Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security

Renas Bacho[1,2], Sourav Das[3], Julian Loss[1] and Ling Ren[3]

[1]CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2]Saarland University, Saarbrücken, Germany
[3]University of Illinois at Urbana Champaign
{renas.bacho, loss}@cispa.de, {souravd2, renling}@illinois.edu

Abstract. Threshold signatures are one of the most important cryptographic primitives in distributed systems. The threshold Schnorr signature scheme, an efficient and pairing-free scheme, is a popular choice and is included in NIST's standards and recent call for threshold cryptography. Despite its importance, most threshold Schnorr signature schemes assume a static adversary in their security proof. A recent scheme proposed by Katsumata et al. (Crypto 2024) addresses this issue. However, it requires linear-sized signing keys and lacks the identifiable abort property, which makes it vulnerable to denial-of-service attacks. Other schemes with adaptive security either have reduced corruption thresholds or rely on non-standard assumptions such as the algebraic group model (AGM) or hardness of the algebraic one-more discrete logarithm (AOMDL) problem.

In this work, we present Glacius, the first threshold Schnorr signature scheme that overcomes all these issues. Glacius is adaptively secure based on the hardness of decisional Diffie-Hellman (DDH) in the random oracle model (ROM), and it supports a full corruption threshold $t < n$, where n is the total number of signers and t is the signing threshold. Additionally, Glacius provides constant-sized signing keys and identifiable abort, meaning signers can detect misbehavior. We also give a formal game-based definition of identifiable abort, addressing certain subtle issues present in existing definitions, which may be of independent interest.

1 Introduction

Threshold signatures [Des88, DF89] are an interactive type of digital signature scheme where the signing key is shared among a group of n signers, any $t + 1$ of which can jointly issue signatures. In this manner, signatures remain unforgeable with respect to an adversary that can corrupt at most $t < n$ of these signers. The increasing demand for decentralized applications has resulted in large-scale adoptions of threshold signatures [dra23, ic23]. Many state-of-the-art BFT protocols also use threshold signatures to lower communication and computation costs [MXC$^+$16, YMR$^+$19].

A popular choice for a threshold signature are threshold Schnorr signatures, because they do not rely on pairings, and their signature verification is more than $10\times$ faster than pairing-based schemes [ZWHZ19]. The Schnorr signature scheme has been standardized by NIST as the EdDSA signature, and the threshold Schnorr signature is sought by NIST's recent call for threshold cryptography [BP23]. Moreover, popular cryptocurrencies such as Bitcoin [Nak08] also support Schnorr signatures.

Static and adaptive security. Despite its popularity, until very recently, all efficient schemes for threshold Schnorr signatures have been proven secure only against a static adversary. A static adversary must declare the set of signers it will corrupt at the beginning of the protocol before the public key is set up. In contrast, an adaptive adversary can decide which signers to corrupt at any time during the execution of the protocol. Clearly, an adaptive adversary is a safer and more realistic assumption for the decentralized setting.

A large body of work has focused on designing threshold schemes that remain secure against such a powerful attacker [CGJ$^+$99, KRT24, CKM23, BLSW24]. Despite significant progress in this direction, existing adaptively secure threshold Schnorr variants come with an array of limitations or aggressive modeling assumptions that make them difficult to use in practical applications. These include the ability of signers to erase their internal states [CGJ$^+$99, BLSW24], reliance on strong number theoretic hardness assumptions

Table 1: Comparison of adaptively secure threshold Schnorr signature schemes. We do not consider schemes that assume a broadcast channel. We compare: number of signing rounds, supported corruption threshold, size of signing keys given as number of field elements, support for identifiable abort, reliance on the AGM, security loss of the reduction (for an adversary with advantage ϵ against the scheme, making at most q random oracle and signing queries) and computational assumption. All schemes assume the random oracle model (ROM).

Scheme	Rounds	Corruption threshold	Signing key size (in $\#\mathbb{Z}_p$)	Identifiable abort?	No AGM?	Security loss	Computational assumption
ZeroS [Mak22][‡]	3	t	$n+1$	✓	✓	$\Theta(q/\epsilon)$	DL
Sparkle [CKM23]	3	$t/2$	1	✓	✓	$\Theta(q/\epsilon)$	AOMDL
Sparkle [CKM23]	3	t	1	✓	✗	$\Theta(q/\epsilon)$	AOMDL
KRT [KRT24]	5	t	$n+2$	✗	✓	$\Theta(q^3/\epsilon)$	DL
Glacius (ours)	5	t	3	✓	✓	$\Theta(q/\epsilon)$	DDH

[‡] The scheme assumes private channels and relies on secure erasures.

such as the (algebraic) one-more discrete logarithm assumption (AOMDL) and the algebraic group model (AGM) [CKM23, BLSW24], or sub-optimal corruptions thresholds [CKM23]. Most recently, Katsumata et al. [KRT24] presented a scheme that bypasses all of the above problems. However, their approach requires keys that grow linearly in the number of signers and inherently does not allow the detection of a cheating party in case of an abort of their signing protocol. This property, also known as *identifiable abort* (IA) [KG21], has proven instrumental in the design of robust threshold Schnorr protocols such as ROAST [RRJ$^+$22].

Our contribution. Motivated by the above discussion, we present Glacius, a new threshold Schnorr signature scheme that overcomes all of these limitations. Concretely, Glacius has the following properties:

– Glacius supports $t < n$ adaptive corruptions under a well-studied and non-interactive assumption, namely, the DDH assumption. This is in contrast to the scheme by Crites et al. [CKM23], which can only support a sub-optimal corruption threshold of $t/2$ and relies on the hardness of AOMDL.
– Glacius supports identifiable abort, which allows signers to detect misbehaving signers after a signing session fails. This is in contrast to the scheme by Katsumata et al. [KRT24], where even one malicious signer can make any signing session fail without ever being detected.
– Glacius has signing keys consisting of three field elements per party, compared to $n + 2$ field elements in Katsumata et al.'s protocol [KRT24].[*]

One of our major technical innovations over the works of both Crites et al. [CKM23], and Katsumata et al. [KRT24] is to be able to tolerate $t < n$ corruptions without increasing the sizes of signing keys when the reduction rewinds. Of independent interest, our work also identifies (and fixes) some subtle issues in the game-based definition of identifiable abort for threshold signatures put forth by Ruffing et al. [RRJ$^+$22]. Informally, we observe that their definition does not capture some contrived scenarios in which more than one signer is honest but obtains differing views in the signing protocol. We discuss this in detail in Appendix B.

1.1 Technical Overview

We begin by giving a brief recap of the Schnorr signature scheme [Sch90], which will be useful for the ensuing discussion. Let $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$, where $\mathbb{G}$ is a cyclic group of prime order p and $g \in \mathbb{G}$ is a generator. Let $\mathsf{H}_{\mathsf{sig}} : \mathbb{G}^2 \times \mathcal{M} \to \mathbb{Z}_p$ be a hash function modeled as a random oracle, where $\mathcal{M}$ is a message space. The signing key $\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$ is a random field element, and $\mathsf{pk} := g^{\mathsf{sk}} \in \mathbb{G}$ is the corresponding public verification

[*]We note that n of these keys are symmetric keys between pairs of parties.

key. A signature σ on a message m is then $(\hat{A}, z) \in \mathbb{G} \times \mathbb{Z}_p$. To validate a signature $\sigma = (\hat{A}, z)$ on a message m, a validator first computes $c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$ and then checks that $g^z = \hat{A} \cdot \mathsf{pk}^c$. The security of the Schnorr signature scheme relies on the hardness of discrete logarithm (DL) in the ROM.

Das-Ren threshold BLS construction. Our starting point is the recent work by Das and Ren [DR24]. In their paper, they design a new threshold BLS signature scheme and prove its adaptive security under the DDH and CDH assumptions in the ROM. More precisely, in [DR24], the public parameters consist of three uniformly random generators $(g, h, v) \in \mathbb{G}^3$. The signing key sk_i of each signer i is $\mathsf{sk}_i = (s(i), r(i), u(i)) \in \mathbb{Z}_p^3$, where $s(x)$, $r(x)$, and $u(x)$ are three uniformly random degree-t polynomials with the constraint that $r(0) = u(0) = 0$, and its verification key is $\mathsf{pk}_i = g^{s(i)} h^{r(i)} v^{u(i)}$. The public key of the scheme is then $\mathsf{pk} = g^{s(0)}$, which is identical to the standard (i.e., non-threshold) BLS signature scheme [BLS01].

To prove adaptive security, [DR24] employs the following strategy. First, the reduction algorithm $\mathcal{B}$ uses the additional public parameter h to embed one component of the given CDH instance. This crucial change lets $\mathcal{B}$ locally sample the secret key polynomials. Second, $\mathcal{B}$ uses rigged keys during its interaction during reduction. Specifically, $\mathcal{B}$ during its interaction with $\mathcal{A}$ samples the polynomial $r(x)$ and $u(x)$ with non-zero $r(0)$ and uniformly random $u(0)$. These two crucial changes, along with additional techniques such as correlated programming of random oracles, allow them to prove the adaptive security of their BLS threshold signature scheme from standard assumptions.

Our approach: from BLS to Schnorr. We now explain the challenges we run into when we adapt the ideas of Das and Ren [DR24] to the Schnorr threshold signature scheme and how we overcome them.

In $\mathsf{Glacius}$, the signing key of signer i is $\mathsf{sk}_i := (s(i), r(i), u(i)) \in \mathbb{Z}_p^3$, where $s(x), r(x), u(x) \leftarrow\$ \mathbb{Z}_p[x]$ are three uniformly random degree-t polynomials such that $r(0) = u(0) = 0$. The public key is then $\mathsf{pk} := g^{s(0)} h^{r(0)} v^{u(0)} = g^{s(0)}$, where $g, h, v \in \mathbb{G}$ are three uniformly random and independent generators of $\mathbb{G}$.

Let $\mathcal{A}_{\mathsf{dl}}$ be the reduction algorithm that, given a discrete logarithm tuple $(g, h = g^{\alpha_h})$, seeks to compute α_h. $\mathcal{A}_{\mathsf{dl}}$ interacts with the forger $\mathcal{A}$ as follows. First, $\mathcal{A}_{\mathsf{dl}}$ computes $v := g^{\alpha_v}$ for some known $\alpha_v \leftarrow\$ \mathbb{Z}_p$ and uses (g, h, v) as the public parameters. Next, $\mathcal{A}_{\mathsf{dl}}$ samples the signing key polynomials $s(x)$, $r(x)$, and $u(x)$, so that $r(0) \neq 0$ and $u(0) \leftarrow\$ \mathbb{Z}_p$. This ensures that the signing keys are rigged. Specifically, if $s(0) = s$, $r(0) = 1$, and $u(0) = u$, the rigged public key is $\mathsf{pk} = g^{s(0)} h^{r(0)} v^{u(0)} = g^s h v^u$. During this interaction, whenever $\mathcal{A}$ corrupts a new signer, $\mathcal{A}_{\mathsf{dl}}$ reveals the corresponding signing keys using its knowledge of $s(x)$, $r(x)$, and $u(x)$. Finally, when $\mathcal{A}$ produces a forgery σ with respect to the rigged public key, $\mathcal{A}_{\mathsf{dl}}$ uses (s, u, α_v) and σ to get the discrete logarithm α_h.

The changes we describe so far are directly from [DR24]. However, these changes are not immediately compatible with the Schnorr signature scheme.

Challenge I: verification under rigged keys. The main issue is that honestly generated signatures no longer verify against the rigged public key. Specifically, let $\alpha := \alpha_h + \alpha_v u \in \mathbb{Z}_p$, so that the rigged public key is $\mathsf{pk} = g^s h v^u = g^{s+\alpha}$. Then, the signature $\sigma = (\hat{A} = g^a, z = a + c \cdot s)$ does not satisfy the Schnorr verification equation with respect to pk, because

$$g^z = g^{a+s\cdot c} \neq g^a \cdot (g^{s+\alpha})^c = \hat{A} \cdot \mathsf{pk}^c, \quad \text{where } c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m).$$

One possible solution is to modify the signature structure or verification algorithm to make it compatible with the rigged key, but this would break the compatibility with the (non-threshold) Schnorr verification. Instead, we resolve this by modifying how signers compute the nonce $\hat{A}$. Specifically, in our scheme, signers compute $\hat{A} := g^a \cdot \mathsf{H}_0(x)^{r(0)} \cdot \mathsf{H}_1(x)^{u(0)}$, where H_0 and H_1 are random oracles mapping to $\mathbb{G}$, and x is a common input we specify later. Note that in the real protocol, since $r(0) = u(0) = 0$, this modification does not affect the final signature or its verification.

We now briefly describe how this modification resolves the abovementioned issue. Note that when the keys are rigged (i.e., $r(0) = 1$ and $u(0) = u$), we have that $\hat{A} = g^a \cdot \mathsf{H}_0(x) \cdot \mathsf{H}_1(x)^u = g^a \cdot g^\gamma \cdot g^{u\beta}$ where $\mathsf{H}_0(x) = g^\gamma$ and $\mathsf{H}_1(x) = g^\beta$ for some $\beta, \gamma \in \mathbb{Z}_p$. Now, during simulation, if $\mathcal{A}_{\mathsf{dl}}$ chooses c such that

$$\gamma + \beta \cdot u + c \cdot \alpha = 0, \tag{1}$$

and programs the hash function $\mathsf{H}_{\mathsf{sig}}$ on input $\hat{A}$ to output c, i.e., $\mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m) := c$, then we get

$$\hat{A} \cdot \mathsf{pk}^c = g^a \cdot \mathsf{H}_0(x) \cdot \mathsf{H}_1(x)^u \cdot (g^{s+\alpha})^c = g^a \cdot g^\gamma \cdot g^{\beta u} \cdot g^{(s+\alpha)c} = g^a \cdot g^{sc} = g^z. \tag{2}$$

Therefore, by appropriately sampling (β, γ, c) and correlating the programming of the random oracles H_0, H_1, and $\mathsf{H}_{\mathsf{sig}}$, $\mathcal{A}_{\mathsf{dl}}$ can ensure that the final signature satisfies the Schnorr signature verification equation, even with the rigged public key.

For the security proof to go through, it is critical that $\mathcal{A}$ must not detect these changes. We will prove this is indeed the case, assuming the hardness of DDH in $\mathbb{G}$. More precisely, in each signing session, we sample $c, \beta \leftarrow\$ \mathbb{Z}_p$, and then program the random oracles as

$$\mathsf{H}_0(x) := g^{-(\beta u + c\alpha)}, \quad \mathsf{H}_1(x) := g^\beta,$$

where x is a carefully computed input, which we elaborate on in the next paragraph. We prove that such correlated programming is indistinguishable from uniform random programming of the random oracles. Compared to the work of Das and Ren, whose proof contains a similar step, our protocol has a significantly larger number of random variables and induces a more involved probability distribution. To tame the complexity of this step in our proof, we rely on Patarin's H-coefficient method [Pat08].

Challenge II: choosing the input. Now we specify how the important input x to the random oracles H_0 and H_1 is chosen. Note that for successful simulation of the signature scheme, equation (1) must hold for every signing session. Now, since $\hat{A}$ and hence $c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$ change with each signing session, the tuple (β, γ) must also change. This is because, for any fixed (β, γ), only one value of c can satisfy equation (1). Now, since (β, γ) depends on x, x must change in every signing session as well. Using a different x in every session allows $\mathcal{A}_{\mathsf{dl}}$ to sample a new (β, γ, c) tuple to satisfy equation (1) for that session.

The challenge, however, is to ensure that x changes every session without compromising security. One option is to use a monotonic session identifier as x, but this is undesirable as it requires signers to maintain a consistent state across sessions (see [NRSW20] for a detailed discussion of why this can be problematic). Additionally, using the message to be signed or the set of signers as x is also not viable, as an adversary could reuse these values across different sessions, preventing $\mathcal{A}_{\mathsf{dl}}$ from generating a new (β, γ, c) tuple.

In $\mathsf{Glacius}$, we ensure a unique x in each signing session by requiring every signer i to send a fresh random value $\rho_i \leftarrow\$ \mathbb{Z}_p$ to the other signers as the first message of the protocol. The value of x is then chosen as $\vec{\boldsymbol{\rho}} := [\rho_i]_{i \in \mathsf{SS}}$, where SS is the set of signers participating in that signing session. Note that $\vec{\boldsymbol{\rho}}$ is unique with high probability as each session consists of at least one honest signer.

Final challenge: programming the random oracles. Although the approach we describe above ensures a unique x for each signing session, it introduces a subtle yet critical issue. In any given signing session, let (β, c) denote the tuple such that $\mathsf{H}_0(x) = g^{-(\beta u + c\alpha)}$ and $\mathsf{H}_1(x) = g^\beta$. According to equation (2), for the simulation to succeed, we need to program $\mathsf{H}_{\mathsf{sig}}(\hat{A}) := c$ for the combined nonce $\hat{A}$. The problem is that the combined nonce $\hat{A}$ depends on the second-round messages that $\mathcal{A}$ sends to the honest signers. This allows an $\mathcal{A}$ to simultaneously pick multiple combined nonces by sending different second-round messages to different honest signers. In such cases, $\mathcal{A}_{\mathsf{dl}}$ would need to program $\mathsf{H}_{\mathsf{sig}}$ to return the same c for all these different nonces – an event highly improbable with an untampered random oracle. In $\mathsf{Glacius}$, we address this issue by requiring the signers to exchange the cryptographic hash of their view during the third round of the protocol to ensure that that protocol aborts in case $\mathcal{A}$ sends different messages to different honest signers.

1.2 Related Work

We discuss further related work, including other adaptively secure threshold signatures and multi-party signatures.

Threshold Schnorr signatures. Initial works [GJKR07, AF04, SS01, CGJ+99] mostly consider robust schemes for threshold Schnorr, where a signing session will always result in a valid signature output despite the presence of misbehaving signers. To achieve this, for each signing request, parties run a protocol similar to a distributed key generation (DKG) protocol. However, this assumes a broadcast channel and introduces inefficiency. The schemes [GJKR07, CGJ+99] achieve adaptive security but rely on secure erasures. The scheme of Abe-Fehr [AF04] achieves adaptive security without relying on secure erasures but needs to reduce the threshold to $t < n/2$.

Initiated by Komlo and Goldberg [KG21], many works have proposed efficient threshold Schnorr signatures in recent years [BCK+22, CKM21, CGRS23, RRJ+22, BTZ22, Lin22, Mak22, CKM23, KRT24]. Some of these schemes [Mak22, CKM23, KRT24] achieve adaptive security, but each comes with its own limitations. Table 1 gives a comparison between them and our scheme. Recently, there have also been many works that focus on robustness in asynchronous networks [RRJ+22, GS24, BHK+24, BLSW24]. Among these, HARTS [BLSW24] achieves adaptive security but assumes a broadcast channel and relies on the AGM and AOMDL.

Adaptively secure threshold signatures. The first adaptively secure threshold signatures were independently described by Canetti et al. [CGJ+99] and Frankel et al. [FMY99a, FMY99b]. Both works get adaptive security by introducing the "single-inconsistent player" (SIP) technique, and both rely on secure erasures. Jarecki-Lysyanskaya [JL00] extended the SIP technique to remove the need for secure erasures. Similar techniques were employed by Lysyanskaya-Peikert [LP01] and Abe-Fehr [AF04] to design adaptively secure threshold signatures without relying on erasures. Notably, the latter [AF04] is a threshold Schnorr signature. Later works [ADN06, WQL09] also apply the SIP technique to Rabin's threshold RSA signature [Rab98] and the Waters signature [Wat05].

Libert et al. [LJY14] presented a pairing-based, non-interactive, adaptively secure threshold signature scheme under DDH and the double-pairing assumption. Bacho and Loss in [BL22] proved adaptive security of Bolyreva's threshold BLS signature scheme [Bol03] under OMDL in the AGM. Very recently, Das and Ren [DR24] proposed an adaptively secure threshold BLS signature under DDH and CDH in asymmetric pairing groups. A recent pairing-free scheme, $\mathsf{Twinkle}$ [BLT+24], achieves full corruption threshold with adaptive security under DDH. Another recent pairing-based scheme [MMS+24] also achieves full corruption threshold with adaptive security under the bilinear DDH assumption in asymmetric groups.

Other related multi-party signatures. Lattice-based threshold signatures have recently gained increased attention [EKT24, dPKM+24, BGG+18, CATZ24, GKS24]. There is also an extensive amount of works on multi-signatures [BCJ08, BN06, PW23, TZ23], some of which focus on Schnorr multi-signatures [NRS21, NRSW20, MPSW19, KAB21]. Finally, there are also many other threshold signatures [CKP+23, GJKR96, GG20, CGG+20].

2 Preliminaries

Notation. Let λ denote the security parameter. We assume all algorithms get λ in unary as input. For a finite set S, we write $s \leftarrow\!\$\; S$ to denote that s is sampled uniformly at random from S, and we write $|S|$ to denote the size of S. For an integer $a \in \mathbb{N}$, we use $[a]$ to denote the ordered set $\{1, \ldots, a\}$. Further, we write "$\leftarrow$" for probabilistic assignment and ":=" for deterministic assignment. We use the terms *party* (resp. *parties*) and *signer* (resp. *signers*) interchangeably. We use bold fonts with arrows such as $\vec{\boldsymbol{\rho}}$ to denote vectors. For any $i \in [n]$ and $\mathsf{SS} \subseteq [n]$, we use $L_{i,\mathsf{SS}} := \prod_{k \in \mathsf{SS} \setminus \{i\}} k/(k-i)$ for the i-th Lagrange coefficient.

Threat model. We consider a set of n signers denoted by $\{1, 2, \ldots, n\}$. We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ who can corrupt up to $t < n$ signers adaptively. Corrupted signers can deviate arbitrarily from the protocol specification. We assume an asynchronous network with authenticated channels. We do not assume broadcast channels or secure erasures.

However, we make the following important note. For identifiable abort, we require a public key infrastructure (PKI). Concretely, each signer $i \in [n]$ has a public-secret key pair $(\mathsf{ek}_i, \mathsf{dk}_i)$ from a secure standard digital signature scheme $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ that is used to sign each signing protocol message before sending it to the other signers. For instance, we can instantiate DS with either EdDSA or standard Schnorr. But we emphasize that our protocol remains unforgeable even without PKI.

2.1 Secret Sharing and Computational Assumptions

Shamir secret sharing. The Shamir secret sharing [Sha79] embeds the secret $s \in \mathbb{Z}_p$ in the constant term of a polynomial $f(x) = s + a_1x + a_2x^2 + \cdots + a_dx^d$, where other coefficients $a_1, \ldots, a_d \leftarrow\$ \mathbb{Z}_p$ are chosen uniformly randomly. The i-th share of the secret s is then $f(i)$, i.e., the polynomial f evaluated at $x = i$. Given $d+1$ distinct shares, one can efficiently reconstruct the polynomial f and the secret s using Lagrange interpolation. Also, s is information theoretically hidden from an adversary that knows d or fewer shares.

Computational assumptions. Our protocol assumes the hardness of the discrete logarithm and decisional Diffie-Hellman problems. Let GGen be a group generation algorithm that, on input 1^λ, outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p, a generator $g \in \mathbb{G}$, and a description of the group operation. In our protocols, we assume that the discrete logarithm (DL) and decisional Diffie-Hellman (DDH) assumptions hold in the group $\mathbb{G}$, which we formally define in Appendix A.

2.2 Threshold Signatures

In this section, we introduce the syntax and security definitions for an R-round threshold signature scheme with identifiable abort (IA).

We define threshold signatures following [KRT24]. Let $t < n$ and R be natural numbers. Then, an R-round threshold signature scheme with IA is a tuple of PPT algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Detect})$ with the following specification. Intuitively, the Setup algorithm outputs public system parameters that all remaining algorithm takes as input. The KGen generates the signing and public keys of all signers. The Sig algorithm specifies the steps a signer should take in each round of the R round protocol to sign any given message, and the Ver algorithm specifies the final signature verification. Finally, signers use the Detect algorithm to identify misbehaving signers if a signing session fails, i.e., if any honest signer outputs $\perp$ instead of a valid signature. We provide formal descriptions of these algorithms next.

Definition 1 (Threshold Signatures with Identifiable Abort). *Let n be the total number of signers and $t < n$ be the threshold. Also, let $\mathsf{SS} \subseteq [n]$ be a set of signers with $|\mathsf{SS}| \geq t+1$. Each signer i maintains a state st_i to retain short-lived session-specific information. An R-round threshold signature scheme with identifiable abort TS for message space $\mathcal{M}$ is a tuple of PPT algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Detect})$ defined as follows:*

- $\mathsf{Setup}(1^\lambda, n, t) \to \mathsf{par}$: *The setup algorithm takes as input the security parameter 1^λ, the number n of total signers, and a threshold $t < n$, and outputs public parameters par. We assume that all other algorithms implicitly take par as input.*
- $\mathsf{KGen}(\mathsf{par}) \to (\mathsf{pk}, \{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in[n]})$: *The key generation algorithm takes as input the public parameters par and outputs a public key pk, an ordered set of threshold public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, and an ordered set of secret signing keys $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$. Each signer $j \in [n]$ receives the tuple $(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{sk}_j)$.*
- $\mathsf{Sig} = (\mathsf{Sig}_1, \ldots, \mathsf{Sig}_R, \mathsf{Comb})$: *The signing protocol is split into $R+1$ algorithms:*
  - $\mathsf{Sig}_k(\mathsf{SS}, m, i, (\mathsf{pm}_{k-1,j})_{j\in SS}, \mathsf{sk}_i, \mathsf{st}_i) \to (\mathsf{pm}_{k,i}, \mathsf{st}_i)$: *The k-th round signing algorithm for $k \in [R]$ takes as input a signer set SS, a message m, an index $i \in [n]$, a tuple of protocol messages of the $(k-1)$-th round $(\mathsf{pm}_{k-1,j})_{j\in SS}$, a secret signing key sk_i, and a state st_i. It outputs a protocol message $\mathsf{pm}_{k,i}$ for the k-th round and the updated state st_i. Here, we define $\mathsf{pm}_{0,j} := \perp$ for all $j \in \mathsf{SS}$.*
  - $\mathsf{Comb}(\mathsf{SS}, m, (\mathsf{pm}_{k,i})_{k\in[R], i\in\mathsf{SS}}) \to \sigma$: *The deterministic combine algorithm takes as input a signer set SS, a message m, and a tuple of protocol messages $(\mathsf{pm}_{k,i})_{k\in[R], i\in\mathsf{SS}}$, and outputs a signature σ.*
- $\mathsf{Ver}(\mathsf{pk}, m, \sigma) \to b$: *The deterministic verification algorithm takes as input a public key pk, a message m, and a signature σ, and outputs a bit $b \in \{0, 1\}$.*
- $\mathsf{Detect}(\mathsf{SS}, m, (\mathsf{trx}_j)_{j\in\mathsf{SS}}) \to \mathcal{J}$: *The detection algorithm takes as input a signer set SS, a message m, and a list $(\mathsf{trx}_j)_{j\in\mathsf{SS}}$ of transcripts, where each transcript $\mathsf{trx}_j := (\mathsf{pm}^j_{k,i})_{k\in[R], i\in\mathsf{SS}}$ is a tuple of protocol messages, and outputs a set $\mathcal{J} \subseteq [n]$ of signers.*

```
$\underline{\mathsf{Game}^{\mathsf{cor}}_{\mathsf{TS}}(1^\lambda, n, t, \mathsf{SS}, m)\text{:}}$
1: $\textbf{for } i \in \mathsf{SS} : \mathsf{st}_i := \varnothing$
2: $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
3: $(\mathsf{pk}, \{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{par})$
4: $\textbf{for } i \in \mathsf{SS} : \mathsf{pm}_{0,i} := \bot$
5: $\textbf{for } k \in [R] :$
6:     $\textbf{for } i \in \mathsf{SS} :$
7:         $(\mathsf{pm}_{k,i}, \mathsf{st}_i) \leftarrow \mathsf{Sig}_k(\mathsf{SS}, m, i, (\mathsf{pm}_{k-1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$
8: $\sigma := \mathsf{Comb}(\mathsf{SS}, m, (\mathsf{pm}_{k,i})_{k\in[R], i\in\mathsf{SS}})$
9: $\textbf{return } \mathsf{Ver}(\mathsf{pk}, m, \sigma)$
```

Fig. 1: The game $\mathsf{Game}^{\mathsf{cor}}_{\mathsf{TS}}$ for an R-round threshold signature scheme TS.

We require TS to satisfy the *correctness*, *unforgeability*, and the *identifiable abort* properties. Correctness ensures that the protocol behaves as expected when everyone is honest. Unforgeability ensures that the adversary cannot forge signatures, even after engaging in previous signing sessions and corrupting up to t signers adaptively. Finally, the identifiable abort property allows parties after a failed signing session to identify at least one misbehaving party that made the session fail.

Definition 2 (Correctness). *Consider the game $\mathsf{Game}^{\mathsf{cor}}_{\mathsf{TS}}$ defined in Figure 1. Then, an R-round threshold signature scheme TS is correct, if for all $\lambda \in \mathbb{N}$, $n, t \in \mathsf{poly}(\lambda)$ with $t < n$, messages $m \in \mathcal{M}$, and $\mathsf{SS} \subseteq [n]$ with $|\mathsf{SS}| \geq t + 1$, the following holds:*

$$\Pr\left[\mathsf{Game}^{\mathsf{cor}}_{\mathsf{TS}}(1^\lambda, n, t, \mathsf{SS}, m) \Rightarrow 1\right] \geq 1 - \mathsf{negl}(\lambda).$$

Unforgeability. Our unforgeability requirement is standard, which we formalize using the game $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ defined in Figure 2. Let $\mathcal{A}$ be the adversary in this game. Initially, $\mathcal{A}$ gets the public parameters par, an honestly generated public key pk, and threshold public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ of all signers as input. At any point in time, $\mathcal{A}$ can start a new signing session with identifier sid for signer set SS and message m by calling the oracle $\textsc{Next}(\mathsf{sid}, \mathsf{SS}, m)$. As such, we allow $\mathcal{A}$ to start and participate in any number of concurrent signing sessions. Additionally, $\mathcal{A}$ can corrupt up to t signers throughout the protocol using the oracle $\textsc{Corr}$. Upon corrupting signer $i \in [n]$, $\mathcal{A}$ learns its secret signing key sk_i and internal state st_i across all signing sessions. Further, $\mathcal{A}$ can interact with honest signers using the signing oracles $\textsc{Sig}_k$ for $k \in [R]$. $\mathcal{A}$ can query each of these oracles for an individual honest signer i and a session identifier sid. When querying $\textsc{Sig}_k$, $\mathcal{A}$ can freely choose the protocol messages pm_{k-1} of the $(k-1)$-th round. Importantly, we do not assume broadcast channels for the signing protocol, and the adversary could send different messages to different honest signers. However, we do assume authenticated channels, and our unforgeability game captures this via the $\mathsf{Allowed}$ algorithm. The $\mathsf{Allowed}$ algorithm also enforces that $\mathcal{A}$'s queries for each session are consistent. Finally, when $\mathcal{A}$ outputs a forgery (m^*, σ^*), we say that $\mathcal{A}$ wins if it has not initiated a signing session for the message m^*.

Our definition is inspired by [BLT+24] and adapted to the setting with authenticated channels. Essentially, it models an interactive version of the TS-UF-0 [BTZ22, BCK+22] unforgeability notion. However, we note that we work with TS-UF-0 for simplicity and believe that a similar analysis will also hold for the stronger notion of TS-UF-i for $i > 0$ [BCK+22].

Remark. Our threshold signature scheme has the special property that the internal state used in all completed signing sessions can be efficiently computed given only the secret signing key sk_i and the public protocol messages sent by parties. This allows us in the security proof to focus on revealing the internal states of incomplete signing sessions along with the secret signing key upon corruption. Importantly, our scheme does not rely on secure erasures for its security proof.

Definition 3 (Unforgeability Under Chosen-Message Attacks). *Let TS be an R-round threshold signature scheme, and consider the game $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ defined in Figure 2. Then, we say that TS is $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ secure, if for all $\lambda \in \mathbb{N}$, $n, t \in \mathsf{poly}(\lambda)$ with $t < n$, PPT adversaries $\mathcal{A}$, the following advantage is negligible:*

$$\varepsilon_\sigma := \mathsf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(1^\lambda, n, t) := \Pr\left[\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}(1^\lambda, n, t) \Rightarrow 1\right].$$

```
Game $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}(1^\lambda, n, t)$:
1: $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
2: $(\mathsf{pk}, \{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{par})$
3: $\mathcal{C} := \varnothing,\ \mathcal{H} := [n]$
4: $\mathsf{Queried} := \varnothing,\ \mathsf{pmsg} := \varnothing$
5: $\textsc{Sig} := (\textsc{Next}, (\textsc{Sig}_k)_{k\in[R]})$
6: $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\textsc{Corr},\textsc{Sig}}(\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$
7: if $m^* \in \mathsf{Queried}$ : return $0$
8: return $\mathsf{Ver}(\mathsf{pk}, m^*, \sigma^*)$

// Oracle to corrupt signers
Oracle $\textsc{Corr}(i)$:
9: if $(|\mathcal{C}| \geq t) \vee (i \in \mathcal{C})$ : return $\bot$
10: $\mathcal{C} := \mathcal{C} \cup \{i\},\ \mathcal{H} := \mathcal{H} \setminus \{i\}$
11: return $(\mathsf{sk}_i, \mathsf{st}_i)$

// Oracle to start a new signing session
Oracle $\textsc{Next}(\mathsf{sid}, \mathsf{SS}, m)$:
12: if $(|\mathsf{SS}| < t+1) \vee (\mathsf{SS} \not\subseteq [n])$ :
13:     return $\bot$
14: if $\mathsf{sid} \in \mathsf{Sessions}$ : return $\bot$
15: $\mathsf{Sessions} := \mathsf{Sessions} \cup \{\mathsf{sid}\}$
16: $\mathsf{Queried} := \mathsf{Queried} \cup \{m\}$
17: $\mathsf{message}[\mathsf{sid}] := m$
18: $\mathsf{signers}[\mathsf{sid}] := \mathsf{SS}$
19: for $i \in \mathsf{SS}$ : $\mathsf{round}[\mathsf{sid}, i] := 1$

// Check if the input to $\textsc{Sig}_k$ is valid
$\mathsf{Allowed}(\mathsf{sid}, k, \mathsf{SS}, m, i, (\mathsf{pm}_{k-1,j})_{j\in\mathsf{SS}})$:
  // assert returns 0 if the check fails
20: assert $\mathsf{sid} \in \mathsf{Sessions}$
21: assert $\mathsf{SS} = \mathsf{signers}[\mathsf{sid}]$
22: assert $i \in (\mathsf{SS} \cap \mathcal{H})$
23: assert $k = \mathsf{round}[\mathsf{sid}, i]$
24: assert $m = \mathsf{message}[\mathsf{sid}]$
25: if $k = 0$ : return $1$
26: for $j \in (\mathsf{SS} \cap \mathcal{H})$ :
27:     if $\mathsf{pm}_{k-1,j} \neq \mathsf{pmsg}[\mathsf{sid}, k-1, j]$ :
28:         return $0$
29: return $1$
```

```
// Oracle for the k-th signing round
Oracle $\textsc{Sig}_k(\mathsf{sid}, \mathsf{SS}, m, i, (\mathsf{pm}_{k-1,j})_{j\in\mathsf{SS}})$:
30: $\mathsf{input} := (\mathsf{SS}, m, i, (\mathsf{pm}_{k-1,j})_{j\in\mathsf{SS}})$
31: if $\mathsf{Allowed}(\mathsf{sid}, k, \mathsf{input}) = 0$ :
32:     return $\bot$
33: $(\mathsf{pm}_{k,i}, \mathsf{st}_i) \leftarrow \mathsf{Sig}_k(\mathsf{input}, \mathsf{sk}_i, \mathsf{st}_i)$
34: $\mathsf{pmsg}[\mathsf{sid}, k, i] := \mathsf{pm}_{k,i}$
35: $\mathsf{round}[\mathsf{sid}, i] := k+1$
36: $\mathsf{view}[\mathsf{sid}, k, i] := (\mathsf{pm}_{k-1,j})_{j\in\mathsf{SS}}$
37: if $((\mathsf{sid}, i) \notin \mathsf{failed}) \wedge (\mathsf{pm}_{k,i} = \bot)$ :
38:     $\mathsf{failed} := \mathsf{failed} \cup \{(\mathsf{sid}, i)\}$
39: return $\mathsf{pm}_{k,i}$

Game $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}(1^\lambda, n, t)$:
40: $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$
41: $(\mathsf{pk}, \{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KGen}(\mathsf{par})$
42: $\mathcal{C} := \varnothing,\ \mathcal{H} := [n]$
43: $\mathsf{pmsg} := \varnothing,\ \mathsf{failed} := \varnothing$
44: $\textsc{Sig} := (\textsc{Next}, (\textsc{Sig}_k)_{k\in[R]}, \textsc{Comb})$
45: $\mathsf{pk}' := (\mathsf{pk}, \{\mathsf{pk}_i\}_{i\in[n]})$
46: $(\mathsf{sid}^*, i^*, (\mathsf{trx}_j)_{j\in\overline{\mathsf{SS}}}) \leftarrow \mathcal{A}^{\textsc{Corr},\textsc{Sig}}(\mathsf{pk}')$
47: $\mathsf{SS}^* := \mathsf{Sessions}[\mathsf{sid}^*]$
48: $m^* := \mathsf{message}[\mathsf{sid}^*]$
49: assert $\overline{\mathsf{SS}} \neq (\mathsf{SS}^* \cap \mathcal{C})$
50: assert $(i^* \in \mathcal{H}) \wedge ((\mathsf{sid}^*, i^*) \in \mathsf{failed})$
51: for $i \in (\mathsf{SS}^* \cap \mathcal{H})$ :
52:     $\mathsf{trx}_i := (\mathsf{view}[\mathsf{sid}^*, k+1, i])_{k\in[R]}$
53: $\mathcal{J} \leftarrow \mathsf{Detect}(\mathsf{SS}^*, m^*, (\mathsf{trx}_j)_{j\in\mathsf{SS}^*})$
54: if $(\mathcal{J} \cap \mathcal{H} \neq \varnothing) \vee (\mathcal{J} \cap \mathsf{SS} = \varnothing)$ :
55:     return $1$
56: return $0$

Oracle $\textsc{Comb}(\mathsf{sid}, i, (\mathsf{pm}_{R,j})_{j\in\mathsf{SS}})$:
57: if $\mathsf{sid} \notin \mathsf{Sessions}$ : return $\bot$
58: $\mathsf{SS} := \mathsf{Sessions}[\mathsf{sid}], m := \mathsf{message}[\mathsf{sid}]$
59: $\mathsf{input} := (\mathsf{SS}, m, i, (\mathsf{pm}_{R,j})_{j\in\mathsf{SS}})$
60: if $\mathsf{Allowed}(\mathsf{sid}, R+1, \mathsf{input}) = 0$ :
61:     return $\bot$
62: $\mathsf{view}[\mathsf{sid}, R+1, i] := (\mathsf{pm}_{R,j})_{j\in\mathsf{SS}}$
63: $T_i := (\mathsf{view}[\mathsf{sid}, k+1, i])_{k\in[R]}$
64: $\sigma := \mathsf{Comb}(\mathsf{SS}, m, T_i)$
65: if $(\mathsf{sid}, i) \notin \mathsf{failed} \wedge \mathsf{Ver}(\mathsf{pk}, \sigma, m) = 0$ :
66:     $\mathsf{failed} := \mathsf{failed} \cup \{(\mathsf{sid}, i)\}$
```

Fig. 2: The games $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ and $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ for an R-round threshold signature scheme TS. We highlight the part needed for $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ in pink.

Identifiable abort. The Identifiable Abort (IA) property allows the identification of misbehaving signers when a signing session fails for an honest signer (i.e., the output is $\bot$). Our IA property builds on the definition in [RRJ+22], but we introduce key modifications to support (partially) interactive threshold signatures. We provide a detailed discussion on the reasons for our modifications in Appendix B. We formalize our IA property using the game $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ defined in Figure 2 and describe it next.

Let $\mathcal{A}$ be the adversary in this game. During the game, $\mathcal{A}$ outputs a tuple $(\mathsf{sid}^*, i^*, (\mathsf{trx}_j)_{j \in \overline{\mathsf{SS}}})$ for a signing session in which at least one honest signer output $\perp$. We capture this by maintaining the set failed. Concretely, we add the tuple (sid, i) to failed if (i) any honest signer i received invalid messages that resulted in signer i having output $\perp$ or (ii) the combined signature does not verify at some honest signer. Assuming $\mathcal{A}$ outputs $(\mathsf{sid}^*, i^*, (\mathsf{trx}_j)_{j \in \overline{\mathsf{SS}}})$ such that $(i^*, \mathsf{sid}^*) \in \mathsf{failed}$ and $i^* \in \mathcal{H}$, we run the Detect algorithm on the views of all the signers, where we let $\mathcal{A}$ arbitrarily choose the views of all corrupt signers in that session $\overline{\mathsf{SS}} := \mathsf{SS} \cap \mathcal{C}$. The Detect algorithm outputs a subset of signers $\mathcal{J} \subseteq [n]$. Finally, we say that $\mathcal{A}$ wins the game if at least one of the following conditions is satisfied: (i) the Detect algorithm fails to identify any misbehaving signer, or (ii) the Detect algorithm identifies an honest signer as malicious. Otherwise, $\mathcal{A}$ loses the game.

Definition 4 (Identifiable Abort Under Chosen-Message Attacks). *Let TS be an R-round threshold signature scheme, and consider the game $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ defined in Figure 2. Then, we say that TS is $\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ secure, if for all $\lambda \in \mathbb{N}$, $n, t \in \mathsf{poly}(\lambda)$ with $t < n$, PPT adversaries $\mathcal{A}$, the following advantage is negligible:*

$$\varepsilon_{\mathsf{ia}} := \mathsf{Adv}^{\mathsf{IA\text{-}CMA}}_{\mathcal{A},\mathsf{TS}}(1^\lambda, n, t) := \Pr\left[\mathsf{IA\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}(1^\lambda, n, t) \Rightarrow 1\right].$$

Remark. We note that our IA property differs from the IA property in the secure multiparty computation (MPC) literature. In the MPC literature, the IA property must also detect parties who fail to send required protocol messages. In contrast, our IA property focuses solely on detecting explicit misbehavior, not crash failures. The IA in MPC literature is stronger, but achieving this property requires broadcast channels and, consequently, imposes constraints on failure bounds and network conditions. Contrary to this, our IA definition is agnostic to network conditions and fault-tolerance levels.

3 Our Design

In this section, we present our five-round threshold Schnorr signature scheme $\mathsf{Glacius}$, assuming a trusted key generation. For our construction, we use $\mathcal{M}$ to denote the message space, and we use $\mathbb{Z}_p[x]_{(t)}$ to denote the set of all polynomials in $\mathbb{Z}_p[x]$ of degree t. Further, we let GGen denote a group generation algorithm that on input 1^λ outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p, a generator $g \in \mathbb{G}$, and a description of the group operation. We formally define our scheme as pseudocode in Figure 3 and give a verbal description next.

Setup. The setup algorithm Setup runs $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$, and then samples two uniformly random generators $h, v \leftarrow\$ \mathbb{G}$. Further, it selects five random oracles $\mathsf{H}_{\mathsf{com}} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{R}$, $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_{\mathsf{sig}} : \mathbb{G}^2 \times \mathcal{M} \to \mathbb{Z}_p$, and $\mathsf{H}_{\mathsf{view}} : \{0,1\}^* \to \mathcal{Y}$. Here, $\mathcal{R}, \mathcal{Y} \subseteq \{0,1\}^*$ are two sets such that $|\mathcal{R}|, |\mathcal{Y}| \geq 2^\lambda$. The public parameters of the scheme are then $\mathsf{par} := (n, t, \mathbb{G}, g, h, v, p, \mathsf{H}_{\mathsf{com}}, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_{\mathsf{sig}}, \mathsf{H}_{\mathsf{view}})$, which are public and known to all signers.

As we discussed earlier, we assume that all the algorithms below implicitly take par as input.

Key generation. The KGen algorithm takes as input the public parameters and samples three uniformly random polynomials $s(x), r(x), u(x) \leftarrow\$ \mathbb{Z}_p[x]_{(t)}$ of degree t each such that $r(0) = u(0) = 0$. The secret signing key of signer i is then $\mathsf{sk}_i := (s(i), r(i), u(i))$, and its public key share is $\mathsf{pk}_i := g^{s(i)} h^{r(i)} v^{u(i)}$. Further, the public key of the system is $\mathsf{pk} := g^{s(0)} h^{r(0)} v^{u(0)} = g^{s(0)}$.

Signing protocol. We assume an external mechanism that specifies the signer set $\mathsf{SS} \subseteq [n]$ with $|\mathsf{SS}| \geq t+1$ and a message m to be signed. In particular, we assume that all signers know and agree on (SS, m). For the identifiable abort property, we further require signers $i \in [n]$ to sign each protocol message before sending it to the other signers, using any secure digital signature scheme $\mathsf{DS} = (\mathsf{Key}, \mathsf{Sign}, \mathsf{Verify})$ such as EdDSA. An honest signer accepts protocol messages from other signers only if they are accompanied by a valid signature. We reiterate that we do not need this for unforgeability but only for identifiable abort. Our scheme still remains unforgeable even without an underlying signature scheme or a public-key infrastructure. For the sake of clarity, we omit these signatures in our subsequent description.

The signers in SS run the following protocol to compute a signature on m.

```
$\underline{\mathsf{Setup}(1^\lambda, n, t):}$
$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda),\ h, v \leftarrow\$\ \mathbb{G}$
$\mathcal{R}, \mathcal{Y} \subseteq \{0,1\}^*$ s.t. $|\mathcal{R}|, |\mathcal{Y}| \geq 2^\lambda$
    // Select hash functions
$\mathsf{H}_{\mathsf{com}} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{R}$
$\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$
$\mathsf{H}_{\mathsf{sig}} : \mathbb{G}^2 \times \mathcal{M} \to \mathbb{Z}_p$
$\mathsf{H}_{\mathsf{view}} : \{0,1\}^* \to \mathcal{Y}$
$\mathsf{H}_{\mathsf{all}} := (\mathsf{H}_{\mathsf{com}}, \mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_{\mathsf{sig}}, \mathsf{H}_{\mathsf{view}})$
**return** $\mathsf{par} := (n, t, \mathbb{G}, g, h, v, p, \mathsf{H}_{\mathsf{all}})$

// All algorithms take par as input
$\underline{\mathsf{KGen}(\mathsf{par}):}$
$s(x), r(x), u(x) \leftarrow\$\ \mathbb{Z}_p[x]_{(t)}$
s.t. $r(0) = u(0) = 0$
**for** $i \in [n]$ :
    $\mathsf{sk}_i := (s(i), r(i), u(i))$
    $\mathsf{pk}_i := g^{s(i)} h^{r(i)} v^{u(i)}$
$\mathsf{pk} := g^{s(0)} h^{r(0)} v^{u(0)} = g^{s(0)}$
**return** $(\mathsf{pk}, \{\mathsf{pk}_i, \mathsf{sk}_i\}_{i \in [n]})$

// All signers are aware of SS and m
$\underline{\mathsf{Sig}_1(\mathsf{SS}, m, i, \mathsf{sk}_i):}$
$\rho_i \leftarrow\$\ \{0,1\}^\lambda$
$\mathsf{pm}_{1,i} := \rho_i$
$\mathsf{st}_i := (i, \rho_i)$
**return** $(\mathsf{pm}_{1,i}, \mathsf{st}_i)$

$\underline{\mathsf{Sig}_2(\mathsf{SS}, m, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i):}$
**parse** $(i, \rho_i) := \mathsf{st}_i$
**if** $\mathsf{pm}_{1,i} \neq \rho_i$ : **return** $\perp$
**parse** $(\rho_j)_{j \in \mathsf{SS}} := (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}$
$\vec{\boldsymbol{\rho}} := ((j, \rho_j))_{j \in \mathsf{SS}}$
$a_i \leftarrow\$\ \mathbb{Z}_p$
$A_i := (g^{a_i} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(i)})^{L_{i,\mathsf{SS}}}$
$\mu_i := \mathsf{H}_{\mathsf{com}}(i, A_i)$
$\mathsf{pm}_{2,i} := \mu_i,$
$\mathsf{st}_i := (\vec{\boldsymbol{\rho}}, a_i, A_i, \mu_i, \mathsf{st}_i)$
**return** $(\mathsf{pm}_{2,i}, \mathsf{st}_i)$

// Verification algorithm
$\underline{\mathsf{Ver}(\mathsf{pk}, m, \sigma = (\hat{A}, z)):}$
$c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$
**if** $g^z = \hat{A} \cdot \mathsf{pk}^c$ :
    **return** 1
**return** 0

$\underline{\mathsf{Sig}_3(\mathsf{SS}, m, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i):}$
**parse** $(\vec{\boldsymbol{\rho}}, a_i, A_i, \mu_i, i, \rho_i) := \mathsf{st}_i$
**if** $\mathsf{pm}_{2,i} \neq \mu_i$ : **return** $\perp$
**parse** $\vec{\boldsymbol{\mu}} := (\mu_j)_{j \in \mathsf{SS}} := (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}$
$y_i := \mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}})$
$\mathsf{pm}_{3,i} := y_i$
$\mathsf{st}_i := (\vec{\boldsymbol{\mu}}, y_i, \mathsf{st}_i)$
**return** $(\mathsf{pm}_{3,i}, \mathsf{st}_i)$

$\underline{\mathsf{Sig}_4(\mathsf{SS}, m, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i):}$
**parse** $(\vec{\boldsymbol{\mu}}, y_i, \vec{\boldsymbol{\rho}}, a_i, A_i, \mu_i, i, \rho_i) := \mathsf{st}_i$
**if** $\mathsf{pm}_{3,i} \neq y_i$ : **return** $\perp$
**parse** $(y_j)_{j \in \mathsf{SS}} := (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}}$
**if** $\exists j \in \mathsf{SS}$ s.t. $y_j \neq y_i$ :
    **return** $\perp$
$\mathsf{pm}_{4,i} := A_i,$
$\mathsf{st}_i := (\vec{\boldsymbol{\mu}}, \vec{\boldsymbol{\rho}}, a_i, A_i, \mu_i, i, \rho_i)$
**return** $(\mathsf{pm}_{4,i}, \mathsf{st}_i)$

$\underline{\mathsf{Sig}_5(\mathsf{SS}, m, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i):}$
**parse** $(\vec{\boldsymbol{\mu}}, \vec{\boldsymbol{\rho}}, a_i, A_i, \mu_i, i, \rho_i) := \mathsf{st}_i$
**if** $\mathsf{pm}_{4,i} \neq A_i$ : **return** $\perp$
**parse** $(A_j)_{j \in \mathsf{SS}} := (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}}$
**if** $\exists j \in \mathsf{SS}$ s.t. $\mu_j \neq \mathsf{H}_{\mathsf{com}}(j, A_j)$ :
    **return** $\perp$
$\hat{A} := \prod_{j \in \mathsf{SS}} A_j$
$c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$
$z_i := L_{i,\mathsf{SS}} \cdot (a_i + cs(i))$
    // See Figure 9 for SigProve
$\pi_i := \mathsf{SigProve}(\mathsf{pk}_i, A_i, \vec{\boldsymbol{\rho}}, c, z; a, \mathsf{sk}_i)$
$\mathsf{pm}_{5,i} := (z_i, \pi_i)$
**return** $(\mathsf{pm}_{5,i}, \mathsf{st}_i)$

// Combine algorithm
$\underline{\mathsf{Comb}(\mathsf{SS}, m, (\mathsf{pm}_{k,j})_{k \in [5], j \in \mathsf{SS}}):}$
**parse** $(A_j)_{j \in \mathsf{SS}} := (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}}$
**parse** $(z_j, \cdot)_{j \in \mathsf{SS}} := (\mathsf{pm}_{5,j})_{j \in \mathsf{SS}}$
$\hat{A} := \prod_{j \in \mathsf{SS}} A_j$
$z := \sum_{j \in \mathsf{SS}} z_j$
**return** $\sigma := (\hat{A}, z)$
```

Fig. 3: Our threshold Schnorr signature scheme Glacius. We describe the Detect algorithm in Figure 8, and describe the notations in §3.

1. *Randomness sampling* (Sig_1): Each signer i samples a uniformly random string $\rho_i \leftarrow\$\ \{0,1\}^\lambda$, and sends the random string ρ_i to all other signers.
2. *Commitment phase* (Sig_2): Upon receiving all random strings $\vec{\boldsymbol{\rho}} := ((j, \rho_j))_{j \in \mathsf{SS}}$ from signers, each signer i proceeds as follows. It samples a random $a_i \leftarrow\$\ \mathbb{Z}_p$ and computes the nonce

$$A_i := \left(g^{a_i} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(i)}\right)^{L_{i,\mathsf{SS}}},$$

where $L_{i,\mathsf{SS}}$ denotes the i-th Lagrange coefficient for the set SS. Signer i then sends its commitment $\mu_i := \mathsf{H}_{\mathsf{com}}(i, A_i)$ to all other signers.

3. *View exchange* (Sig_3): Upon receiving all commitments $\vec{\boldsymbol{\mu}} := (\mu_j)_{j \in \mathsf{SS}}$ from signers, each signer i proceeds as follows. It takes its current view of the protocol messages $(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}})$ and computes the hash $y_i := \mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}})$. Then, it sends the hash y_i to all other signers.
4. *Opening phase* (Sig_4): Upon receiving all (compressed) views $\vec{\boldsymbol{y}} := (y_j)_{j \in \mathsf{SS}}$ from signers, each signer i proceeds as follows. For all $j \in \mathsf{SS}$, it checks whether $y_j = y_i$ holds, i.e., checks whether its view matches with the views of all other honest signers. If one of these checks fails, the signer outputs $\perp$ and aborts. Otherwise, it sends the opening A_i to all other signers.
5. *Signing phase* (Sig_5): Upon receiving all openings $(A_j)_{j \in \mathsf{SS}}$ from signers, each signer i proceeds as follows. First, it retrieves the commitments $\vec{\boldsymbol{\mu}} := (\mu_j)_{j \in \mathsf{SS}}$ from the second round. Then, for all $j \in \mathsf{SS}$, it checks whether $\mu_j = \mathsf{H}_{\mathsf{com}}(j, A_j)$ holds. If either of these checks fails, signer i outputs $\perp$ and aborts. Otherwise, it computes the combined nonce $\hat{A}$, challenge c, and its signature share z_i, as follows:

$$\hat{A} := \prod_{j \in \mathsf{SS}} A_j, \quad c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m), \quad z_i := L_{i,\mathsf{SS}} \cdot (a_i + c \cdot s(i)).$$

   Additionally, signer i also computes a proof of correctness $\pi_i := \mathsf{SigProve}(\mathsf{pk}_i, A_i, c, z_i; a_i, \mathsf{sk}_i)$ for its signature share z_i as defined in Figure 9. Finally, it sends its signature share (z_i, π_i) to the other signers. We note that Glacius uses the correctness proof π_i only in the Detect protocol.

Combine algorithm. The combine algorithm gets all the protocol messages $(\rho_j, \mu_j, y_j, A_j, (z_j, \pi_j))_{j \in \mathsf{SS}}$ for a signing session and computes $\hat{A} := \prod_{j \in \mathsf{SS}} A_j$ and $z := \sum_{j \in \mathsf{SS}} z_j$. It outputs $\sigma := (\hat{A}, z)$ as the final signature.

Verification algorithm. The verifier gets a public key pk, a message $m \in \mathcal{M}$, and a signature $\sigma = (\hat{A}, z)$. Then, it computes $c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$ and accepts the signature (i.e., outputs $b = 1$) if and only if $g^z = \hat{A} \cdot \mathsf{pk}^c$.

Detection algorithm (Figure 8). The Detect algorithm takes as input the signer set SS, message m, and the protocol messages received by all signers in SS, where $\mathsf{trx}_j = (\mathsf{pm}^j_{k,i})_{k \in [R], i \in \mathsf{SS}}$ is the protocol message received by signer j. The algorithm classifies a signer i as misbehaving, i.e., adds i to $\mathcal{J}$, only if:

(1) Signer i provably equivocated during the signing session, i.e., sent different messages to different honest signers. To detect equivocation, the algorithm uses the digital signatures (from the underlying signature scheme DS) attached to the protocol messages (step 3-6 in Figure 8).
(2) Signer i sent an invalid signature share z_i, i.e., the proof π_i sent by i does not verify (step 7-17 in Figure 8). Note that we check the validity of z_i with respect to the local view of signer i.

4 Security Analysis

In this section, we give an adaptive security proof for Glacius. For our security analysis, we rely on the discrete logarithm (DL) assumption and the decisional Diffie-Hellman (DDH) assumption, which are defined in Appendix A.

4.1 Correctness

The correctness of our scheme is straightforward. For this, consider a signing session among (at least) $t+1$ signers $\mathsf{SS} \subseteq [n]$ with the message m to be signed. In the first round, each signer receives a common randomness vector $\vec{\boldsymbol{\rho}} = (\rho_i)_{i \in \mathsf{SS}}$. In the second round, each signer i computes its nonce $A_i := g^{a_i} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(i)}$, and sends a commitment μ_i to the other signers. Since all signers behave honestly, they will all obtain the same vectors $(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}})$ and thus have the same compressed hash view from the third round. In the fourth round, nonces $(A_i)_{i\in\mathsf{SS}}$ are revealed and each signer computes the combined nonce:

$$\hat{A} = \prod_{i\in\mathsf{SS}} A_i = g^{\sum_{i\in\mathsf{SS}} a_i \cdot L_{i,\mathsf{SS}}} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(0)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(0)} = g^{\sum_{i\in\mathsf{SS}} a_i \cdot L_{i,\mathsf{SS}}},$$

where we use $r(0) = u(0) = 0$. Next, the challenge is derived $c := \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$, and each signer i computes its signature share $z_i := L_{i,\mathsf{SS}} \cdot (a_i + c \cdot s(i))$. Thus, the combined signature is $z := \sum_{i\in\mathsf{SS}} z_i = c \cdot s + \sum_{i\in\mathsf{SS}} L_{i,\mathsf{SS}} \cdot a_i$, where $s := s(0)$. As a result, the Schnorr verification equation $g^z = \hat{A} \cdot \mathsf{pk}^c$ holds.

4.2 Helper Lemmas

Our unforgeability proof relies on the following two lemmas.

Lemma 1 ([NR04]). *For any $q \in \mathsf{poly}(\lambda)$, assuming hardness of the* DDH *assumption in the group $\mathbb{G}$, the following two distributions are indistinguishable:*

$$\mathcal{D}_0 := \left\{ (g, g^{\alpha}, (g^{\beta_i}, g^{\gamma_i})_{i\in[q]}) \;\middle|\; \alpha \leftarrow\!\$\; \mathbb{Z}_p,\; (\beta_i, \gamma_i) \leftarrow\!\$\; \mathbb{Z}_p^2 \;\forall i \in [q] \right\},$$
$$\mathcal{D}_1 := \left\{ (g, g^{\alpha}, (g^{\beta_i}, g^{\alpha\cdot\beta_i})_{i\in[q]}) \;\middle|\; \alpha \leftarrow\!\$\; \mathbb{Z}_p,\; \beta_i \leftarrow\!\$\; \mathbb{Z}_p \;\forall i \in [q] \right\}.$$

More precisely, if an adversary $\mathcal{A}$ can distinguish between a sample from $\mathcal{D}_0$ and $\mathcal{D}_1$ with probability ε, then we can break the DDH *assumption in the group $\mathbb{G}$ with probability at least $\varepsilon - 1/p$. This implies $\varepsilon \leq \varepsilon_{\mathsf{ddh}} + 1/p$ in time $\mathsf{poly}(q, \lambda)$ and running time of $\mathcal{A}$.*

Lemma 2 ([DR24], Lemma 4). *Let (X_0, Y_0) and (X_1, Y_1) be two tuples of discrete random variables, where X_0 is independent of Y_0 and X_1 is independent of Y_1. Then, for every function $f(X_\theta, Y_\theta)$ for either $\theta \in \{0, 1\}$, if $X_0 \equiv X_1$ and $Y_0 \equiv Y_1$, where $\equiv$ indicates that the two random variables are identically distributed, then $(X_0, Y_0, f(X_0, Y_0)) \equiv (X_1, Y_1, f(X_1, Y_1))$.*

4.3 Unforgeability Proof

We will show unforgeability assuming hardness of discrete logarithm (DL) and decisional Diffie-Hellman (DDH) in the group $\mathbb{G}$. Note that the hardness of DDH implies the hardness of DL. Nevertheless, we will keep this separation in our discussion for the modularity of the proof.

We achieve this via a sequence of games $\mathbf{G}_0$-$\mathbf{G}_{11}$, where $\mathbf{G}_0$ is the real protocol execution and $\mathbf{G}_{11}$ is the interaction of a PPT adversary $\mathcal{A}$ with a reduction algorithm $\mathcal{A}_{\mathsf{dl}}$. Here on, for any game $\mathbf{G}_i$, we will use "$\mathbf{G}_i \Rightarrow 1$" as a shorthand notation for the event that the adversary $\mathcal{A}$ forges a signature in $\mathbf{G}_i$.

Game $\mathbf{G}_0$: This game is the security game $\mathsf{UF\text{-}CMA}_{\mathsf{TS}}^{\mathcal{A}}$ for our threshold signature scheme, where the game follows the honest protocol. Here, the game provides $\mathcal{A}$ access to any random oracle using standard lazy sampling. Let $h := g^{\alpha_h}$ and $v := g^{\alpha_v}$ for some $\alpha_h, \alpha_v \in \mathbb{Z}_p^*$. Recall that $h, v \in \mathbb{G}$ are uniformly random generators (along with $g \in \mathbb{G}$) sampled by the setup algorithm Setup.

We also make some purely conceptual changes to the game. Assuming the game outputs 1, let $(m^*, \sigma^* = (\hat{A}, \hat{z}))$ denote the forgery output by $\mathcal{A}$. First, we assume that $\mathcal{A}$ always queries $\mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m^*)$ before outputting the forgery. Second, we assume that $\mathcal{A}$ makes exactly t (distinct) corruption queries. These changes are without loss of generality and do not change the advantage of $\mathcal{A}$. To see this, we can always build a wrapper adversary that internally runs $\mathcal{A}$, but makes a query $\mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m^*)$ and corrupts the required number of signers (i.e., in total t signers) before terminating. Clearly, we have

$$\mathsf{Adv}_{\mathcal{A},\mathsf{TS}}^{\mathsf{UF\text{-}CMA}}(\lambda) = \Pr[\mathbf{G}_0 \Rightarrow 1] = \varepsilon_\sigma.$$

Game $\mathbf{G}_1$: In this game, we rule out collisions for $\mathsf{H}_0, \mathsf{H}_1$, $\mathsf{H}_{\mathsf{com}}$, and $\mathsf{H}_{\mathsf{view}}$. Specifically, the game aborts if one of the following events occurs:

(i) There are $\vec{\boldsymbol{\rho}} \neq \vec{\boldsymbol{\rho}}'$ such that either $\mathsf{H}_0(\vec{\boldsymbol{\rho}}) = \mathsf{H}_0(\vec{\boldsymbol{\rho}}')$ or $\mathsf{H}_1(\vec{\boldsymbol{\rho}}) = \mathsf{H}_1(\vec{\boldsymbol{\rho}}')$. Recall that $\mathsf{H}_0, \mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$ are the random oracles we use in the second round to compute the nonces A_j.

(ii) There are $(j, A_j) \neq (j', A'_j)$ such that $\mathsf{H}_{\mathsf{com}}(j, A_j) = \mathsf{H}_{\mathsf{com}}(j', A'_j)$. Recall that $\mathsf{H}_{\mathsf{com}} : \{0,1\}^\lambda \times \mathbb{G} \to \mathcal{R}$ is the random oracle we use in the second round to commit to nonces A_j.

(iii) There are $(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}}) \neq (\vec{\boldsymbol{\rho}}', \vec{\boldsymbol{\mu}}')$ such that $\mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}, \vec{\boldsymbol{\mu}}) = \mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}', \vec{\boldsymbol{\mu}}')$. Recall that $\mathsf{H}_{\mathsf{view}} : \{0,1\}^* \to \mathcal{Y}$ is the random oracle used in the third round to exchange compressed views (of the first two rounds).

Using standard collision probability analysis, we find that (i) happens with probability at most q_{H}^2/p, (ii) happens with probability at most $q_{\mathsf{com}}^2/|\mathcal{R}|$, and (iii) happens with probability at most $q_{\mathsf{view}}^2/|\mathcal{Y}|$. Here, q_{H} is an upper bound on the number of total queries $\mathcal{A}$ can make to H_0 and H_1 combined. Similarly, q_{com} and q_{view} are an upper bound on the total number of queries $\mathcal{A}$ can make to $\mathsf{H}_{\mathsf{com}}$ and $\mathsf{H}_{\mathsf{view}}$, respectively. Thus, we get

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \frac{(q_{\mathsf{H}})^2}{p} + \frac{(q_{\mathsf{com}})^2}{|\mathcal{R}|} + \frac{(q_{\mathsf{view}})^2}{|\mathcal{Y}|}.$$

<u>GAME $\mathbf{G}_2$</u>: In this game, we rule out the event that upon receiving a first-round signing request from the adversary for honest signer i, we sample the same ρ_i twice. To do so, we maintain a list $\mathsf{PastRho}$ to keep track of previously sampled first-round messages sampled by honest signers. We populate $\mathsf{PastRho}$ upon each $\textsc{Sig}_1$ oracle query. More specifically, upon each $\textsc{Sig}_1$ on input of the form $(\cdot, i, \cdot)$, the game samples $\rho_i \leftarrow\$ \{0,1\}^\lambda$. Next, if $(i, \rho_i) \in \mathsf{PastRho}$, the game aborts. Otherwise, the updates $\mathsf{PastRho} := \mathsf{PastRho} \cup \{(i, \rho_i)\}$.

Looking ahead, this game, combined with our use of authenticated channels ensure that the first-round messages an honest signer receives in each signing session are unique. This is because the $\vec{\boldsymbol{\rho}}$ vector includes contributions from at least one honest signer. Since honest signers do not sample the same first-round message twice, the $\vec{\boldsymbol{\rho}}$ vector will be unique for every signing session.

We now bound the probability of aborting in this game. For each signing query, since ρ_i is uniformly random, the game aborts with probability at most $q_s/2^\lambda$ (where q_s is an upper bound on the number of signing queries $\mathcal{A}$ can make). As a result, by a union bound over all signing queries, we get

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \frac{(q_s)^2}{2^\lambda}.$$

<u>GAME $\mathbf{G}_3$</u>: In this game, we change the signing oracle, specifically the commitment and opening phases of the signing protocol (i.e., Sig_2 and Sig_4). Recall that until now, during the commitment phase (i.e., Sig_2), an honest signer $i \in \mathsf{SS}$ computes $A_i := g^{a_i} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(i)}$ for $a_i \leftarrow\$ \mathbb{Z}_p$ and then sends $\mu_i := \mathsf{H}_{\mathsf{com}}(i, A_i)$ to the other signers. Let $\vec{\boldsymbol{\mu}} := (\mu_j)_{j \in \mathsf{SS}}$ be the vector of commitments sent by signers. Note that $\mathbf{G}_1$ ensures (due to collision resistance property of $\mathsf{H}_{\mathsf{view}}$) that all honest signers proceed to Sig_4 step of a signing session only if all honest signers receive the same vectors $\vec{\boldsymbol{\rho}}$ and $\vec{\boldsymbol{\mu}}$ at the end of round 2 of that session. Next, during the opening phase (i.e., Sig_4), signer i reveals its nonce A_i.

In this game, we change this as follows. During Sig_2 with session identifier sid, the game samples a random commitment $\mu_i \leftarrow\$ \mathcal{R}$ and sends it on behalf of each honest signer i. The game also inserts an entry (sid, i, μ_i) into a list $\mathsf{Pending\text{-}H_{com}}$. If there is already an entry $(\cdot, \cdot, \mu_i) \in \mathsf{Pending\text{-}H_{com}}$, then the game aborts.

There are two situations where we need to reveal the preimage of the commitment μ_i to $\mathcal{A}$. Namely, (i) in the opening phase Sig_4, we need to reveal A_i, after signer i receives all Sig_3 messages for that respective session, and (ii) when $\mathcal{A}$ corrupts signer i, we further need to reveal a_i (which also gives A_i). To handle this, we consider two cases:

1. Signer i gets corrupted before or during the opening phase: In this case, we sample a random $a_i \leftarrow\$ \mathbb{Z}_p$, compute $A_i := \left(g^{a_i} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{r(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{u(i)}\right)^{L_{i,\mathsf{SS}}}$, where $\vec{\boldsymbol{\rho}}$ is the Sig_1 vector signer i obtains,[†] and then check if $\mathsf{H}_{\mathsf{com}}(i, A_i)$ is already defined. If so, we abort the game. Otherwise, we program $\mathsf{H}_{\mathsf{com}}(i, A_i) := \mu_i$ and remove the entry $(\cdot, \cdot, \mu_i)$ from $\mathsf{Pending\text{-}H_{com}}$. In particular, we can reveal A_i in the opening phase, and in case the corruption happens before that, we can reveal both a_i and A_i for that session.

[†]Note that if signer i gets corrupted before it receives the first-round vector $\vec{\boldsymbol{\rho}}$, then there is nothing to reveal for that particular session, as the ρ_i are public anyway.

2. Signer i reaches the opening phase or gets corrupted after the opening phase: In this case, we proceed as above and do the following in the opening phase: sample a random $a_i \leftarrow\$ \mathbb{Z}_p$, compute A_i honestly, program $\mathsf{H}_{\mathsf{com}}(i, A_i) := \mu_i$ (after a check as above), and reveal A_i. In particular, a_i is already defined if i gets corrupted after the opening phase, and we can reveal it upon corruption.

With this change, we emphasize the following: For honest signers i that reach the opening phase without being corrupted (the second case above), the element $a_i \in \mathbb{Z}_p$ is sampled only after signer i receives all the third-round Sig_3 messages.

Clearly, the view of $\mathcal{A}$ is only affected by these changes if (i) the game samples the same μ_i twice, or (ii) if A_i matches a previous $\mathsf{H}_{\mathsf{com}}$ query. Using standard collision probabilities, we find that (i) happens with probability at most $q_s^2/|\mathcal{R}|$. Further, since A_i is uniformly random in $\mathbb{Z}_p$, it will match a previous $\mathsf{H}_{\mathsf{com}}$ with probability at most q_{com}/p. Thus, using a union bound over all signing queries, we find that (ii) happens with probability at most $q_s \cdot q_{\mathsf{com}}/p$. We get

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \frac{(q_s)^2}{|\mathcal{R}|} + \frac{q_s \cdot q_{\mathsf{com}}}{p}.$$

<u>GAME $\mathbf{G}_4$</u>: In this game, we abort if $\mathcal{A}$ breaks the observability of the random oracle $\mathsf{H}_{\mathsf{com}}$. More precisely, the game aborts if for any signing session and $j \in \mathcal{C}$, $\mathcal{A}$ manages to output an element A_j in round 4 for its round 2 message μ_j such that $\mu_j = \mathsf{H}_{\mathsf{com}}(j, A_j)$, without having queried the random oracle $\mathsf{H}_{\mathsf{com}}$ on (j, A_j).

Note that for each $j \in \mathcal{C}$, unless $\mathsf{H}_{\mathsf{com}}(j, \cdot) \neq \perp$, the game outputs $\mathsf{H}_{\mathsf{com}}(j, \cdot)$ with uniformly random values in $\mathcal{R}$. Moreover, after $\mathcal{A}$ corrupts the signer j, the game populates $\mathsf{H}_{\mathsf{com}}$ for inputs of the form $(j, \cdot)$ only when $\mathcal{A}$ explicitly queries $\mathsf{H}_{\mathsf{com}}$ on the input $(j, \cdot)$. Therefore, the probability of $\mathsf{H}_{\mathsf{com}}(j, A_j) = \mu_j$ for each (j, A_j) is $1/|\mathcal{R}|$. Hence, by taking union bound over all signing sessions, we get:

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \frac{q_s}{|\mathcal{R}|}.$$

<u>GAME $\mathbf{G}_5$</u>: In this game, we introduce a map $\mathcal{X}$, initially empty, and maintain it as follows. For every random oracle query to H_b on input $\vec{\boldsymbol{\rho}}_\ell$ for either $b \in \{0, 1\}$, we do: if $\mathsf{H}_b(\vec{\boldsymbol{\rho}}_\ell) = \perp$, then we sample three random values $\beta_\ell, \gamma_\ell, c_\ell \leftarrow\$ \mathbb{Z}_p$ and program the random oracles as

$$\mathsf{H}_0(\vec{\boldsymbol{\rho}}_\ell) := g^{\gamma_\ell}, \quad \mathsf{H}_1(\vec{\boldsymbol{\rho}}_\ell) := g^{\beta_\ell}, \tag{3}$$

and we also update the map $\mathcal{X}$ as $\mathcal{X}[\vec{\boldsymbol{\rho}}_\ell] := c_\ell$. Otherwise, we output the already defined values $\mathsf{H}_b(\vec{\boldsymbol{\rho}}_\ell)$. Note that we always do simultaneous programming of both the random oracles H_0 and H_1 upon receiving a query $\mathsf{H}_b(\vec{\boldsymbol{\rho}}_\ell)$. We reiterate that the vectors $\vec{\boldsymbol{\rho}}_\ell$ generated in a signing session are always unique (guaranteed by game $\mathbf{G}_2$) since they have contributions from at least one honest signer.

Additionally, we change the game as follows: Consider a signing session with signer set SS and message m. The game waits until it receives the second-round messages for honest signers from $\mathcal{A}$. Then, it programs the random oracle $\mathsf{H}_{\mathsf{sig}}$ as follows, considering two cases:

1. If the game receives different Sig_1 and Sig_2 messages from $\mathcal{A}$ for different honest signers i, j in that session (i.e., $(\vec{\boldsymbol{\rho}}_{(i)}, \vec{\boldsymbol{\mu}}_{(i)}) \neq (\vec{\boldsymbol{\rho}}_{(j)}, \vec{\boldsymbol{\mu}}_{(j)})$), then the game continues as before. In this case, we know that honest signers will not reach the opening phase anyway because of different compressed view hashes $\mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}_{(i)}, \vec{\boldsymbol{\mu}}_{(i)}) \neq \mathsf{H}_{\mathsf{view}}(\vec{\boldsymbol{\rho}}_{(j)}, \vec{\boldsymbol{\mu}}_{(j)})$ in the third round (note that collisions for random oracle $\mathsf{H}_{\mathsf{view}}$ are ruled out by $\mathbf{G}_1$).
2. Otherwise, let $\vec{\boldsymbol{\rho}}$ and $\vec{\boldsymbol{\mu}}$ be the common vectors of first- and second-round messages received by honest signers, respectively, who did not abort after the third round. Then, the game proceeds as follows to the opening phase. Let $(\mu_j)_{j \in \mathsf{SS}} := \vec{\boldsymbol{\mu}}$ and $\mu_j = \mathsf{H}_{\mathsf{com}}(j, A_j)$ for $j \in (\mathcal{C} \cap \mathsf{SS})$. By the changes in game $\mathbf{G}_4$, we know that the preimage of μ_j from corrupt signers j exists and can be extracted via observability of $\mathsf{H}_{\mathsf{com}}$. As such, the game extracts (j, A_j) for $j \in (\mathcal{C} \cap \mathsf{SS})$, and then computes its own A_i for $i \in (\mathcal{C} \cap \mathsf{SS})$

as in game $\mathbf{G}_3$ (i.e., after it has received the third-round messages and did not abort). With that, the game computes the combined nonce $\hat{A}$ as

$$\hat{A} := \prod_{j \in (\mathcal{H} \cap \mathsf{SS})} A_j \cdot \prod_{j \in (\mathcal{C} \cap \mathsf{SS})} A_j = \prod_{j \in \mathsf{SS}} A_j, \tag{4}$$

and programs $\mathsf{H}_{\mathsf{sig}}(\hat{A}, m, \mathsf{pk}) := \mathcal{X}[\vec{\boldsymbol{\rho}}]$ in case $\mathsf{H}_{\mathsf{sig}}(\hat{A}, m, \mathsf{pk}) = \bot$. In case the random oracle $\mathsf{H}_{\mathsf{sig}}(\hat{A}, m, \mathsf{pk})$ is already defined, the game aborts.

We now bound the probability of aborting in this game. Recall from game $\mathbf{G}_2$ that the vectors $\vec{\boldsymbol{\rho}}_\ell$ generated in a signing session ℓ are always unique, and that the value $\mathcal{X}[\vec{\boldsymbol{\rho}}_\ell] = c_\ell$ is sampled uniformly at random and independent for each $\vec{\boldsymbol{\rho}}_\ell$. Additionally, since we know that there is at most one $\vec{\boldsymbol{\rho}}$ and $\vec{\boldsymbol{\mu}}$ vector for honest signers that did not abort after the third round (see the second case above), it implies that (i) we extract at most one $\hat{A}$, and (ii) we program $\mathsf{H}_{\mathsf{sig}}$ at most once with a uniformly random $\mathcal{X}[\vec{\boldsymbol{\rho}}]$.

With this observation, we can now easily bound the probability of aborting. As mentioned above, we sample the A_i for honest signers (due to game $\mathbf{G}_3$) after we have extracted the nonces A_j from corrupt signers $j \in (\mathcal{C} \cap \mathsf{SS})$ via observability of $\mathsf{H}_{\mathsf{com}}$. Since the nonces A_i for honest signers are sampled uniformly random, we know that the combined nonce $\hat{A}$ will also be uniformly random and hidden from $\mathcal{A}$ at the time when we program $\mathsf{H}_{\mathsf{sig}}$. Therefore, for a fixed signing session, the game aborts with probability at most q_{sig}/p (where q_{sig} is an upper bound on the number of queries to $\mathsf{H}_{\mathsf{sig}}$).

Thus, by a union bound over all signing queries, we get

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \frac{q_s \cdot q_{\mathsf{sig}}}{p}. \tag{5}$$

<u>Game $\mathbf{G}_6$</u>: 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 $\mathsf{H}_b(\vec{\boldsymbol{\rho}}_\ell)$ 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

$$\mathsf{H}_0(\vec{\boldsymbol{\rho}}_\ell) := g^{(-\gamma_\ell - \alpha_h \cdot \beta_\ell) \cdot \alpha_v^{-1} - \alpha \cdot c_\ell}, \quad \mathsf{H}_1(\vec{\boldsymbol{\rho}}_\ell) := g^{\beta_\ell}. \tag{6}$$

Recall that $h := g^{\alpha_h}$ and $v := g^{\alpha_v}$ by definition (see notation in $\mathbf{G}_0$). Again, we update the map $\mathcal{X}$ as $\mathcal{X}[\vec{\boldsymbol{\rho}}_\ell] := c_\ell$, and program $\mathsf{H}_{\mathsf{sig}}$ as in the previous game. Here, we reiterate that one of the following two cases happens: First, the game cannot extract a unique combined nonce because of conflicting views among honest signers, in which case none of the honest signers will reach the opening phase. Or second, the game can extract a unique combined nonce on which it programs $\mathsf{H}_{\mathsf{sig}}$ as $\mathcal{X}[\vec{\boldsymbol{\rho}}_\ell]$. 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 $(-\gamma_\ell - \alpha_h \cdot \beta_\ell) \cdot \alpha_v^{-1} - c_\ell \cdot \alpha$ is also uniformly random and independent of α, β_ℓ and c_ℓ. Therefore, $\mathcal{A}$'s view in game $\mathbf{G}_6$ is identically distributed as its view in game $\mathbf{G}_5$, and we get $\Pr[\mathbf{G}_5 \Rightarrow 1] = \Pr[\mathbf{G}_6 \Rightarrow 1]$.

<u>Game $\mathbf{G}_7$</u>: So far, we have programmed H_0 and H_1 with uniformly random and independent values in $\mathbb{G}$. In this game, we change this using correlated values. More specifically, for each query $\mathsf{H}_b(\vec{\boldsymbol{\rho}}_\ell)$ 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

$$\mathsf{H}_0(\vec{\boldsymbol{\rho}}_\ell) := g^{(-\alpha \cdot \beta_\ell - \alpha_h \cdot \beta_\ell) \cdot \alpha_v^{-1}} \cdot (g^{-\alpha})^{c_\ell}, \quad \mathsf{H}_1(\vec{\boldsymbol{\rho}}_\ell) := g^{\beta_\ell}.$$

The indistinguishability between games $\mathbf{G}_6$ and $\mathbf{G}_7$ is a crucial step in our proof, and we prove this assuming the hardness of DDH in the group $\mathbb{G}$. We also rely on the following corollary of Lemma 1.

Corollary 1. *For any $n, q \in \mathsf{poly}(\lambda)$, and for any $\alpha_h, \alpha_v \in \mathbb{Z}_p^*$, assuming hardness of the* DDH *assumption in the group $\mathbb{G}$, the following two distributions are indistinguishable:*

$$\mathcal{D}'_0 := \left(g, \alpha_h, \alpha_v, \{(g^{\beta_i}, g^{-\gamma_i}, c_i)\}_{i\in[q]}\}\right) \text{ where} \begin{cases} \alpha \leftarrow\$ \mathbb{Z}_p, \\ \beta_i, c_i, \tilde{\gamma}_i \leftarrow\$ \mathbb{Z}_p, \\ \gamma_i := (\tilde{\gamma}_i + \alpha_h \cdot \beta_i) \cdot \alpha_v^{-1} + c_i \cdot \alpha. \end{cases}$$

$$\mathcal{D}'_1 := \left(g, \alpha_h, \alpha_v, \{(g^{\beta_i}, g^{-\gamma_i}, c_i)\}_{i\in[q]}\right) \text{ where} \begin{cases} \alpha \leftarrow\$ \mathbb{Z}_p, \\ \beta_i, c_i \leftarrow\$ \mathbb{Z}_p, \\ \gamma_i := (\alpha \cdot \beta_i + \alpha_h \cdot \beta_i) \cdot \alpha_v^{-1} + c_i \cdot \alpha. \end{cases}$$

Proof. Fix $\alpha_h, \alpha_v \in \mathbb{Z}_p^*$. Given a sample $(g, g^\alpha, \{(g^{\beta_i}, g^{\gamma_i})\}_{i\in[q]})$ from $\mathcal{D}_b$ for either $b \in \{0,1\}$ as defined in Lemma 1, we can get a sample from $\mathcal{D}'_b$ as follows:

1. For all $i \in [q]$, sample $c_i \leftarrow\$ \mathbb{Z}_p$, define $g^{\beta'_i} := g^{\beta_i}$, and compute
$$g^{\gamma'_i} := (g^{\gamma_i})^{\alpha_v^{-1}} \cdot (g^{\beta_i})^{\alpha_h \cdot \alpha_v^{-1}} \cdot (g^\alpha)^{c_i}. \tag{7}$$
2. Output the tuple $(g, \alpha_h, \alpha_v, \{(g^{\beta'_i}, g^{-\gamma'_i}, c_i)\}_{i\in[q]})$.

We now analyze the distribution from which the tuple in step (2) above is sampled. When $b = 0$, then for all $i \in [q]$, $g^{-\gamma'_i}$ is uniformly random. Thus, the tuple in step (2) above is a sample from $\mathcal{D}'_0$. Similarly, when $b = 1$, then $g^{\gamma_i} = g^{\alpha\cdot\beta_i}$ for all $i \in [q]$. Thus, $g^{-\gamma'_i}$ in equation (7) is correctly distributed as in $\mathcal{D}'_1$. Consequently, if a PPT adversary $\mathcal{A}$ can distinguish between a sample from $\mathcal{D}'_0$ and $\mathcal{D}'_1$ with probability ε, then $\mathcal{A}$ can distinguish between $\mathcal{D}_1$ and $\mathcal{D}_2$ also with probability ε. Thus, from Lemma 1, we then get $\varepsilon \leq \varepsilon_{\mathsf{ddh}} + 1/p$. □

It is easy to see that in game $\mathbf{G}_6$ we use a sample from the distribution $\mathcal{D}'_0$ to program the random oracles H_0 and H_1, whereas in game $\mathbf{G}_7$ we use a sample from the distribution $\mathcal{D}'_1$. Therefore, the advantage of $\mathcal{A}$ in distinguishing between these two games $\mathbf{G}_6$ and $\mathbf{G}_7$ is at most $\varepsilon_{\mathsf{ddh}} + 1/p$, and we get

$$|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \varepsilon_{\mathsf{ddh}} + \frac{1}{p}.$$

<u>Game $\mathbf{G}_8$</u>: This game is identical to game $\mathbf{G}_7$, except we now use simulated NIZK proofs $\bar{\pi}_i$ for honest signers. For each NIZK simulation, the game programs the random oracle H_{FS} on input $\mathsf{stm} := (X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}, A, c, z, g_0, g_1)$ at its choice of a challenge and aborts if $\mathsf{H}_{\mathsf{FS}}(\mathsf{stm})$ is already defined.[‡] Clearly, since the elements $X_{\mathsf{pk}}, X_A \leftarrow\$ \mathbb{G}$, $X_z \leftarrow\$ \mathbb{Z}_p$ are sampled uniformly random and hidden from $\mathcal{A}$ (before we output the NIZK proof), the probability that the game aborts is at most q_{FS}/p^3. By a union bound over all signing queries, we get

$$|\Pr[\mathbf{G}_7 \Rightarrow 1] - \Pr[\mathbf{G}_8 \Rightarrow 1]| \leq \frac{q_s \cdot q_{\mathsf{FS}}}{p^3}.$$

<u>Game $\mathbf{G}_9$</u>: In this game, we change how we sample α (which we introduce in game $\mathbf{G}_6$). 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 $\mathbf{G}_9$ is also uniformly random and independent. Thus, we get $\Pr[\mathbf{G}_8 \Rightarrow 1] = \Pr[\mathbf{G}_9 \Rightarrow 1]$.

<u>Game $\mathbf{G}_{10}$</u>: In this game, we change how we sample the signing keys. To illustrate our modification, we will distinguish between the signing key polynomials of games $\mathbf{G}_9$ and $\mathbf{G}_{10}$. More precisely, let $(s_9(x), r_9(x), u_9(x))$ and $(s_{10}(x), r_{10}(x), u_{10}(x))$ be the signing key polynomials in game $\mathbf{G}_9$ and game $\mathbf{G}_{10}$, respectively. Then, in game $\mathbf{G}_9$ we sample the signing key polynomial $s_{10}(x) := s_9(x) + \alpha$ for α we define in the previous game. The other two signing key polynomials remain unchanged, i.e., $r_{10}(x) := r_9(x)$ and $u_{10}(x) := u_9(x)$.

[‡]Note that our NIZK protocol has perfect honest-verifier zero-knowledge (HVZK).

Observe that for any fixed α, since $s_9(x)$ is a random degree-t polynomial, $s_{10}(x) := s_9(x) + \alpha$ is also a random degree-t polynomial. Hence, $\mathcal{A}$'s view in game $\mathbf{G}_9$ is identically distributed to its view in game $\mathbf{G}_{10}$. As a result, we get $\Pr[\mathbf{G}_9 \Rightarrow 1] = \Pr[\mathbf{G}_{10} \Rightarrow 1]$.

Game $\mathbf{G}_{11}$: In this game, we change how we sample the signing keys again. More precisely, we sample signing key polynomials such that $s_{11}(x) := s_9(x)$, $r_{11}(x) := r_9(x) + 1$, and $u_{11}(x) := u_9(x) + u$, for the uniformly random $u \in \mathbb{Z}_p$ (we introduce in game $\mathbf{G}_9$) we used to define $\alpha = \alpha_h + \alpha_v u$.

The indistinguishability between $\mathcal{A}$'s view in these two games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$ is another crucial step of our proof. To prove this, we will use Lemma 2 and Patarin's H-coefficient technique [Pat08, HT16].

Lemma 3. $\Pr[\mathbf{G}_{10} \Rightarrow 1] = \Pr[\mathbf{G}_{11} \Rightarrow 1]$.

Proof. We prove this lemma using the H-coefficient technique [Pat08, CS14]. Let T_0 and T_1 be the random variables denoting the transcripts of $\mathcal{A}$'s interaction in games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$, respectively. Then, for any potential value τ of T_θ for $\theta \in \{0,1\}$, let $\mathsf{p}_0(\tau)$ and $\mathsf{p}_1(\tau)$ be the interpolation probabilities, i.e., the probabilities of choosing randomness in the respective game that would lead to transcript τ, if the corruption set, the signing queries, and the random oracle queries are fixed in advance. These probabilities depend solely on τ and the game's randomness, and are independent of $\mathcal{A}$. The H-coefficient technique [Pat08] now tells us that, to argue indistinguishability between games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$, it it sufficient to show that for all possible transcripts τ, $\mathsf{p}_0(\tau) = \mathsf{p}_1(\tau)$. We reiterate that we fix $\mathcal{A}$'s queries when computing the interpolation probabilities.

Since the interpolation probabilities are independent of $\mathcal{A}$, instead of working with the random variables T_0 and T_1, we can work with the marginal transcript random variables we get after fixing $\mathcal{A}$'s queries. Let $W_0 = (X_0, Y_0, f(X_0, Y_0))$ and $W_1 = (X_1, Y_1, f(X_1, Y_1))$ be the marginal transcript random variables of game $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$, respectively, where:

$$X_0 := (\alpha_h, \alpha_v, s_8(0), u, \{s_{10}(i), r_{10}(i), u_{10}(i)\}_{i \in \mathcal{C}}, \{(c_j, \beta_j, \{A_{i,j}, z_{i,j}\}_{i \in \mathsf{SS}_j \cap \mathcal{H}})\}_{j \in [q_s]}),$$
$$X_1 := (\alpha_h, \alpha_v, s_8(0), u, \{s_{11}(i), r_{11}(i), u_{11}(i)\}_{i \in \mathcal{C}}, \{(c_j, \beta_j, \{A_{i,j}, z_{i,j}\}_{i \in \mathsf{SS}_j \cap \mathcal{H}})\}_{j \in [q_s]}).$$

Y_0 (resp. Y_1) denotes the random variable for: (i) the Sig_1, Sig_2, and Sig_3 messages of honest signers; (ii) the outputs of the random oracle H_b for both $b \in \{0,1\}$ for all inputs except for the Sig_1 messages of any signing session; (iii) all outputs of $\mathsf{H}_{\mathsf{view}}$; (iv) the outputs of $\mathsf{H}_{\mathsf{sig}}$ on all inputs except for the inputs the game programs $\mathsf{H}_{\mathsf{sig}}$ on by extracting $\hat{A}$ during any signing session; (v) all $\{a_{i,j}\}_{i,j}$ values for $i \in \mathsf{SS}_j \cap \mathcal{C}$ that the game samples for the j-th signing session (with signer set SS_j) before $\mathcal{A}$ corrupts the signer i; (vi) the simulated NIZK proofs of all honest signers. It is easy to see that Y_0 (resp. Y_1) is independent of X_0 (resp. X_1). Also, by design of games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$, Y_0 is identically distributed as Y_1.

We now argue that for any fixed queries of $\mathcal{A}$, given (X_θ, Y_θ) for either $\theta \in \{0,1\}$, the rest of the transcript is a deterministic function of (X_θ, Y_θ). We also argue that in both games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$, this (deterministic) function is the same, and we use $f(\cdot,\cdot)$ to denote it, as we describe below:

- Let $\alpha := \alpha_h + u \cdot \alpha_v$. Then, the H_0 outputs on the first-round messages of all signing sessions are a deterministic function of $(\alpha_h, \alpha_v, \alpha, \{\beta_j, c_j\}_{j \in [q_s]}, Y_\theta)$.
- The discrete logarithm of the public key in both games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$ is the same and is equal to $s_8(0) + \alpha$. More precisely, $\mathsf{pk}_{\mathbf{G}_{10}} = g^{s_8(0)+\alpha}$ by definition. Also, recall that we have $r_{11}(0) = 1$ and $u_{11}(0) = u$. Therefore,
$$\mathsf{pk}_{\mathbf{G}_{11}} = g^{s_{11}(0)} h^{r_{11}(0)} v^{u_{11}(0)} = g^{s_8(0)} h v^u = g^{s_8(0)+\alpha_h+\alpha_v u} = g^{s_8(0)+\alpha}.$$
Since $|\mathcal{C}| = t$, given $s_8(0) + \alpha$ and the signing keys of signers in $\mathcal{C}$, the threshold public keys of all signers are fixed and a deterministic function of these values.
- The combined nonces and final signatures are deterministic function of $\{c_j, \{(A_{i,j}, z_{i,j})\}_{i \in \mathsf{SS}_j \cap \mathcal{H}}\}_{j \in [q_s]}$, the signing keys of the corrupt signers, $\mathcal{A}$'s internal state, and Y_θ.
- The random oracle outputs of $\mathsf{H}_{\mathsf{com}}$ are also deterministic function of (X_θ, Y_θ).

Given Lemma 2, to prove that games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$ are identically distributed, it remains to show that X_0 and X_1 are identically distributed, i.e., $X_0 \equiv X_1$. Concretely, for any potential value τ of X_θ for $\theta \in \{0,1\}$, that we denote as

$$\tau = \Big(\underline{\alpha}_h, \underline{\alpha}_v, \underline{s}, \underline{u}, \{\underline{s}_i, \underline{r}_i, \underline{u}_i\}_{i\in\mathcal{C}}\,, \big\{\big(\underline{c}_j, \underline{\beta}_j, \{\underline{A}_{i,j}, \underline{z}_{i,j}\}_{i\in\mathsf{SS}_j\cap\mathcal{H}}\big)\big\}_{j\in[q_s]}\Big)\,,$$

let $\mathsf{p}_\theta(\tau) = \Pr[X_\theta = \tau]$ for either $\theta \in \{0,1\}$. For $\mathsf{p}_0(\tau)$, note that the randomness consists of $\alpha_h, \alpha_v \leftarrow\!\$\; \mathbb{Z}_p^*$ and $s_8(0), u \leftarrow\!\$\; \mathbb{Z}_p$. To generate a particular transcript τ, we therefore need the identities:

$$\alpha_h = \underline{\alpha}_h, \quad \alpha_v = \underline{\alpha}_v, \quad u = \underline{u}, \quad s_8(0) = \underline{s}.$$

Since $(\alpha_h, \alpha_v, s_8(0), u)$ are chosen independently, this is true with probability

$$\frac{1}{p-1}\cdot\frac{1}{p-1}\cdot\frac{1}{p}\cdot\frac{1}{p} = \frac{1}{p^2(p-1)^2}. \tag{8}$$

Further, we need to ensure that $\{(s_8(i), r_8(i), u_8(i))\}_{i\in\mathcal{C}} = \{(\underline{s}_i, \underline{r}_i, \underline{u}_i)\}_{i\in\mathcal{C}}$. Since $|\mathcal{C}| = t$, conditioned on $(\alpha_h, \alpha_v, s_8(0), u) = (\underline{\alpha}_h, \underline{\alpha}_v, \underline{s}, \underline{u})$, there exists a unique set of three polynomials of degree at most t each, with constant terms being equal to $(\underline{s} + \underline{\alpha}, 0, 0)$ for $\underline{\alpha} := \underline{\alpha}_h + \underline{u}\cdot\underline{\alpha}_v$, such that the above equality holds. Since in game $\mathbf{G}_{10}$, we sample the t additional coefficients of each of these polynomials uniformly at random, the equality holds with probability $1/p^{3t}$.

For all $j \in [q_s]$, the values c_j and β_j are uniformly random. Thus, the probability of obtaining a particular sequence of tuples $(\underline{c}_j, \underline{\beta}_j)_{j\in[q_s]}$ is $1/p^{2q_s}$. Next, consider the tuples $\{(A_{i,j}, z_{i,j})\}_{i\in\mathcal{H}\cap\mathsf{SS}_j, j\in[q_s]}$. For each (i,j), we have:

$$\begin{aligned} A_{i,j} &= \left(g^{a_{i,j}}\mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_{10}(i)}\mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_{10}(i)}\right)^{L_{i,\mathsf{SS}_j}} = \left(g^{a_{i,j}}\mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_8(i)}\mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_8(i)}\right)^{L_{i,\mathsf{SS}_j}}\,, \\ z_{i,j} &= L_{i,\mathsf{SS}_j}\cdot(a_{i,j} + c_j\cdot s_{10}(i)) = L_{i,\mathsf{SS}_j}\cdot(a_{i,j} + c_j\cdot(s_8(i) + \alpha)), \end{aligned}$$

where $a_{i,j} \leftarrow\!\$\; \mathbb{Z}_p$ is the randomness of honest signer i in the j-th signing session (with signer set SS_j), and $\vec{\boldsymbol{\rho}}_j$ and c_j are the Sig_1 message and the challenge of the j-th signing session, respectively. Now, given that everything else is fixed, the tuple $(A_{i,j}, z_{i,j})$ is a function of $a_{i,j}$. Thus, the probability that $(A_{i,j}, z_{i,j}) = (\underline{A}_{i,j}, \underline{z}_{i,j})$ holds is equal to the probability that

$$a_{i,j} = \underline{a}_{i,j} \text{ for } \underline{a}_{i,j} \in \mathbb{Z}_p. \tag{9}$$

Since $a_{i,j}$ is chosen uniformly at random, the probability of $a_{i,j} = \underline{a}_{i,j}$ is $1/p$. As each honest signer i samples its value $a_{i,j}$ independently, the probability of $\{(A_{i,j}, z_{i,j}) = (\underline{A}_{i,j}, \underline{z}_{i,j})\}_{j\in[q_s], i\in\mathsf{SS}_j\cap\mathcal{H}}$ is then $1/p^k$, where k is the total number of honest signers across all signing sessions.

Combining all of the above, for all transcripts $\tau \in W_0$ with $\mathsf{p}_0(\tau) > 0$, we get the following interpolation probability:

$$\mathsf{p}_0(\tau) = \frac{1}{p^2(p-1)^2}\cdot\frac{1}{p^{3t}}\cdot\frac{1}{p^{2q_s}}\cdot\frac{1}{p^k}.$$

We now turn to the analysis of $\mathsf{p}_1(\tau)$. First, by a similar argument as for the calculation of $\mathsf{p}_0(\tau)$, we need the identities

$$\alpha_h = \underline{\alpha}_h, \quad \alpha_v = \underline{\alpha}_v, \quad u = \underline{u}, \quad s_8(0) = \underline{s},$$

which again are true with probability

$$\frac{1}{p-1}\cdot\frac{1}{p-1}\cdot\frac{1}{p}\cdot\frac{1}{p} = \frac{1}{p^2(p-1)^2}. \tag{10}$$

Again, for the required identity $\{(s_{11}(i), r_{11}(i), u_{11}(i))\}_{i\in\mathcal{C}} = \{(\underline{s}_i, \underline{r}_i, \underline{u}_i)\}_{i\in\mathcal{C}}$, conditioned on $(\alpha_h, \alpha_v, s_8(0), u) = (\underline{\alpha}_h, \underline{\alpha}_v, \underline{s}, \underline{u})$, there is a unique set of three degree-t polynomials with constant terms $(\underline{s}, 1, \underline{u})$ for $\underline{\alpha} := \underline{\alpha}_h + \underline{u}\cdot\underline{\alpha}_v$ such that this identity holds. Since in game $\mathbf{G}_{11}$, we sample the t additional coefficients of each of these polynomials uniformly at random, the equality holds with probability $1/p^{3t}$. Further, for all $j \in [q_s]$, the values c_j and β_j are uniformly random. Thus, we get a particular tuple $(\underline{c}_j, \underline{\beta}_j)_{j\in[q_s]}$ with probability $1/p^{2q_s}$.

Lastly, consider the tuples $\{(A_{i,j}, z_{i,j})\}_{i\in\mathcal{H}\cap\mathsf{SS}_j, j\in[q_s]}$. For each (i,j), for uniformly random $a_{i,j}$, we have $z_{i,j} = L_{i,\mathsf{SS}_j}(a_{i,j} + c_j \cdot s_{11}(i)) = L_{i,\mathsf{SS}_j}(a_{i,j} + c_j \cdot s_8(i))$. Further, using $\mathsf{H}_0(\vec{\boldsymbol{\rho}}_j) = g^{-u\cdot\beta_j - c_j\cdot\alpha}$ and $\mathsf{H}_1(\vec{\boldsymbol{\rho}}_j) = g^{\beta_j}$, we get

$$\begin{aligned}
A_{i,j} &= \left(g^{a_{i,j}} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_{11}(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_{11}(i)}\right)^{L_{i,\mathsf{SS}_j}} \\
&= \left(g^{a_{i,j}} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{1+r_8(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u+u_8(i)}\right)^{L_{i,\mathsf{SS}_j}} \\
&= \left(g^{a_{i,j}} g^{-u\cdot\beta_j - c_j\alpha} g^{u\cdot\beta_j} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_8(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_8(i)}\right)^{L_{i,\mathsf{SS}_j}} \\
&= \left(g^{a_{i,j} - c_j\cdot\alpha} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_8(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_8(i)}\right)^{L_{i,\mathsf{SS}_j}},
\end{aligned}$$

where $a_{i,j} \leftarrow\$\ \mathbb{Z}_p$ is the randomness of honest signer i in the j-th signing session (with signer set SS_j), and $\vec{\boldsymbol{\rho}}_j$ and c_j are the Sig_1 message and the challenge of the j-th signing session, respectively. Let $\tilde{a}_{i,j} := a_{i,j} - c_j \cdot \alpha$, then

$$A_{i,j} = \left(g^{\tilde{a}_{i,j}} \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}}_j)^{r_8(i)} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}}_j)^{u_8(i)}\right)^{L_{i,\mathsf{SS}_j}}, \tag{11}$$

$$z_{i,j} = L_{i,\mathsf{SS}_j} \cdot (\tilde{a}_{i,j} + c_j \cdot (s_8(i) + \alpha)). \tag{12}$$

Now, given that everything else is fixed, both $A_{i,j}$ and $z_{i,j}$ are a function of $\tilde{a}_{i,j} := a_{i,j} - c_j \cdot \underline{\alpha}$. Therefore, we get $(A_{i,j}, z_{i,j}) = (\underline{A}_{i,j}, \underline{z}_{i,j})$ only if $a_{i,j} - \underline{c}_j \cdot \alpha = \underline{a}_{i,j}$ for the same $\underline{a}_{i,j} \in \mathbb{Z}_p$ we used in equation (9). Since $a_{i,j}$ is chosen uniformly at random, the probability of $a_{i,j} - \underline{c}_j \cdot \underline{\alpha} = \underline{a}_{i,j}$ is also $1/p$. As each signer i selects its $a_{i,j}$ independently, the probability of $\{(A_{i,j}, z_{i,j}) = (\underline{A}_{i,j}, \underline{z}_{i,j})\}_{j\in[q_s], i\in\mathsf{SS}_j\cap\mathcal{H}}$ is $1/p^k$, where k is the total number of honest signers across all signing sessions.

Combining all of the above, for all transcripts $\tau \in W_1$ with $\mathsf{p}_1(\tau) > 0$, we get the following interpolation probability:

$$\mathsf{p}_1(\tau) = \frac{1}{p^2(p-1)^2} \cdot \frac{1}{p^{3t}} \cdot \frac{1}{p^{2q_s}} \cdot \frac{1}{p^k}.$$

Since the above holds for any (possible) transcripts, we get $\mathsf{p}_0(\tau) = \mathsf{p}_1(\tau)$ for all transcripts. This implies that $X_0 \equiv X_1$. Finally, by Lemma 2, we get that the view of $\mathcal{A}$ in the games $\mathbf{G}_{10}$ and $\mathbf{G}_{11}$ is identically distributed. □

From the above sequence of games, we finally get

$$\begin{aligned}
\varepsilon_{\mathbf{G}_{11}} := \Pr[\mathbf{G}_{11} \Rightarrow 1] \geq \varepsilon_\sigma - \Bigg(&\frac{q_{\mathsf{H}}^2}{p} + \frac{q_{\mathsf{com}}^2}{|\mathcal{R}|} + \frac{q_{\mathsf{view}}^2}{|\mathcal{Y}|} + \frac{q_s^2}{2^\lambda} + \frac{q_s^2}{|\mathcal{R}|} + \frac{q_s \cdot q_{\mathsf{com}}}{p} \\
&+ \frac{q_s}{|\mathcal{R}|} + \frac{q_s \cdot q_{\mathsf{sig}}}{p} + \frac{q_s \cdot q_{\mathsf{FS}}}{p^3} + \varepsilon_{\mathsf{ddh}} + \frac{1}{p}\Bigg).
\end{aligned} \tag{13}$$

Combining all the above, we get our following main theorem.

Theorem 1 (Adaptively Secure Threshold Schnorr Signature). *For any $n \in \mathsf{poly}(\lambda)$ and $t < n$, assuming hardness of* DDH *and* DL *in the group $\mathbb{G}$, and assuming the random oracle model (ROM), any PPT adversary making at most q_{H} random oracle queries to H_0 and H_1 combined, at most q_{view} random oracle queries $\mathsf{H}_{\mathsf{view}}$, at most q_{com} random oracle queries to $\mathsf{H}_{\mathsf{com}}$, at most q_{sig} random oracle queries to $\mathsf{H}_{\mathsf{sig}}$, and at most q_s signing queries, wins the $\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ game Figure 2 for our scheme in Figure 3 with probability at* *most ε_σ where:*

$$\varepsilon_\sigma \leq \frac{q_{\mathsf{sig}}}{p} + \sqrt{q_{\mathsf{sig}} \cdot \varepsilon_{\mathsf{dl}}} + \frac{q_{\mathsf{H}}^2}{p} + \frac{q_{\mathsf{com}}^2}{|\mathcal{R}|} + \frac{q_{\mathsf{view}}^2}{|\mathcal{Y}|} + \frac{q_s^2}{2^\lambda} + \frac{q_s^2}{|\mathcal{R}|} + \frac{q_s \cdot q_{\mathsf{com}}}{p} + \frac{q_s}{|\mathcal{R}|} + \frac{q_s \cdot q_{\mathsf{sig}}}{p} + \frac{q_s \cdot q_{\mathsf{FS}}}{p^3} + \varepsilon_{\mathsf{ddh}} + \frac{1}{p}.$$

Here, $\varepsilon_{\mathsf{dl}}$ and $\varepsilon_{\mathsf{ddh}}$ are the advantages of an adversary running in $T \cdot \mathsf{poly}(\lambda, q_{\mathsf{H}}, n)$ time in breaking the DL *and* DDH *assumption in $\mathbb{G}$, respectively. This implies that ε_σ is negligible, and hence, our threshold signature scheme in §3 is unforgeable.*

Proof. To prove Theorem 1, we rely on the generalized forking lemma [PS96, BN06]. More specifically, given an adversary $\mathcal{A}$ that forges a signature in game $\mathbf{G}_{11}$, we build a "wrapping" algorithm $\mathcal{B}$ (see Figure 4) which runs $\mathcal{A}$ and returns information regarding the forgery. We then use $\mathcal{B}$ to construct an algorithm $\mathcal{A}_{\mathsf{dl}}$ (see Figure 6) that first runs the forking algorithm $\mathsf{Fork}^{\mathcal{B}}$ (see Figure 5) which forks $\mathcal{B}$ with respect to a $\mathsf{H}_{\mathsf{sig}}$ query. Algorithm $\mathcal{A}_{\mathsf{dl}}$ then uses the output of the $\mathsf{Fork}^{\mathcal{B}}$ algorithm to solve for discrete logarithm in $\mathbb{G}$. In this proof, for notational simplicity, we will use $q := q_{\mathsf{sig}}$.

Input: Generators $(g, h, v) \in \mathbb{G}^3$, secret key polynomials $s(\cdot), r(\cdot), u(\cdot)$, randomness for random oracle programming $\{h_1, h_2, \ldots, h_q\} \leftarrow\!\$\ \mathbb{Z}_p$.

KGen simulation

1. Use (g, h, v) as the generators and $s(\cdot), r(\cdot), u(\cdot)$ as the secret key polynomials.

Corruption simulation:

2. When $\mathcal{A}$ corrupts a signer $i \in \mathcal{H}$ if $|\mathcal{C}| < t$:
   (a) Update $\mathcal{H} := \mathcal{H} \setminus \{i\}$ and $\mathcal{C} := \mathcal{C} \cup \{i\}$.
   (b) Faithfully reveals the internal state of signer i to $\mathcal{A}$.

Simulating random oracle queries:

3. Simulate random oracles $\mathsf{H}_{\mathsf{com}}, \mathsf{H}_0, \mathsf{H}_1$ as per game $\mathbf{G}_{11}$.
4. For all inputs in which game $\mathbf{G}_{11}$ programs $\mathsf{H}_{\mathsf{sig}}$ after extracting the combined nonce as we describe in game $\mathbf{G}_5$, program $\mathsf{H}_{\mathsf{sig}}$ as in game $\mathbf{G}_{11}$.
5. For all other $\mathsf{H}_{\mathsf{sig}}$ queries, use the input $\{h_1, \ldots, h_q\}$ for programming the random oracle output.

Simulating signing protocol for any signing session

6. Follow the strategy of game $\mathbf{G}_{11}$ until $\mathcal{A}$ outputs a forgery $(m, \sigma = (\hat{A}, z))$.
7. Identify the random oracle query to $\mathsf{H}_{\mathsf{sig}}$ with input $(\hat{A}, \mathsf{pk}, m^*)$ such that $c = \mathsf{H}_{\mathsf{sig}}(\hat{A}, \mathsf{pk}, m)$. Let idx be the index, i.e., $c = h_{\mathsf{idx}}$.
8. If no such idx exists, **return** $(0, \perp)$. Otherwise, **return** (idx, z)

Fig. 4: Description of Algorithm $\mathcal{B}$ that simulates game $\mathbf{G}_{11}$ to the adversary $\mathcal{A}$.

Description of algorithm $\mathcal{B}$ (Figure 4). $\mathcal{B}$ takes as input the public parameters (g, h, v), the signing keys $(s(\cdot), r(\cdot), u(\cdot))$, and a vector $\{h_1, h_2, \ldots, h_q\}$ of uniformly random field elements. $\mathcal{B}$ then interacts with $\mathcal{A}$ with these inputs. During this interaction, $\mathcal{B}$ uses $\{h_1, h_2, \ldots, h_q\}$ to program the random oracle $\mathsf{H}_{\mathsf{sig}}$ on inputs where the game does not explicitly program $\mathsf{H}_{\mathsf{sig}}$ after extracting the combined nonce as we show in game $\mathbf{G}_5$. Simultaneously, $\mathcal{B}$ also locally checks for forgery (step 6 in Figure 10).

When $\mathcal{A}$ forges a signature, $\mathcal{B}$ identifies the $\mathsf{H}_{\mathsf{sig}}$ query associated with the forgery. Let $(m, \sigma = (\hat{A}, z))$ be the forgery. $\mathcal{B}$ finds the index idx such that $\mathcal{B}$ programmed the $\mathsf{H}_{\mathsf{sig}}$ query on input $(\hat{A}, \mathsf{pk}, m)$ with c, for some $c = h_{\mathsf{idx}}$. $\mathcal{B}$ then returns the tuple (idx, z) as its output. If no such idx exists, $\mathcal{B}$ returns $(0, \perp)$.

Analysis of $\mathcal{B}$. The game programs $\mathsf{H}_{\mathsf{sig}}$ with values other than $\{h_1, \ldots, h_q\}$ only when $\mathcal{A}$ makes a signature query on the message. Therefore, given $(m, \sigma = (\hat{A}, z))$ is a valid forgery, $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{sig}}$ on the forged input $(\hat{A}, \mathsf{pk}, m)$ using a value from $\{h_1, \ldots, h_q\}$. This implies that when $\mathcal{A}$ outputs a valid forgery, $\mathcal{B}$ will always find such an index h_{idx}.

```
Algorithm $\mathsf{Fork}^{\mathcal{B}}(x = (g, h, v, s(\cdot), r(\cdot), u(\cdot)))$:
1: Sample randomness tape $\zeta$ for $\mathcal{B}$
2: Sample $h_1, h_2, \ldots, h_q \leftarrow\$ \mathbb{Z}_p$
3: Let $(\mathsf{idx}, z) := \mathcal{B}(g, h, v, s(\cdot), r(\cdot), u(\cdot), \{h_1, h_2, \ldots, h_q\}; \zeta)$
4: if $\mathsf{idx} = 0$ : return $(0, \perp, \perp)$
5: Sample $h'_{\mathsf{idx}}, \ldots, h'_q \leftarrow\$ \mathbb{Z}_p$
6: Let $(\mathsf{idx}', z) := \mathcal{B}(g, h, v, s(\cdot), r(\cdot), u(\cdot), \{h_1, h_2, h_{\mathsf{idx}-1}, h'_{\mathsf{idx}}, \ldots, h'_q\}; \zeta)$
7: if $\mathsf{idx} = \mathsf{idx}' \vee h_{\mathsf{idx}} \neq h'_{\mathsf{idx}}$ : return $(0, \perp, \perp)$
8: Let $\mathsf{out} := (h_{\mathsf{idx}}, z)$;    $\mathsf{out}' := (h'_{\mathsf{idx}}, z')$
9: return $(1, \mathsf{out}, \mathsf{out}')$
```

Fig. 5: Description of Algorithm $\mathsf{Fork}^{\mathcal{B}}$.

Description of algorithm $\mathsf{Fork}^{\mathcal{B}}$ (Figure 5). $\mathsf{Fork}^{\mathcal{B}}$ takes as input the generators (g, h, v) and the secret signing keys $(s(\cdot), r(\cdot), u(\cdot))$. $\mathsf{Fork}^{\mathcal{B}}$ samples the randomness tape ζ for $\mathcal{B}$ and the random oracle outputs $\{h_1, \ldots, h_q\}$. Next, $\mathsf{Fork}^{\mathcal{B}}$ runs $\mathcal{B}$ on these inputs. Let (idx, z) be the output of $\mathcal{B}$. If $\mathsf{idx} = 0$, then $\mathsf{Fork}^{\mathcal{B}}$ returns $(0, \perp, \perp)$. Otherwise, $\mathsf{Fork}^{\mathcal{B}}$ samples $\{h'_{\mathsf{idx}}, \ldots, h'_q\} \leftarrow\$ \mathbb{Z}_p$ uniformly at random. $\mathsf{Fork}^{\mathcal{B}}$ then runs the second execution of $\mathcal{B}$ by changing the random oracle programming starting at the index idx with $\{h'_{\mathsf{idx}}, \ldots, h'_q\}$.

Let (idx', z') be the output of $\mathcal{B}$ from the second execution. $\mathsf{Fork}^{\mathcal{B}}$ then checks whether $\mathsf{idx} = \mathsf{idx}'$ and $h_{\mathsf{idx}} \neq h'_{\mathsf{idx}}$. If both of these conditions hold, then $\mathsf{Fork}^{\mathcal{B}}$ returns $(1, \mathsf{out}, \mathsf{out}')$ where $\mathsf{out} := (h_{\mathsf{idx}}, z)$ and $\mathsf{out}' := (h'_{\mathsf{idx}}, z')$. Otherwise, $\mathsf{Fork}^{\mathcal{B}}$ returns $(0, \perp, \perp)$.

```
Algorithm $\mathcal{A}_{\mathsf{dl}}(\mathbb{G}, p, g, y)$:
1: Sample $\alpha_v \leftarrow\$ \mathbb{Z}_p^*$. Let $(g, h, v) := (g, y, g^{\alpha_v})$.
2: Sample $s(x), r(x), u(x) \leftarrow\$ \mathbb{Z}_p[x]_{(t)}$ s.t. $r(0) = 1$ and $u(0) \leftarrow\$ \mathbb{Z}_p$
3: Let $(\mathsf{val}, \mathsf{out}, \mathsf{out}') \leftarrow \mathsf{Fork}^{\mathcal{B}}(g, h, v, s(\cdot), r(\cdot), u(\cdot))$
4: if $\mathsf{val} = 0$ : return $\perp$
5: parse $(c, z) := \mathsf{out}$ and $(c', z') := \mathsf{out}'$.
6: return $\alpha_h = (z - z') \cdot (c - c')^{-1} - s(0) - u(0) \cdot \alpha_v$
```

Fig. 6: Description of Algorithm $\mathcal{A}_{\mathsf{dl}}$ solves discrete logarithm in $\mathbb{G}$

Description of algorithm $\mathcal{A}_{\mathsf{dl}}$ (Figure 6). $\mathcal{A}_{\mathsf{dl}}$ takes as input $(\mathbb{G}, p, g, y)$: the description of the group $\mathbb{G}$ of order p, a generator g and a uniformly random element $y \in \mathbb{G}$. $\mathcal{A}_{\mathsf{dl}}$ then samples uniformly random $\alpha_v \leftarrow\$ \mathbb{Z}_p$ and sets the public parameters $(g, h, v) := (g, y, g^{\alpha_v})$. Next, $\mathcal{A}_{\mathsf{dl}}$ samples the signing key polynomials $s(\cdot), r(\cdot), u(\cdot)$ as per game $\mathbf{G}_{11}$. $\mathcal{A}_{\mathsf{dl}}$ then runs $\mathsf{Fork}^{\mathcal{B}}$ with $(g, h, v, s(\cdot), r(\cdot), u(\cdot))$ as input. Let $(\mathsf{val}, \mathsf{out}, \mathsf{out}')$ be the output of $\mathsf{Fork}^{\mathcal{B}}$. If $\mathsf{val} = 0$, then $\mathcal{A}_{\mathsf{dl}}$ returns $\perp$. Otherwise, let $\mathsf{out} = (c, z)$ and $\mathsf{out}' = (c', z')$.

Then, $\mathcal{A}_{\mathsf{dl}}$ outputs α_h as the discrete logarithm solution where

$$\alpha_h := \left(\frac{z - z'}{c - c'}\right) - (s(0) + u(0) \cdot \alpha_v) \,. \tag{14}$$

Analysis of $\mathcal{A}_{\mathsf{dl}}$. Let ε be the probability that $\mathsf{Fork}^{\mathcal{B}}$ outputs $(1, \mathsf{out}, \mathsf{out}')$. Also, let $\varepsilon_{\mathbf{G}_{11}}$ be the probability that $\mathcal{A}$ forges a signature in game $\mathbf{G}_{11}$. Then, using the generalized forking lemma (see Lemma 4), we get the identity

$$\varepsilon \geq \varepsilon_{\mathbf{G}_{11}} \cdot \left(\frac{\varepsilon_{\mathbf{G}_{11}}}{q} - \frac{1}{p}\right).$$

Now, note that $\mathsf{Fork}^{\mathcal{B}}$ outputting $(1, \mathsf{out}, \mathsf{out}')$ implies that $\mathcal{A}$ forges a signature for the first time during $\mathcal{B}$'s interaction with $\mathcal{A}$ for the idx-th $\mathsf{H}_{\mathsf{sig}}$ query for both executions of $\mathcal{B}$. Since $\mathcal{A}$'s view in both executions is identical until the idx-th query to $\mathsf{H_{sig}}$, the input to the idx-th $\mathsf{H_{sig}}$ is the same in both executions. Let $(\hat{A}, \mathsf{pk}, m)$ be that (identical) idx-th $\mathsf{H_{sig}}$ input. Then, the outputs $\mathsf{out} := (c, z)$ and $\mathsf{out}' = (c', z')$ of $\mathsf{Fork}^{\mathcal{B}}$ satisfy:

$$g^{z} = \hat{A} \cdot \mathsf{pk}^{c} \quad \text{and} \quad g^{z'} = \hat{A} \cdot \mathsf{pk}^{c'}.$$

Therefore, we get:

$$\left(g^{(z-z')(c-c')^{-1}} = \mathsf{pk} = g^{s(0)} h v^{u(0)}\right) \implies \left(h = g^{(z-z')(c-c')^{-1} - (s(0) + u(0)\cdot\alpha_v)}\right),$$

which implies that $\alpha_h := (z - z')(c - c')^{-1} - (s(0) + u(0) \cdot \alpha_v)$ is the discrete logarithm of y. Further, it also implies that whenever $\mathsf{Fork}^{\mathcal{B}}$ outputs $(1, \cdot, \cdot)$, $\mathcal{A}_{\mathsf{dl}}$ can output α_h. Therefore, we get:

$$\varepsilon_{\mathsf{dl}} \geq \varepsilon_{\mathbf{G}_{11}} \left(\frac{\varepsilon_{\mathbf{G}_{11}}}{q} - \frac{1}{p}\right) \implies \varepsilon_{\mathbf{G}_{11}} \leq \frac{q}{p} + \sqrt{q \cdot \varepsilon_{\mathsf{dl}}}. \tag{15}$$

Combining equation (13) with equation (15), we get the desired bound. □

Acknowledgments

This work is funded by the National Science Foundation award #2240976, Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585, and by the European Union, ERC-2023-StG-101116713. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

ADN06. Jesús F Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold rsa with adaptive and proactive security. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006.

AF04. Masayuki Abe and Serge Fehr. Adaptively secure feldman vss and applications to universally-composable threshold cryptography. In *Annual International Cryptology Conference*, pages 317–334. Springer, 2004.

BCJ08. Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 449–458, 2008.

BCK+22. Mihir Bellare, Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In *Annual International Cryptology Conference*, pages 517–550. Springer, 2022.

BGG+18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 565–596, Cham, 2018. Springer International Publishing.

BHK+24. Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. Sprint: High-throughput robust distributed schnorr signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 62–91. Springer, 2024.

BL22. Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 193–207, 2022.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*, pages 514–532. Springer, 2001.

BLSW24. Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. Harts: High-threshold, adaptively secure, and robust threshold schnorr signatures. *Cryptology ePrint Archive*, 2024.

BLT$^+$24. Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from ddh with full adaptive security. In *Advances in Cryptology-EUROCRYPT 2024: 43th Annual International Conference on the Theory and Applications of Cryptographic Techniques.*, 2024.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, volume 2567, pages 31–46. Springer, 2003.

BP23. Luís T. A. N. Brandão and Rene Peralta. Nist ir 8214c: First call for multi-party threshold schemes. `https://csrc.nist.gov/pubs/ir/8214/c/ipd`, 2023.

BTZ22. Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: Bls and frost. *Cryptology ePrint Archive*, 2022.

CATZ24. Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Partially non-interactive two-round lattice-based threshold signatures. Cryptology ePrint Archive, Paper 2024/467, 2024.

CGG$^+$20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, New York, NY, USA, 2020. Association for Computing Machinery.

CGJ$^+$99. Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 98–116. Springer, 1999.

CGRS23. Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical schnorr threshold signatures without the algebraic group model. In *Annual International Cryptology Conference*, pages 743–773. Springer, 2023.

CKM21. Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove schnorr assuming schnorr: security of multi-and threshold signatures. *Cryptology ePrint Archive*, 2021.

CKM23. Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive schnorr threshold signatures. In *Annual International Cryptology Conference*. Springer, 2023.

CKP$^+$23. Elizabeth Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. Berlin, Heidelberg, 2023. Springer-Verlag.

CS14. Shan Chen and John Steinberger. Tight security bounds for key-alternating ciphers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 327–350. Springer, 2014.

Dam02. Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, page 84, 2002.

Des88. Yvo Desmedt. Society and group oriented cryptography: A new concept. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*, pages 120–127. Springer, 1988.

DF89. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Springer, 1989.

dPKM$^+$24. Rafael del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 219–248, Cham, 2024. Springer Nature Switzerland.

DR24. Sourav Das and Ling Ren. Adaptively secure bls threshold signatures from ddh and co-cdh. In *Advances in Cryptology–CRYPTO 2024: 44nd Annual International Cryptology Conference, CRYPTO 2024, Santa Barbara, CA, USA*. Springer, 2024.

dra23. Distributed randomness beacon: Verifiable, unpredictable and unbiased random numbers as a service. `https://drand.love/docs/overview/`, 2023.

EKT24. Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VII*, page 387–424, Berlin, Heidelberg, 2024. Springer-Verlag.

FMY99a. Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure distributed public-key systems. In *European Symposium on Algorithms*, pages 4–27. Springer, 1999.

FMY99b. Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive rsa. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 180–194. Springer, 1999.

GG20. Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. *Cryptology ePrint Archive*, 2020.

GJKR96. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *Advances in Cryptology—EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15*, pages 354–371. Springer, 1996.

GJKR07. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 2007.

GKS24. Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In *Post-Quantum Cryptography: 15th International Workshop, PQCrypto 2024, Oxford, UK, June 12–14, 2024, Proceedings, Part II*, page 266–300, Berlin, Heidelberg, 2024. Springer-Verlag.

GS24. Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 370–400. Springer, 2024.

HT16. Viet Tung Hoang and Stefano Tessaro. Key-alternating ciphers and key-length extension: Exact bounds and multi-user security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 3–32, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

ic23. Internet computer: Chain-key cryptography. `https://internetcomputer.org/how-it-works/chain-key-technology/`, 2023.

JL00. Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pages 221–242. Springer, 2000.

KAB21. Handan Kılınç Alper and Jeffrey Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 157–188, Cham, 2021. Springer International Publishing.

KG21. Chelsea Komlo and Ian Goldberg. Frost: flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography*. Springer, 2021.

KRT24. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from mlwe/msis and dl with rewinding. In *Annual International Cryptology Conference*, pages 459–491. Springer, 2024.

Lin22. Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *Cryptology ePrint Archive*, 2022.

LJY14. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *Proceedings of the ACM symposium on Principles of distributed computing*, 2014.

LP01. Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*, pages 331–350. Springer, 2001.

Mak22. Nikolaos Makriyannis. On the classic protocol for MPC schnorr signatures. Cryptology ePrint Archive, Paper 2022/1332, 2022.

MMS$^+$24. Aikaterini Mitrokotsa, Sayantan Mukherjee, Mahdi Sedaghat, Daniel Slamanig, and Jenit Tomy. Threshold structure-preserving signatures: Strong and adaptive security under standard assumptions. In *IACR International Conference on Public-Key Cryptography*, 2024.

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

MXC$^+$16. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

Nak08. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

NR04. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262, 2004.

NRS21. Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: simple two-round schnorr multi-signatures. In *Annual International Cryptology Conference*, pages 189–221. Springer, 2021.

NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1717–1731, 2020.

Pat08. Jacques Patarin. The "coefficients h" technique. In *International Workshop on Selected Areas in Cryptography*, pages 328–345. Springer, 2008.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398. Springer, 1996.

PW23. Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 597–627. Springer, 2023.

Rab98. Tal Rabin. A simplified approach to threshold and proactive rsa. In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18*, pages 89–104. Springer, 1998.

RRJ$^+$22. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. Roast: Robust asynchronous schnorr threshold signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

SS01. Douglas R. Stinson and Reto Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *Information Security and Privacy*, pages 417–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

TZ23. Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 628–658. Springer, 2023.

Wat05. Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 114–127. Springer, 2005.

WQL09. Zecheng Wang, Haifeng Qian, and Zhibin Li. Adaptively secure threshold signature scheme in the standard model. *Informatica*, 20(4):591–612, 2009.

YMR$^+$19. Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

ZWHZ19. Michael Zochowski, Peng Wang, Carl Hua, and Michael Zochowski. Benchmarking hash and signature algorithms. *Logos Network*, Mar 2019.

```
Algorithm $\mathsf{Fork}^{\mathcal{A}}(x)$:
1: Pick the random coins $\rho$ of $\mathcal{A}$ at random
2: $\{h_1, \ldots, h_q\} \leftarrow\$ H^q$
3: $(k, \mathsf{aux}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q)$
4: if $k = 0$ : return $\perp$
5: $\{h'_k, \ldots, h'_q\} \leftarrow\$ H^q$
6: $(k', \mathsf{aux}') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{k-1}, h'_k, \ldots, h'_q)$
7: if $k \neq k'$ : return $\perp$
8: return $(k, \mathsf{aux}, \mathsf{aux}')$
```

Fig. 7: The generalized forking algorithm.

A Additional Preliminaries

We cover additional preliminaries. More precisely, we define the computational assumptions used in our security proof and the generalized forking lemma.

A.1 Computational Assumptions

Assumption 2 (DL) *We say that the discrete logarithm (DL) assumption holds, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathcal{A},\mathsf{GGen}}(\lambda) := \Pr\left[\mathcal{A}(g, g^{\alpha}) = \alpha \;\middle|\; (\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^{\lambda}),\ \alpha \leftarrow\$ \mathbb{Z}_p\right] = \varepsilon_{\mathsf{dl}}.$$

Assumption 3 (DDH) *We say that the decisional Diffie-Hellman (DDH) assumption holds, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A},\mathsf{GGen}}(\lambda) := \left|\Pr\left[\mathcal{A}(g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1 \;\middle|\; \begin{array}{r}(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^{\lambda}),\\ \alpha, \beta \leftarrow\$ \mathbb{Z}_p\end{array}\right] - \Pr\left[\mathcal{A}(g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1 \;\middle|\; \begin{array}{r}(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^{\lambda}),\\ \alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_p\end{array}\right]\right| = \varepsilon_{\mathsf{ddh}}.$$

A.2 Generalized Forking Lemma

We recall the generalized forking lemma [PS96, BN06].

Lemma 4. *Let $q \geq 1$ be an integer, and H be a set. Let $\mathcal{A}$ be a randomized algorithm that on input $x, h_1, h_2, \ldots, h_q$ outputs a pair (k, aux), where $k \in [0, q]$ and aux is a side output. Let IG be a randomized algorithm that generates x. The accepting probability of $\mathcal{A}$ is defined as:*

$$\mathsf{acc} = \Pr_{x \leftarrow \mathsf{IG}, h_1, h_2, \ldots, h_q \leftarrow\$ H}[(k, \mathsf{aux}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q) : k \neq 0].$$

Consider algorithm $\mathsf{Fork}^{\mathcal{A}}$ described in Figure 7. The accepting probability of $\mathsf{Fork}^{\mathcal{A}}$ is defined as:

$$\mathsf{frk} := \Pr_{x \leftarrow \mathsf{IG}}[\nu \leftarrow \mathsf{Fork}^{\mathcal{A}}(x) : \nu \neq \perp].$$

Then, we have:

$$\mathsf{frk} \geq \mathsf{acc}\left(\frac{\mathsf{acc}}{q} - \frac{1}{|H|}\right) \implies \mathsf{acc} \geq \frac{q}{|H|} + \sqrt{q \cdot \mathsf{frk}}.$$

B Identifiable Abort for Threshold Signatures

In this section, we explain why the identifiable abort (IA) definition in [RRJ$^+$22] is insufficient to capture some important scenarios that may occur when running an interactive threshold signatures scheme. These scenarios, in particular, include partially non-interactive schemes, to which Ruffing et al.'s definition is supposed to apply. In particular, their IA definition does not cover scenarios in which an adversary, $\mathcal{A}$, sends different messages to different honest signers. This allows $\mathcal{A}$ to falsely frame an honest signer as malicious to other honest signers. We illustrate this more clearly using the example of the $\mathsf{Frost3}$ threshold signature, a slightly simplified version of Frost [KG21], described by Ruffing et al. [RRJ$^+$22].

Short recap of $\mathsf{Frost3}$. To illustrate this issue, we recall the necessary details $\mathsf{Frost3}$ scheme. Let $\mathsf{SS} \subseteq [n]$ denote a set of $t+1$ signers that seeks to compute a signature on m. In the first round of $\mathsf{Frost3}$, each signer $i \in \mathsf{SS}$ samples two uniformly random secret nonces $d_i, e_i \leftarrow\$ \mathbb{Z}_p$ and then sends the commitments $(D_i, E_i) := (g^{d_i}, g^{e_i}) \in \mathbb{G}^2$ to the other signers. Upon receiving commitments from all signers $B := ((D_j, E_j))_{j \in \mathsf{SS}}$, each signer computes its signature share $z_i := c \cdot \mathsf{sk}_i + (d_i + \rho e_i)$. Here, ρ is the aggregation coefficient and c is the challenge, both of which are derived deterministically from $(\mathsf{pk}, \mathsf{SS}, B, m)$ by two independent hash functions. The validity of z_i can be checked via equation

$$g^{z_i} \stackrel{?}{=} \mathsf{pk}_i^c \cdot (D_i E_i^\rho), \tag{16}$$

given signer i's public key pk_i.

At first glance, this *(signature) share validation check* should suffice to detect misbehaving signers in a signing protocol. This is also the reason why we believe the authors in [RRJ$^+$22] introduce the additional "share validation" algorithm $\mathsf{ShareVal}$ for general (partially non-interactive, two-round) threshold signature schemes such as $\mathsf{Frost3}$. In essence, this algorithm would take as input $(\mathsf{pk}, \mathsf{SS}, B, m)$, all signers' public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and a signer's signature share z_i, and then output the result of the check in equation 16.

Concrete attack. The authors [RRJ$^+$22] define IA via a security game $\mathsf{IA\text{-}CMA}_{\mathsf{TS}}^{\mathcal{A}}$ (page 6, Figure 2) with an adversary $\mathcal{A}$. There, the adversary $\mathcal{A}$ controls all but one signer i^* and wins the game if at least one of the two following events happens in some signing session. (1) It sends presignatures (i.e., first-round messages) and signature shares that verify via $\mathsf{ShareVal}$ but result in an invalid combined signature for honest signer i^*. (2) The honest signer i^* outputs a presignature and signature shares that does not verify via $\mathsf{ShareVal}$.

The authors state that their IA property guarantees that $\mathsf{ShareVal}$ reliably identifies disruptive malicious signers who send wrong shares. We argue that their definition does not satisfy this notion of IA. The problem stems from their security game assuming only one honest signer. Instead, imagine a scenario in which there are two honest signers P_1 and P_2, and a malicious signer $P_{\mathcal{A}}$. Again, for our illustration, we use the $\mathsf{Frost3}$ scheme.

Upon receiving the presignatures R_1 and R_2 from honest signers P_1 and P_2, respectively, $P_{\mathcal{A}}$ sends *different* presignatures $R_3 \neq R_3'$ to P_1 and P_2, respectively. As such, signer P_1 receives the presignature vector $B_1 := (R_1, R_2, R_3)$, whereas signer P_2 receives the presignature vector $B_2 := (R_1, R_2, R_3')$. As described before, from these presignatures (more concrete, from $(\mathsf{pk}, \mathsf{SS}, B_i, m)$), each signer locally and deterministically derives an aggregation coefficient ρ_i and a challenge c_i. Note that, with overwhelming probability, these values will be different for the both the honest signers, i.e., $\rho_1 \neq \rho_2$ and thus $c_1 \neq c_2$. Then, each signer P_i will compute its signature share $z_i := c_i \cdot \mathsf{sk}_i + (d_i + \rho_i e_i)$ using these values and send it to the other signers. Now, upon receiving the share z_2 from signer P_2, the signer P_1 will run the share validation algorithm $\mathsf{ShareVal}$ by checking the equation

$$g^{z_2} \stackrel{?}{=} \mathsf{pk}_2^{c_1} \cdot (D_2 E_2^{\rho_1}),$$

where z_2 was honestly computed as $z_2 = c_2 \cdot \mathsf{sk}_2 + (d_2 + \rho_2 e_2)$ and thus by default satisfies the equation

$$g^{z_2} = \mathsf{pk}_2^{c_2} \cdot (D_2 E_2^{\rho_2}).$$

Note that each signer computes and verifies the signature shares with respect to its local view. Now, with overwhelming probability, we will have

$$\mathsf{pk}_2^{c_1} \cdot (D_2 E_2^{\rho_1}) \neq \mathsf{pk}_2^{c_2} \cdot (D_2 E_2^{\rho_2}),$$

which implies that the honestly computed signature share z_2 will not verify at honest signer P_1. The converse is also true: the honestly computed signature share z_1 will not verify at honest signer P_2. As a consequence, each honest signer would blame the other honest signer for misbehaving. On the other hand, the actual malicious signer $P_{\mathcal{A}}$ could even send correct shares z_3 and z_3' to signer P_1 and P_2, respectively, and would not be blamed by either. Since the security game by Ruffing et al. considers the limited case where there is only one honest signer, this type of malicious behavior is not covered by their security definition.

Our solution. We find this unsatisfactory and therefore propose a (slightly adjusted) security game that captures this type of scenario. In our security game of IA, we want to be able to detect misbehaving signers, given the views of all the signers, where we let the adversary arbitrarily choose the views of malicious signers. This is different from the previous definition that only considers the view of one honest signer, which as explained above, does not suffice to detect misbehaving signers and could even result in blaming honest signers.

For this purpose, we introduce a detection algorithm Detect which takes as input the transcripts $(\mathsf{trx}_j)_{j \in \mathsf{SS}}$ (i.e., all received protocol messages) for a signing session and outputs a potentially empty set $\mathcal{J} \subset [n]$ of signers. In our security game, we then consider a signing session that failed for at least one honest signer i^* (i.e., the signer aborted) and allow $\mathcal{A}$ to specify all transcripts from corrupt signers. Then, $\mathcal{A}$ wins the game if the detect algorithm (i) identifies an honest signers as cheating (i.e., $\mathcal{J} \cap \mathcal{H} \neq \varnothing$), or (ii) does not identify any malicious signer (i.e., $\mathcal{J} \cap \mathcal{C} = \varnothing$).

Intuitively, with our definition we capture two types of adversarial behaviour. First, when the adversary *equivocates*, i.e., send different messages to different signers. This can be identified from the views of all honest signers. Second, when the adversary sends consistent (not equivocating) but invalid messages, e.g., signature shares that do not verify via some share validation algorithm. To implement such an algorithm in our construction, we use a public-key infrastructure (PKI) and let each signer sign its protocol messages. This allows the identification of cheating signers after a failed session, given all transcripts. We give our security game in Figure 2.

Final note on ROAST [RRJ+22]. Despite what we have said above, we emphasize that their notion of IA suffices to build their ROAST protocol. The reason for this is that all communication goes through coordinator nodes, which are the signers themselves. And for an honest coordinator, the above issue does not arise, since $P_{\mathcal{A}}$ cannot send him two different presignatures without being detected. Thus, equivocation cannot happen, and the only possibility for $\mathcal{A}$ to make a signing session fail is to send invalid signature shares, which can also directly be checked by the (honest) coordinator via equation 16.

C Analysis of the Identifiable Abort Property

C.1 Analysis of Σ-protocol

To prove unforgeability of $\mathsf{Glacius}$ we require the Σ-protocol to satisfy the standard honest-verifier zero-knowledge property (HVZK) property. Informally, the zero-knowledge property ensures that the proof reveals no information other than the statement's truth. For IA, we additionally require it to satisfy *completeness* and *soundness* in the concurrent setting. Briefly, the completeness property guarantees that an honest prover will always be able to convince an honest verifier about true statements. The soundness property prevents malicious prover from convincing a verifier about wrong statements. We refer the reader to [Dam02] for formal definitions of these properties.

The completeness of the Σ-protocol is straightforward. Similarly, the HVZK property also follows using standard techniques. Let $\mathcal{S}$ be the simulator. $\mathcal{S}$ samples uniformly random $(e, \beta_a, \beta_s, \beta_r, \beta_u) \in \mathbb{Z}_p^5$ and computes $(X_A, X_{\mathsf{pk}}, X_z)$ as

$$X_{\mathsf{pk}} := g^{\beta_s} h^{\beta_r} v^{\beta_u} \cdot \mathsf{pk}^{-e}; \quad X_A =: g^{\beta_a} g_0^{\beta_s} g_1^{\beta_r} \cdot A^{-e/L}; \quad X_z =: \beta_a + e \cdot \beta_s - \frac{ez}{L}.$$

```
Detect(SS, $m$, $(\mathsf{trx}_j)_{j\in \mathsf{SS}}$) :
1: $\forall j \in \mathsf{SS}$ : parse $(\mathsf{pm}^j_{k,i})_{k\in[R], i\in\mathsf{SS}} := \mathsf{trx}_j$
2: $\mathcal{J} := \varnothing$
   // Detect all equivocating signers
3: for $k \in [R],\ i, j, j' \in \mathsf{SS}$ :
4:     if $(\mathsf{pm}^j_{k,i} \neq \mathsf{pm}^{j'}_{k,i}) \wedge$ (both $\mathsf{pm}^j_{k,i}, \mathsf{pm}^{j'}_{k,i}$ have valid DS signatures from $i$) :
5:         $\mathcal{J} := \mathcal{J} \cup \{i\}$ // caught $i$ as equivocating
6:         continue // update $i$ with next signer in SS
   // Detect signers who sent invalid signature shares
7: $\mathsf{NE} := \mathsf{SS} \setminus \mathcal{J}$ // set of non-equivocating signers
8: for $i \in \mathsf{NE}$ :
9:     parse $(z_i, \pi_i) := \mathsf{pm}^i_{5,i}$ // $\mathsf{Sig}_5$ message signer $i$ sent to itself
10:    parse $\vec{\boldsymbol{\rho}}_{(i)} := (\mathsf{pm}^i_{1,j})_{j\in\mathsf{SS}},\ \ (A_j)_{j\in\mathsf{SS}} := (\mathsf{pm}^i_{3,j})_{j\in\mathsf{SS}}$
11:    $\hat{A}_{(i)} := \prod_{j\in\mathsf{SS}} A_j,\ \ c_{(i)} := \mathsf{H}_{\mathsf{sig}}(\hat{A}_{(i)}, \mathsf{pk}, m)$
12:    if $\mathsf{SigVer}(\mathsf{pk}_i, \vec{\boldsymbol{\rho}}_{(i)}, L_{i,\mathsf{SS}}, A_i, c_{(i)}, z_i, \pi_i) = 0$ :
13:        $\mathcal{J} := \mathcal{J} \cup \{i\}$
14: return $\mathcal{J}$
```

Fig. 8: The Detect algorithm for our scheme Glacius.

Input: $g, h, v, \mathsf{pk} \in \mathbb{G}$, $(g_0, g_1) = (\mathsf{H}_0(\vec{\boldsymbol{\rho}}), \mathsf{H}_1(\vec{\boldsymbol{\rho}}))$ for some $\vec{\boldsymbol{\rho}}$, $A \in \mathbb{G}$, $c, z \in \mathbb{Z}_p$, and some public L. Here, L is a Lagrange coefficient.

Witness: $(a, s, r, u) \in \mathbb{Z}_p^4$
The prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that it knows $s, r, u \in \mathbb{Z}_p$ such that $\mathsf{pk} = g^s h^r v^u$ and $A = (g^a g_0^s g_1^r)^L$, and $z = (a + c \cdot s) \cdot L$.

// We assume that both algorithms implicitly take $g, h, v, \mathsf{H}_0, \mathsf{H}_1$ as input

```
SigProve(pk, $A$, $\vec{\boldsymbol{\rho}}$, $(c, z)$; $(a, s, r, u)$):
1: Let $g_0 := \mathsf{H}_0(\vec{\boldsymbol{\rho}})$ and $g_1 := \mathsf{H}_1(\vec{\boldsymbol{\rho}})$
2: Sample $\alpha_a, \alpha_s, \alpha_r, \alpha_u \leftarrow\$\ \mathbb{Z}_p$. Let $X_{\mathsf{pk}} := g^{\alpha_s} h^{\alpha_r} v^{\alpha_u}$, and $X_A := g^{\alpha_a} g_0^{\alpha_r} g_1^{\alpha_u}$, and $X_z = \alpha_a + c \cdot \alpha_s$.
3: Let $e := \mathsf{H}_{\mathsf{FS}}(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}, A, c, z, g_0, g_1)$, for hash function $\mathsf{H}_{\mathsf{FS}} : \{0,1\}^* \to \mathbb{Z}_p$ modeled as a random oracle.
4: Let $\beta_a := \alpha_a + a \cdot e,\ \ \beta_s := \alpha_s + s \cdot e,\ \ \beta_r := \alpha_r + r \cdot e$ and $\beta_u := \alpha_u + u \cdot e$.
5: return $\pi := (X_{\mathsf{pk}}, X_A, X_z, \beta_a, \beta_s, \beta_r, \beta_u)$.

SigVer(pk, $\vec{\boldsymbol{\rho}}$, $L$, $A$, $(c, z)$, $\pi = (X_{\mathsf{pk}}, X_A, X_z, \beta_a, \beta_s, \beta_r, \beta_u)$):
6: Let $g_0 := \mathsf{H}_0(\vec{\boldsymbol{\rho}})$ and $g_1 := \mathsf{H}_1(\vec{\boldsymbol{\rho}})$
7: Let $e := \mathsf{H}_{\mathsf{FS}}(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}, A, c, z, g_0, g_1)$
8: if $g^{\beta_s} h^{\beta_r} v^{\beta_u} = X_{\mathsf{pk}} \cdot \mathsf{pk}^e \wedge g^{\beta_a} g_0^{\beta_r} g_1^{\beta_u} = X_A \cdot A^{e/L} \wedge \beta_a + c\beta_s = X_z + ez/L$ :
9:     return 1
10: return 0
```

Fig. 9: Σ-protocol for computing and verifying the correctness proof for partial signatures we use in the Detect algorithm.

$\mathcal{S}$ then programs H_{FS} as $\mathsf{H}_{\mathsf{FS}}(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}, A, c, z, g_0, g_1) = e$ and outputs $\pi = (e, \beta_a, \beta_s, \beta_r, \beta_u)$ as the proof. Clearly, the simulated transcript is identically distributed to the real-protocol transcript. We will prove the soundness in the concurrent setting directly when we prove the IA property of Glacius in §C.2.

C.2 Identifiable Abort

Recall from §2, for identifiable abort, the Detect algorithm must identify at least one misbehaving signer and must never blame honest signers. Honest signers never equivocate. Therefore, due to the security of DS no honest signer will be added to $\mathcal{J}$. Now, note that z_i is a deterministic function of the a_i, and all the

Input: Generators $(g, h, v) \in \mathbb{G}^3$, secret key polynomials $s(\cdot), r(\cdot), u(\cdot)$, randomness for random oracle programming $\{h_1, h_2, \ldots, h_q\} \leftarrow\$ \mathbb{Z}_p$.

KGen **simulation**

1. Use (g, h, v) as generators and $s(\cdot), r(\cdot), u(\cdot)$ as the secret key polynomials.

Corruption simulation:

2. When $\mathcal{A}$ corrupts a signer $i \in \mathcal{H}$ if $|\mathcal{C}| < t$:
   (a) Update $\mathcal{H} := \mathcal{H} \setminus \{i\}$ and $\mathcal{C} := \mathcal{C} \cup \{i\}$.
   (b) Faithfully reveals the internal state of signer i to $\mathcal{A}$.

Simulating random oracle queries: For each query to $\mathsf{H_{FS}}$ on some input x, use the next unused random value from the input $\{h_1, h_2, \ldots, h_q\}$ to program $\mathsf{H_{FS}}$.

Simulating signing protocol for any signing session

3. Follow the honest protocol for all honest signers. Simultaneously, also check for the Neq as follows.
4. Let A_j and z_j be the Sig_4 and Sig_5 messages, respectively, sent by signer $j \in \mathcal{C} \cap \mathsf{SS}$ during a signing session with session-id sid with signer set SS. Let $\hat{A}$ be the combined nonce for that session and $c = \mathsf{H_{sig}}(\hat{A}, \mathsf{pk}, m)$. Let $\pi_j = (X_{\mathsf{pk}}, X_A, X_z, \beta_a, \beta_s, \beta_r, \beta_u)$ be the proof output by signer j that successfully verifies. Compute Z_j as:
$$Z_j := A_j \cdot \mathsf{H}_0(\vec{\boldsymbol{\rho}})^{-r(j)\cdot L_{j,\mathsf{SS}}} \cdot \mathsf{H}_1(\vec{\boldsymbol{\rho}})^{-u(j)\cdot L_{j,\mathsf{SS}}} \tag{17}$$
   Here, $\vec{\boldsymbol{\rho}}$ is the Sig_1 message of the corresponding signing session.
   Then, if $Z_j \neq g^{z_j - c\cdot s(j)\cdot L_{j,\mathsf{SS}}}$, find the index $\mathsf{idx} \in [q]$ where $\mathcal{B}$ programmed $\mathsf{H_{FS}}$ on input $(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}_j, c, z_j, g_0, g_1)$ with h_{idx}. **return** (idx, j, π_j).
5. If the event Neq does not occur during the interaction with $\mathcal{A}$, **return** $(0, \varepsilon)$.

Fig. 10: Description of Algorithm $\mathcal{B}$ that simulates $\mathsf{Glacius}$ to an adversary $\mathcal{A}$.

messages signer i receives. Therefore, by the perfect completeness property of the Σ-protocol, π_i will always verify. Hence, signer i will not be added to $\mathcal{J}$.

Next, we argue that if an honest signer aborts (i.e., outputs $\perp$), then the Detect algorithm will identify at least one malicious signer. For simplicity, let us assume that $\mathcal{A}$ can not break the collision resistance property of $\mathsf{H_{view}}$. From the correctness property §4.1, when all signers send correct messages during a signing session, no honest signer will output $\perp$ during that session. Therefore, since signer i initiated the Detect protocol, then at least one signer in SS must have misbehaved. Clearly, if any malicious signer $j \in \mathsf{SS}$ equivocates or sends a proof π_j that does not verify, then j will be added to $\mathcal{J}$, and we are done. The interesting case is when π_j successfully verifies, but z_j is computed incorrectly. In Lemma 5, argue that assuming the hardness of discrete logarithm in $\mathbb{G}$, computing such a proof π_j for invalid z_j is infeasible. Therefore, if π_j successfully verifies for any $j \in \mathsf{SS}$, then z_j is valid; hence, due to the correctness property of $\mathsf{Glacius}$ implies that honest signer i will not output $\perp$. Thus, we get a contradiction.

Lemma 5. *Let* Neq *be the event that* $\mathcal{A}$ *outputs an incorrect* z_j *for any* $j \in \mathsf{SS}$ *with an proof* π_j *that verifies. Then, assuming the hardness of discrete logarithm (DL) in* $\mathbb{G}$, $\Pr[\mathsf{Neq}]$ *is negligible.*

Proof. To prove this lemma, we will rely on the generalized forking lemma [PS96, BN06]. More specifically, given an adversary $\mathcal{A}$ that can cause the event Neq to happen, we will build a "wrapping" algorithm $\mathcal{B}$ (see Figure 10) which runs $\mathcal{A}$ and returns information regarding the bad event Neq. Algorithm $\mathcal{B}$ simulates all the random oracles with uniformly random outputs. We then use $\mathcal{B}$ to construct an algorithm $\mathcal{B}_{\mathsf{dl}}$ (see Figure 11) that first runs the forking algorithm $\mathsf{Fork}^{\mathcal{B}}$ (identical to Figure 5) which forks $\mathcal{B}$ with respect to $\mathsf{H_{FS}}$ query. Algorithm $\mathcal{B}_{\mathsf{dl}}$ then uses the output of the $\mathsf{Fork}^{\mathcal{B}}$ algorithm to solve for discrete logarithm in $\mathbb{G}$.

Description of Algorithm $\mathcal{B}$ (Figure 10). $\mathcal{B}$ takes as input the generators (g, h, v), the signing keys $(s(\cdot), r(\cdot), u(\cdot))$, and a vector $\{h_1, h_2, \ldots, h_{q_{\mathsf{FS}}}\}$ of uniformly random field elements. $\mathcal{B}$ then interacts with $\mathcal{A}$ with these inputs. During this interaction, $\mathcal{B}$ uses $\{h_1, h_2, \ldots, h_{q_{\mathsf{FS}}}\}$ to program the random oracle $\mathsf{H_{FS}}$. Simultaneously, $\mathcal{B}$ also locally checks for the event Neq (step 5 in Figure 10).

When the event Neq occurs, $\mathcal{B}$ identifies the $\mathsf{H_{FS}}$ query associated with the event Neq. Let j be the signer associated with the event Neq. Also, let $\pi_j = (X_{\mathsf{pk}}, X_A, X_z, \beta_a, \beta_s, \beta_r, \beta_u)$ and $(\mathsf{pk}_j, A_j, (c, z_j), g_0, g_1)$

```
Algorithm $\mathcal{B}_{\mathsf{dl}}(\mathbb{G}, p, g, y)$:
1: Sample $\alpha \leftarrow\$ \mathbb{Z}_p^*$ and $\theta \leftarrow\$ \{0,1\}$
2: if $\theta = 0 : (h, v) := (y, g^\alpha)$; otherwise $(h, v) := (g^\alpha, y)$
3: Sample the signing keys polynomials $s(\cdot), r(\cdot), u(\cdot)$ as per the protocol specification.
4: Let $(\mathsf{val}, \mathsf{out}, \mathsf{out}') \leftarrow \mathsf{Fork}^{\mathcal{B}}(g, h, v, s(\cdot), r(\cdot), u(\cdot))$
5: if $\mathsf{val} = 0$ : return $\perp$
6: parse $(e, (j, \beta_s, \beta_r, \beta_u)) := \mathsf{out}$ and $(e', (j', \beta'_s, \beta'_r, \beta'_u)) := \mathsf{out}'$.
7: Compute $s_j, r_j$ and $u_j$ as:
       $$s_j := \frac{\beta_s - \beta'_s}{e - e'}; \qquad r_j := \frac{\beta_r - \beta'_r}{e - e'}; \qquad u_j := \frac{\beta_u - \beta'_u}{e - e'} \qquad (18)$$
8: Let $\delta_s := s(j) - s_j$, $\delta_r := r(j) - r_j$, and $\delta_u := u(j) - u_j$.
   // Let $h = g^{\alpha_h}$ and $v = g^{\alpha_v}$
9: if $\theta = 0 \wedge \delta_r \neq 0$ :
10:     return $(-\delta_s - \alpha_v \delta_u) \cdot \delta_r^{-1}$ as the DL solution
11: else if $\theta = 1 \wedge \delta_u \neq 0$ :
12:     return $(-\delta_s - \alpha_h \delta_r) \cdot \delta_u^{-1}$ as the DL solution
13: return $\perp$
```

Fig. 11: Description of Algorithm $\mathcal{B}_{\mathsf{dl}}$ solves discrete logarithm in $\mathbb{G}$

be the NIZK proof and the statement associated with the event Neq. Then, $\mathcal{B}$ finds the index idx such that $\mathcal{B}$ programmed the H_{FS} query on input $(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}_j, c, z_j, g_0, g_1)$ with h_{idx}, and returns the tuple $(\mathsf{idx}, (j, \beta_a, \beta_s, \beta_r, \beta_u))$ as its output.

Description of Algorithm $\mathsf{Fork}^{\mathcal{B}}$. The $\mathsf{Fork}^{\mathcal{B}}$ algorithm $\mathcal{B}_{\mathsf{dl}}$ runs is identical to Figure 5.

Description of Algorithm $\mathcal{B}_{\mathsf{dl}}$ (Figure 11). $\mathcal{B}_{\mathsf{dl}}$ takes as input $(\mathbb{G}, p, g, y)$: the description of the group $\mathbb{G}$ of order p, a generator g and a uniformly random element $y \in \mathbb{G}$. $\mathcal{B}_{\mathsf{dl}}$ then samples uniformly random $\alpha \leftarrow\$ \mathbb{Z}_p$ and a bit $\theta \leftarrow \{0,1\}$. Next, depending upon the value of θ, $\mathcal{B}_{\mathsf{dl}}$ sets the public parameters (g, h, v) in two different manner. More precisely, if $\theta = 0$, $\mathcal{B}_{\mathsf{dl}}$ sets $(g, h, v) := (g, y, g^\alpha)$, otherwise $\mathcal{B}_{\mathsf{dl}}$ sets $(g, h, v) := (g, g^\alpha, y)$.

Next, $\mathcal{B}_{\mathsf{dl}}$ honestly samples the signing key polynomials $s(\cdot), r(\cdot), u(\cdot)$, and runs $\mathsf{Fork}^{\mathcal{B}}$ with $(g, h, v, s(\cdot), r(\cdot), u(\cdot))$ as input. Let $(\mathsf{val}, \mathsf{out}, \mathsf{out}')$ be the output of $\mathsf{Fork}^{\mathcal{B}}$. If $\mathsf{val} = 0$, $\mathcal{B}_{\mathsf{dl}}$ returns $\perp$. Otherwise, let $\mathsf{out} = (e, (j, \beta_s, \beta_r, \beta_u))$ and $\mathsf{out}' = (e', (j', \beta'_s, \beta'_r, \beta'_u))$. $\mathcal{B}_{\mathsf{dl}}$ computes (s_j, r_j, u_j) as:

$$s_j := \frac{\beta_s - \beta'_s}{e - e'}; \qquad r_j := \frac{\beta_r - \beta'_r}{e - e'}; \qquad u_j := \frac{\beta_u - \beta'_u}{e - e'} \tag{19}$$

Let $\delta_s := s(j) - s_j$, $\delta_r := r(j) - r_j$, and $\delta_u := u(j) - u_j$. Also, let $h = g^{\alpha_h}$ and $v = g^{\alpha_v}$ for some $\alpha_h, \alpha_v \in \mathbb{Z}_p^*$. Then, if $\theta = 0$ and $\delta_r \neq 0$, $\mathcal{B}_{\mathsf{dl}}$ outputs $(-\delta_s - \alpha_v \delta_u) \cdot \delta_u^{-1}$ as the DL solution. Alternatively, if $\theta = 1$ and $\delta_u \neq 0$, $\mathcal{B}_{\mathsf{dl}}$ outputs $(-\delta_s - \alpha_h \delta_r) \cdot \delta_u^{-1}$ as the DL solution.

Analysis of $\mathcal{B}_{\mathsf{dl}}$. Let ε be the probability that $\mathsf{Fork}^{\mathcal{B}}$ outputs $(1, \mathsf{out}, \mathsf{out}')$. Also, let $\varepsilon_{\mathsf{dl}}$ be the probability that $\mathcal{B}_{\mathsf{dl}}$ outputs the discrete logarithm of y. Next, we prove that $\varepsilon_{\mathsf{dl}} \geq \varepsilon/2$. Also, let $\varepsilon_{\mathsf{neq}}$ be the probability of the event Neq. Then, from the generalized forking lemma, we get that:

$$\varepsilon \geq \frac{\varepsilon_{\mathsf{neq}}^2}{q_{\mathsf{FS}}} - \frac{\varepsilon_{\mathsf{neq}}}{p}$$

Therefore, by combining all the above, we get:

$$\varepsilon_{\mathsf{dl}} \geq \frac{1}{2} \cdot \left(\frac{\varepsilon_{\mathsf{neq}}^2}{q_{\mathsf{FS}}} - \frac{\varepsilon_{\mathsf{neq}}}{p} \right) \implies \varepsilon_{\mathsf{neq}} \leq \frac{q_{\mathsf{FS}}}{p} + \sqrt{2 \cdot q_{\mathsf{FS}} \cdot \varepsilon_{\mathsf{dl}}} \tag{20}$$

Note that $\mathsf{Fork}^{\mathcal{B}}$ outputting $(1, \mathsf{out}, \mathsf{out}')$ implies that the event Neq happens for the first time during $\mathcal{B}$'s interaction with $\mathcal{A}$ for the idx-th H_{FS} query for both execution of $\mathcal{B}$. Since, $\mathcal{A}$'s view in both execution is identical until the idx-th query, it implies that the input to the idx-th $\mathsf{H_{FS}}$ is identical in both execution. Moreover, $\mathcal{A}$ outputs valid NIZK proof π_j and π'_j for the same statement in both the execution.

Let $(X_{\mathsf{pk}}, X_A, X_z, \mathsf{pk}_j, c, z_j, g_0, g_1)$ be the input of the idx-th $\mathsf{H_{FS}}$ query. Then, $\mathsf{out} := (e, (j, \beta_s, \beta_r, \beta_u))$ and $\mathsf{out}' = (e', (j', \beta'_s, \beta'_r, \beta'_u))$ satisfy that:

$$g^{\beta_s} h^{\beta_r} v^{\beta_u} = X_{\mathsf{pk}} \cdot \mathsf{pk}_j^e; \qquad \text{and} \qquad g^{\beta'_s} h^{\beta'_r} v^{\beta'_u} = X_{\mathsf{pk}} \cdot \mathsf{pk}_j^{e'}.$$

Therefore,

$$g^{s_j} h^{r_j} v^{u_j} = \mathsf{pk}_j = g^{s(j)} h^{r(j)} v^{u(j)}$$

here (s_j, r_j, u_j) are the values $\mathcal{B}_{\mathsf{dl}}$ computes in equation (19), and $(s(j), r(j), u(j))$ are the signing keys of party j as per the protocol specification. This implies that:

$$g^{s_j - s(j)} h^{r_j - r(j)} v^{u_j - u(j)} = 1_{\mathbb{G}} = g^{\delta_s} h^{\delta_r} v^{\delta_u} \tag{21}$$

Proving $(\delta_r, \delta_u) \neq (0,0)$ whenever the event Neq occurs. Next, we argue that $(\delta_r, \delta_u) \neq (0,0)$. For the sake of contradiction, assume that $(\delta_r, \delta_u) = (0,0)$. Also, let $\pi_j = (X_{\mathsf{pk}}, X_A, X_z, , \beta_a, \beta_s, \beta_r, \beta_u)$ and $\pi'_j = (X_{\mathsf{pk}}, X_A, X_z, \beta'_a, \beta'_s, \beta'_r, \beta'_u)$ be the NIZK proofs $\mathcal{A}$ outputs in its two execution, respectively. Then, for $(\beta_a, \beta_r, \beta_u)$ and $(\beta'_a, \beta'_r, \beta'_u)$ it holds that:

$$g^{\beta_a} \cdot g_0^{\beta_r} \cdot g_1^{\beta_u} = X_A \cdot A_j^{e/L_{j,\mathsf{SS}}} \qquad \text{and} \qquad g^{\beta'_a} \cdot g_0^{\beta'_r} \cdot g_1^{\beta'_u} = X_A \cdot A_j^{e'/L_{j,\mathsf{SS}}}$$
$$\beta_a + c\beta_s = X_z + \frac{e \cdot z}{L_{j,\mathsf{SS}}} \qquad \text{and} \qquad \beta'_a + c\beta'_s = X_z + \frac{e' \cdot z}{L_{j,\mathsf{SS}}}$$

Let $a' := L_{j,\mathsf{SS}} \cdot (\beta_a - \beta'_a)/(e - e')$. Then, from equation (18) we have:

$$g^{a'} \cdot g_0^{r_j \cdot L_{j,\mathsf{SS}}} \cdot g_1^{u_j \cdot L_{j,\mathsf{SS}}} = A_j \qquad \text{and} \qquad a' + c \cdot s' \cdot L_{j,\mathsf{SS}} = z \tag{22}$$

However, since by our assumption $(\delta_r, \delta_u) = 0$, we have $(r_j, u_j) = (r(j), u(j))$, and hence

$$g^{a'} \cdot g_0^{r(j)} \cdot g_1^{u(j)} = A_j \implies g^{a'} = A_j \cdot g_0^{-r(j) \cdot L_{j,\mathsf{SS}}} \cdot g_0^{-u(j) \cdot L_{j,\mathsf{SS}}} = Z_j \tag{23}$$

Now we have two cases to analyze: first, $\delta_s = s(j) - s' = 0$; and second, $\delta_s \neq 0$. Next, we argue that in both cases, we get a contradiction.

1. When $\delta_s = 0$, then from equation (22), we get that $a' = z - c \cdot s(j) \cdot L_{j,\mathsf{SS}}$, and hence $Z_j = g^{z - c \cdot s(j) \cdot L_{j,\mathsf{SS}}}$. This implies that the event Neq does not occur for the idx-th $\mathsf{H_{FS}}$ query. Hence, we get a contradiction.
2. When $\delta_s \neq 0$, then since $(\delta_r, \delta_u) = (0,0)$, then from equation (21), we have that $g^{\delta_s} = 1_{\mathbb{G}}$, which is a contradiction.

Computing the success probability. Say $h = g^{\alpha_h}$ and $v = g^{\alpha_v}$ for some $\alpha_h, \alpha_v \in \mathbb{Z}_p$. Then, equation (21), implies that $\delta_s + \delta_r \alpha_h + \delta_u \alpha_v = 0$. If either δ_r or δ_u is non-zero, then $\mathcal{B}_{\mathsf{dl}}$ computes α_h or α_v, respectively, as:

$$\delta_r \neq 0 \Rightarrow \alpha_h = (-\delta_s - \alpha_v \delta_u) \cdot \delta_r^{-1}; \qquad \delta_u \neq 0 \Rightarrow \alpha_v = (-\delta_s - \alpha_h \delta_r) \cdot \delta_u^{-1}$$

Now, note that $\mathcal{B}_{\mathsf{dl}}$ will be able to compute the discrete logarithm of y if either of the following happens: (i) $\theta = 0$ and $\delta_r \neq 0$; and (ii) $\theta = 0$ and $\delta_u \neq 0$. This implies that:

$$\varepsilon_{\mathsf{dl}} \geq \Pr[\theta = 0 \wedge \delta_r \neq 0] + \Pr[\theta = 1 \wedge \delta_u \neq 0] \tag{24}$$

Note that the view of $\mathsf{Fork}^{\mathcal{B}}$ is identically distributed for both $\theta = 0$ and $\theta = 1$, and hence the value of (δ_r, δ_u) is independent of θ. Therefore we get:

$$\begin{aligned} \varepsilon_{\mathsf{dl}} &\geq \Pr[\theta = 0] \cdot \Pr[\delta_r \neq 0] + \Pr[\theta = 1] \cdot \Pr[\delta_u \neq 0] \\ &= \frac{1}{2} \cdot (\Pr[\delta_r \neq 0] + \Pr[\delta_u \neq 0]) \geq \frac{1}{2} \cdot \Pr[\delta_r \neq 0 \vee \delta_u \neq 0] = \frac{1}{2} \cdot \varepsilon \end{aligned} \tag{25}$$

Now, combining equation (25) with equation (20), we get that:

$$\varepsilon_{\mathsf{neq}} \leq \frac{q_{\mathsf{FS}}}{p} + \sqrt{2 \cdot q_{\mathsf{FS}} \cdot \varepsilon_{\mathsf{dl}}}. \qquad \square$$