,x(0)} \;=\; r + c\,x(0),$$

where we used that x is a degree-t polynomial, so $\sum_i \Lambda_i \overline{x}_i = x(0)$.

Finally, recall that the global public key is

$$\mathsf{pk} \;=\; g^{x(0)}\, h^{w(0)}\, v^{u(0)} \;=\; g^{x(0)}$$

since $w(0) = u(0) = 0$. Verification computes $c = \mathsf{H}_\mathsf{Sig}(m, R, \mathsf{pk})$ and checks

$$\mathsf{pk}^{\,c} \cdot R \;=\; \big(g^{x(0)}\big)^c \cdot g^r \;=\; g^{r + c\,x(0)} \;=\; g^s,$$

which holds by the equality derived above. Therefore every honestly produced signature $\sigma = (R, s)$ verifies. We want to emphasize, that the NIZK proofs $\{\pi_i\}$ ensure that each R_i is well-formed with respect to $(\mathsf{pk}_i, \mathbf{d}, \Lambda_i)$, preventing malformed nonce injections; they are not needed for algebraic correctness but for soundness.