Tightly Secure Threshold Signatures over Pairing-Free Groups

Renas Bacho [1,2] Benedikt Wagner [3]

October 3, 2024

[1] CISPA Helmholtz Center for Information Security
renas.bacho@cispa.de
[2] Saarland University, Saarbrücken, Germany

[3] Ethereum Foundation
benedikt.wagner@ethereum.org

Abstract

Threshold signatures have been drawing lots of attention in recent years. Of particular interest are threshold signatures that are proven secure under adaptive corruptions (NIST Call 2023). Sadly, existing constructions with provable adaptive security suffer from at least one of the following drawbacks: (i) strong idealizations such as the algebraic group model (AGM), (ii) an unnatural restriction on the corruption threshold being $t/2$ where t is the signing threshold, or (iii) prohibitively large security loss under established assumptions. Notably, point (iii) has received little to no attention in the literature on this subject.

In this work, we introduce Twinkle-T, a new threshold signature scheme which overcomes these limitations. Twinkle-T is the first scheme to have a fully tight security proof under up to t adaptive corruptions without relying on the AGM. It also has a signing protocol consisting of only three rounds and thus matches the currently best threshold signature with full adaptive security Twinkle (Eurocrypt 2024) in the pairing-free discrete logarithm setting. We prove security from a standard non-interactive assumption, namely, the Decisional Diffie-Hellman (DDH) assumption.

Keywords: Threshold Signatures, Tightness, Adaptive Security, Pairing-Free Groups

Contents

1 Introduction **3**
1.1 Our Contribution . . . 4
1.2 More on Related Work . . . 5
1.3 Paper Organization . . . 5

2 Technical Overview **5**
2.1 Twinkle and its Security Proof . . . 6
2.2 Towards Tight Threshold Partitioning . . . 7
2.3 Reducing the Number of Rounds . . . 8

3 Preliminaries **10**
3.1 Threshold Signatures . . . 10
3.2 Tagged Linear Function Families . . . 12

4 Non-Interactive Arguments for Tagged Linear Functions **13**

5 Our Construction **15**
5.1 Construction . . . 15
5.2 Security Analysis . . . 16

6 Instantiations and Efficiency **24**

A More Details on Non-Interactive Argument Systems **32**
A.1 Formal Definitions . . . 32
A.2 Proofs for Our Construction . . . 35

B Details on Instantiations **38**
B.1 Instantiation from Algebraic One-More CDH . . . 38
B.2 Instantiation from DDH . . . 38

C Pseudocode **40**

1 Introduction

Multi-party variants of digital signatures have recently garnered a lot of attention due to their ability to distribute trust across multiple parties, a feature that proves beneficial in a range of modern applications, including cryptocurrencies. Arguably, one of the most prominent example of multi-party signatures are *threshold signatures* [Des88, DF90, Ped91]. In a $(t+1)$-out-of-n threshold signature scheme, the secret key sk for a public key pk is secret-shared among n signers, ensuring correctness and security:

- *Correctness.* Any subset of $t+1$ signers can collaborate to execute a signing protocol that produces a valid signature σ for a message m. This signature can then be verified against the public key pk and the message m.
- *Security.* No malicious coalition of t or fewer signers can generate a valid signature. This has to hold even if some of them have maliciously participated in signing interactions with honest parties before.

The growing interest in threshold signatures – and threshold cryptography more broadly – has even resulted in ongoing efforts towards standardization [BP22].

Progress in Pairing-Free Groups. Motivated by the appealing features of pairing-free cyclic groups [TZ22, TZ23], recent research has focused on constructing threshold signature schemes in such groups [KG20, BCK$^+$22, TZ23, CKM23b, Lin24, BLT$^+$24, KRT24]. In our work, we also focus on pairing-free constructions, in the random oracle model (ROM). Notably, in this setting, there has been significant progress in terms of security guarantees:

- *Non-Interactive Assumptions.* While early constructions had to rely on interactive assumptions for their security proof, Tessaro and Zhu [TZ23] pioneered a shift towards more conservative, non-interactive assumptions.
- *Adaptive Security.* Early schemes are analyzed only under static corruptions, where the adversary must corrupt parties before learning the public key. More recent developments have tackled the stronger, more realistic scenario where the adversary can adaptively corrupt parties at any time. Namely, Crites et al. [CKM23b] showed a restricted form of adaptive security for their scheme, where the adversary is only allowed to corrupt $t/2$ parties, under an interactive assumption. After that, Bacho et al. [BLT$^+$24] introduced the first fully adaptive (up to t corruptions) scheme in the pairing-free setting, called Twinkle. In contrast to the scheme by Crites et al., this scheme is even proven secure under a non-interactive assumption.

It is worth saying that these security improvements have been achieved with minimal impact on communication complexity and round efficiency, while also avoiding idealized models like the algebraic group model (AGM) [FKL18].

Concrete Security. In this work, we observe that an important dimension in terms of security guarantees has not been considered: namely, the concrete security level guaranteed by the security proof. Most prior schemes [KG20, CKM21, BTZ22, TZ23, CKM23b, Lin24, KRT24] rely on the Forking Lemma [BN06, BDL19], a technique known to yield loose security bounds. Concretely, for any adversary $\mathcal{A}$ that breaks security of the scheme with probability ϵ and makes at most Q random oracle and signing queries, there is a reduction that runs in approximately the same time and breaks the underlying assumption with probability ϵ'. The relation between ϵ, Q, and ϵ' is[1]

$$\epsilon \leq \sqrt{\Theta(Q) \cdot \epsilon'}.$$

In other words, if we assume that the underlying assumption is κ-bit secure ($\epsilon' \leq 2^{-\kappa}$), then the scheme only guarantees a $(\kappa - \log Q)/2$-bit security level. In practice, if the scheme is instantiated with a group providing 128-bit security, the resulting security level would be only 44 bits for $Q \approx 2^{40}$. Bacho et al. [BLT$^+$24] circumvent the use of the Forking Lemma in their construction of Twinkle, allowing them to

[1]In this overview, we ignore additional statistical terms that are independent of the underlying assumption when giving security bounds. We also ignore the running times of reductions and the fact that many of these schemes rely on less standard interactive assumptions [KG20, BCK$^+$22, CKM23b]. Further, the recent schemes Sparkle+ [CKM23a] and KRT [KRT24] also include an additive term related loosely to the security of a standard signature scheme.

achieve full adaptive security. As a side effect, this also leads to an improvement in concrete security. Specifically, the security bound is

$$\epsilon \leq \Theta(Q) \cdot \epsilon'.$$

While this is a clear improvement, it still leads to weak concrete security guarantees. Specifically, assuming 128-bit security for DDH and $Q \approx 2^{40}$, we are left with only $128 - 40 = 88$ bits of security for Twinkle. A tightly secure scheme, i.e., a scheme with a bound $\epsilon \leq O(1) \cdot \epsilon'$ for a small constant, with the same qualitative guarantees as Twinkle remains an open problem.

1.1 Our Contribution

We solve this open problem by presenting Twinkle-T, the first tightly secure threshold signature scheme. We only rely on the random oracle model and not on any other idealizations such as generic or algebraic group models. Our scheme achieves the following:

- *Full Adaptivity.* Twinkle-T is secure in the presence of an adversary that makes up to $t < n$ adaptive corruptions.
- *Non-Interactive Assumptions.* Twinkle-T can be instantiated using the same assumptions as Twinkle, for instance, based on DDH over pairing-free prime order groups.
- *Tight Security.* In contrast to Twinkle, we can give a tight security proof for Twinkle-T.

Concretely, while Twinkle guarantees a security level of at most 88 bits over a 128-bit secure group, our scheme is 127-bit secure. At the same time, we emphasize that the efficiency of our schemes stay within a practical regime, and we achieve this without increasing the number of rounds compared to Twinkle [BLT+24]. That is, our signing protocol consists of only three rounds. Consequently, we view our result as a *strict improvement* over Twinkle. We also emphasize that even in the pairing-friendly setting or the statically secure setting, we are not aware of any tightly secure construction from a standard assumption. For a comparison of threshold signature schemes in pairing-free cyclic groups, see Table 1.

Scheme	Rounds	Adaptive	Assumption	Security Loss
Frost [KG20, BCK+22]	2	✗	AOMDL	$\Theta(Q/\epsilon)$
Frost2 [CKM21, BTZ22, BCK+22]	2	✗	AOMDL	$\Theta(Q/\epsilon)$
Frost3 [RRJ+22, CGRS23]	2	✗	AOMDL	$\Theta(Q/\epsilon)$
TZ [TZ23]	2	✗	DLOG	$\Theta(Q/\epsilon)$
Lindell [Lin24]	3	✗	DLOG	$\Theta(Q/\epsilon)$
Classic-S [Mak22]	3	✗	DLOG	$\Theta(Q/\epsilon)$
Sparkle [CKM23b]	3	✓	AOMDL	$\Theta(Q/\epsilon)$
Sparkle+ [CKM23a]	3	✓	AOMDL	$\Theta(Q/\epsilon)$
Zero-S [Mak22]	3	✓	DLOG	$\Theta(Q/\epsilon)$
Twinkle$_{\mathsf{AOMCDH}}$ [BLT+24]	3	✓	AOMCDH	$\Theta(Q)$
Twinkle$_{\mathsf{DDH}}$ [BLT+24]	3	✓	DDH	$\Theta(Q)$
KRT [KRT24]	5	✓	DLOG	$\Theta(Q^3/\epsilon)$
Twinkle-T$_{\mathsf{AOMCDH}}$ (ours)	3	✓	AOMCDH	$\Theta(1)$
Twinkle-T$_{\mathsf{DDH}}$ (ours)	3	✓	DDH	$\Theta(1)$

Table 1: Comparison of threshold signature schemes in pairing-free cyclic groups. We assume a trusted dealer distributing key shares securely. We do not consider proofs in the algebraic group model (AGM). We do not consider schemes assuming a broadcast channel [CGJ+99, SS01, AF04, GJKR07, BHK+24, GS24]. We denote the total number of random oracle and signing queries by Q, and the advantage of an adversary against the scheme by ϵ. In [BCK+22], the authors show that Frost achieves the stronger security notion of TS-SUF-3 with a security loss of $\Theta(n^2Q^2/\epsilon)$. The same holds for TZ [TZ23]. Further, Sparkle+ and KRT also include an additive term in their security loss related loosely to the security of a standard signature scheme. For simplicity, we omit these terms, which is in favor of them. Lindell, Classic-S, and Zero-S UC-realize an ideal functionality for computing Schnorr signatures, which are known to have an inherent security loss of $\Omega(Q/\epsilon)$ [Seu12].

1.2 More on Related Work

As a starting point for further reading, we give references to relevant related work, particularly, on adaptively secure threshold signatures, other multi-signature variants, and the study of tight security.

Adaptively Secure Threshold Signatures. Initial works [CGJ+99, FMY99, JL00, LP01, AF04, ADN06, WQL09] on threshold cryptography use the *single inconsistent player* (SIP) technique to design adaptively secure threshold signatures. These approaches, however, require a lot of interaction rounds. Notably, none of these constructions has a tight security proof. More critically, existing impossibility results for their underlying signature schemes [Cor00, HJK12, Seu12, KK12] rule out any tight security proofs. Recent works [LJY14, BL22, DR24] establish adaptive security for BLS-type threshold signatures. Among these, only [BL22] gives a tight security proof, but relies on the AGM and the OMDL assumption. Threshold Schnorr signatures have recently gained a lot of attention and several adaptively secure schemes have been proposed [Mak22, CKM23a, CKM23b, KRT24, BLSW24]. Among these, only [CKM23b, BLSW24] give a tight security proof, again by relying on the AGM and the AOMDL assumption. Importantly, existing impossibility results for Schnorr-type signatures rule out tight security proofs in the ROM [Seu12] from standard assumptions. Adaptively secure threshold ECDSA signatures are given in [CGG+20], but only for the case $t = n - 1$.

Other Multi-Party Signatures. Multi-signatures are a form of threshold signatures where $t = n - 1$, meaning all n parties need to participate in the signing protocol, and each party can independently generate its keys. This allows to avoid the need of a distributed key generation (DKG) protocol, as used in threshold signatures. Since their introduction [IN83], a large amount of works have emerged [MOR01, Bol03, BN06, LOS+06, BCJ08, BJ10], and especially in recent years [BDN18, MPSW19, DEF+19, NRSW20, NRS21, AB21, BD21, BTT22, FSZ22, TZ23]. Further, some works also focus on tightness [BN06, RY07, BNN07, BJ08, QLH12, FH20, PW23, PW24, BW24]. Since there is no joint threshold public key pk in multi-signatures and generally adaptive security is not considered, other techniques are needed to construct tightly secure threshold signatures. Existing approaches to obtain tightness in multi-signatures are via lossy identification [AFLT12], requiring more than one key per signer [PW23], and the Katz-Wang pseudorandom bit sampling [KW03]. Crucially, none of these techniques translates to the threshold setting with adaptive corruptions (cf. Section 2). There has recently also been an increase in works on lattice-based threshold signatures [GKS24, dPKM+24, EKT24, CATZ24]. Blind threshold and multi-signatures have also appeared very recently [CKM+23c, KRB+24].

Tightness in General. Tightness has also been considered for other types of signatures, e.g., structure-preserving signatures [HJ12, GHKP18, CKP+23, AHN+23], lattice- and isogeny-based signatures [EKP20, PW22], and identity-based signatures and encryption [HKS15, HJP18, KYY18, PW21, HKK+24].

1.3 Paper Organization

We structure this paper as follows. In Section 2, we give an informal technical overview of our results. Then, in Section 3, we introduce our notation, the definition of threshold signatures, and the definition of tagged linear function families following [BLT+24]. For our construction, we will use a novel efficient construction of a non-interactive argument system related to these tagged linear function families. This construction is the focus of Section 4. Then, in Section 5, we present our threshold signature construction generically from any tagged linear function family. In Section 6, we instantiate the tagged linear function families as in [BLT+24] and discuss the efficiency of the resulting constructions.

2 Technical Overview

In this section, we explain the challenges we face and the ideas we use to overcome them. For simplicity, we only consider an instantiation based on a one-more variant of CDH over a cyclic group $\mathbb{G}$ with generator g of prime order p. In the main body, we abstractly present the construction using tagged linear functions [BLT+24], which can be instantiated from DDH.

2.1 Twinkle and its Security Proof

Recall that $\mathsf{Twinkle}$ [BLT+24] is the first $(t+1)$-out-of-n threshold signature to achieve fully adaptive security in the random oracle model. Technically, this is accomplished by avoiding rewinding, which typically results in highly non-tight security bounds. As such, $\mathsf{Twinkle}$ serves as a natural starting point for our solution. While we encourage readers to refer to the technical overview in [BLT+24] for a deeper understanding, we will focus here on explaining the aspects of $\mathsf{Twinkle}$ relevant to our construction.

Twinkle Signatures. Consider a public key $\mathsf{pk} = g^{\mathsf{sk}}$ and public key shares $\mathsf{pk}_i = g^{\mathsf{sk}_i}$, where $\mathsf{sk}_i = f(i)$ and $\mathsf{sk} = f(0)$ for a polynomial f of degree t. A $\mathsf{Twinkle}$ signature for a message m has two components:

- The element $\mathsf{pk}^{(2)} = h^{\mathsf{sk}}$, where $h = \mathsf{H}(\mathsf{m})$ for random oracle H. This is computed by $t+1$ signers revealing $\mathsf{pk}_i^{(2)} = h^{\mathsf{sk}_i}$. Intuitively, no t colluding signers can compute h^{sk}.
- A proof that $\mathsf{pk}^{(2)} = h^{\mathsf{sk}}$, which is derived from a Σ-protocol with transcript $((R^{(1)}, R^{(2)}), c, s)$, computed interactively by $t+1$ signers in three rounds.

To securely implement this, the computation of these two parts must be interleaved as follows:

1. *Round 1.* Each signer i computes $\mathsf{pk}_i^{(2)}$, samples a random $r_i \in \mathbb{Z}_p$, and defines nonces $R_i^{(1)} = g^{r_i}$, $R_i^{(2)} = h^{r_i}$. They send a commitment $\mathsf{com}_i = \tilde{\mathsf{H}}(R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$ for a random oracle $\tilde{\mathsf{H}}$. Intuitively, this round ensures that parties choose independent nonces.
2. *Round 2.* Signers open their commitments by revealing $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$. If any opening fails, the protocol aborts. Otherwise, $\mathsf{pk}^{(2)}$ is computed as above, and $R^{(1)}$ (resp. $R^{(2)}$) as the product of all $R_i^{(1)}$ (resp. $R_i^{(2)}$).
3. *Round 3.* The challenge $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$ is derived from a random oracle $\bar{\mathsf{H}}$. Each signer computes a response s_i based on c and their (sk_i, r_i). These responses are distributed and combined into a final response s. The signature consists of $\mathsf{pk}^{(2)}$ and (c, s). Intuitively (c, s) is a sound proof that $\mathsf{pk}^{(2)} = h^{\mathsf{sk}}$.

We refer to variables $R_i^{(1)}, R^{(1)}$ as the *g-side* and $\mathsf{pk}_i^{(2)}, \mathsf{pk}^{(2)}, R_i^{(2)}, R^{(2)}$ as the *h-side* of the protocol. In the following, we explain the source of the security loss of $\mathsf{Twinkle}$ mentioned in the introduction.

Security without Signing. We begin by assuming the adversary does not make any signing queries. The adversary is given the public key pk and all public key shares pk_i, then corrupts up to t signers, learning their secret keys sk_i. It eventually outputs a signature $(\mathsf{pk}^{(2)}, (c, s))$ for a message m^* and wins if the signature is valid. Intuitively, we want to use such an adversary to break CDH. Consider a reduction that takes a CDH challenge $(g, X, Y) \in \mathbb{G}^3$ and defines $\mathsf{pk} := X$. The reduction sets up appropriate public key shares pk_i and runs the adversary, simulating the random oracle by defining $\mathsf{H}(\mathsf{m}) := Y^{\delta_{\mathsf{m}}}$ for random δ_{m}. The adversary's forgery contains $\mathsf{pk}^{(2)}$ and a proof that $\mathsf{pk}^{(2)} = \mathsf{H}(\mathsf{m}^*)^{\mathsf{sk}} = Y^{\delta_{\mathsf{m}^*}}$, from which the reduction computes the CDH solution Y^{sk} using δ_{m^*}. However, handling adaptive corruptions is a challenge: when setting up the public key shares, the reduction can only choose up to t signers for which it knows their secret key shares. Since corruptions are adaptive, the adversary may corrupt signers the reduction has not chosen. To manage this, $\mathsf{Twinkle}$ employs a one-more variant[2] of CDH. In this variant, the reduction receives $(g, X = \mathsf{pk}, Y)$ and all valid public key shares pk_i[3]. It also gets t-time oracle access to an oracle that outputs sk_i on input i. This oracle allows the reduction to handle adaptive corruptions. Importantly, the reduction must avoid rewinding to ensure it makes only t oracle queries. Otherwise, an adaptive adversary may corrupt more than t distinct parties over its two executions. For more details see [BLT+24].

Simulating Signing Queries – Non-Tightly. Based on what we have discussed so far, the reduction for $\mathsf{Twinkle}$ would be tight. The security loss arises when the reduction needs to simulate honest signers during signing interactions. This process involves two key steps:

[2]The assumption we sketch here is slightly simplified. Also, we point out again that by phrasing the scheme abstractly and replacing $(g, x) \mapsto g^x$ with a suitable function, the interactive assumption can be avoided.

[3]To recall, valid public key shares here are exponentiated evaluations of a degree-t polynomial f with $X = g^{f(0)}$.

- *Simulating the g-side.* The reduction can simulate the g-side and the responses s_i using random oracles on equivalence classes – a sophisticated technique introduced in [BLT+24] to avoid using a broadcast channel. We will come back to this later.
- *Simulating the h-side.* The reduction translates the g-side into the h-side, i.e., computes $(\mathsf{pk}_i^{(2)}, R_i^{(2)})$ from $(\mathsf{pk}_i, R_i^{(1)})$. This introduces a security loss proportional to the number of signing queries, Q.

For now, let us focus on simulating the h-side given the g-side. The reduction *partitions* the message space into two subspaces:

- *The g-space.* For messages in this space, $\mathsf{H}(\mathsf{m}) = g^{\delta_\mathsf{m}}$. The reduction can simulate the h-side given the g-side and δ_m, e.g., $\mathsf{pk}_i^{(2)} = \mathsf{pk}_i^{\delta_\mathsf{m}}$.
- *The Y-space.* For these messages, $\mathsf{H}(\mathsf{m}) = Y^{\delta_\mathsf{m}}$. The reduction uses a forgery on such messages to solve one-more CDH as explained above.

The reduction succeeds if all signing queries fall within the g-space, and the forgery is in the Y-space. This happens with probability $1/Q$ using well-known partitioning techniques.

2.2 Towards Tight Threshold Partitioning

Thus far, we have outlined the design and analysis of $\mathsf{Twinkle}$ [BLT+24] and discussed the reasons behind its security loss. We will now shift our focus to our solution, explaining how we can achieve tight security. For now, we will not focus on preserving the round complexity of $\mathsf{Twinkle}$.

Pseudorandom Bits. In standard single-signer signatures, a prominent technique to enable tight partitioning is Katz and Wang's pseudorandom bit approach [KW03]. Here, instead of signing m, the signer signs the pair $(\mathsf{m}, b_\mathsf{m})$, where $b_\mathsf{m} \in \{0, 1\}$ is pseudorandomly derived from the message and included in the signature. Using our terminology from above, this minimal modification allows tight partitioning: for each message m, the reduction places $(\mathsf{m}, b_\mathsf{m})$ in the g-space and $(\mathsf{m}, 1 - b_\mathsf{m})$ in the Y-space. With this, the reduction can simulate signing queries for every message. On the other hand, as the bit b_{m^*} remains pseudorandom for the forgery message m^*, the adversary forges a signature for $(\mathsf{m}^*, 1 - b_{\mathsf{m}^*})$ with probability $1/2$. In this case, the reduction can break the underlying assumption.

... and Pseudorandom Paths. Unfortunately, implementing the pseudorandom bit approach in a multi-signer setting is highly challenging, as noted in recent works [PW23, PW24] on multi-signatures. Namely, since each signer selects the pseudorandom bit b_m independently, they would generate different bits, resulting in different $h = \mathsf{H}(\mathsf{m}, b_\mathsf{m})$ values, breaking the scheme's correctness. To address this, Pan and Wagner [PW23, PW24] have introduced the pseudorandom paths technique. In this method, each signer holds two public keys and pseudorandomly choses which one to use for a given message. Critically, the technique exploits that signers have independent keys, and results in exponentially many potential key combinations. Therefore, it is not at all clear how to apply this technique to threshold signatures, where a single public key represents all signers. In fact, how to use their approach to get a tightly secure multi-signature scheme with key aggregation is still an open problem.

Let's Rewind a Bit: Ad Hoc Partitioning. It appears that the Katz-Wang technique is not well suited for our setting. Returning to the single-signer setting, an earlier approach by Bellare and Rogaway [BR96] extends the message not with a pseudorandom bit, but with a λ-bit[4] random salt ρ. In this approach, partitioning is done in a more *ad hoc* manner: for each query $\mathsf{H}(\mathsf{m}, \rho)$ on a fresh pair (m, ρ), the pair is generally placed in the Y-space. However, when signing, the reduction selects a random $\rho \in \{0, 1\}^\lambda$, assigns (m, ρ) to the g-space, and proceeds with the signature. The high entropy of ρ ensures that this pair has not been placed in the Y-space before. As the forgery message m^* has never been signed, it can only be in the Y-space and the reduction succeeds. From the single-signer perspective, it is not clear why ad hoc partitioning should be any better than the Katz-Wang technique. Clearly, it results in larger signatures, and in fact, Katz and Wang's method was intended as an improvement. Surprisingly, however, we find that the Bellare-Rogaway technique is far more suitable for the multi-signer setting.

Ad Hoc Partitioning in a Distributed Setting. To implement ad hoc partitioning in the multi-signer setting, we must ensure that each signer contributes entropy to the selection of the message-dependent

[4]Throughout, λ denotes the security parameter.

group element h. In particular, no malicious signer should be able to predict or bias the choice of h. The first prototype of our scheme proceeds as follows:

1. Each participating signer i samples a random $\rho_i \in \{0,1\}^\lambda$ and sends ρ_i to the other signers.
2. The signers then run $\mathsf{Twinkle}$, using $(\mathsf{m}, (\rho_i)_i)$ as the message, which means that $h = \mathsf{H}(\mathsf{m}, (\rho_i)_i)$.

Although an adversary can choose its ρ_i based on the honest signers' values, the tight ad hoc partitioning remains intact. Namely, when an honest signer j starts participating in the protocol for a message m, the reduction samples a random ρ_j. From that moment on, the reduction places every $(\mathsf{m}, (\rho_i)_i)$ that is queried and contains ρ_j at the appropriate position into the g-space. Then, it sends ρ_i to the adversary, and simulates the remainder of the protocol as in $\mathsf{Twinkle}$. Every other input to the random oracle is placed into the Y-space. The entropy of this single ρ_j makes the argument from above go through.

Compressing the Signature. Our prototype has a major issue: verifying the signature requires knowing *all* the ρ_i's, which would require including them in the signature, leading to a signature size of at least $(t+1)\lambda$ bits. This is far from ideal, and almost as large as a trivial threshold signature[5]. To address this, we observe that the unbiasedness of the individual ρ_i's is only needed by the reduction during signing. In particular, the ad hoc partitioning ensures that every message that has not been signed is in the Y-space, so the individual ρ_i's are not necessary for tightness when it comes to the forgery. With this in mind, we can compress the ρ_i's into a short $\rho = \hat{\mathsf{H}}(\mathsf{m}, (\rho_i)_i)$, run $\mathsf{Twinkle}$ on message (m, ρ), and include only ρ in the signature. The proof requires an additional step to ensure that ρ is not used before the ρ_i's are revealed, but the tight security still holds. As an intermediate result, we achieve a tightly secure protocol with four rounds.

2.3 Reducing the Number of Rounds

Our last goal is to reduce the number of rounds to match the round complexity of $\mathsf{Twinkle}$. As it turns out, the primary technical challenge lies in simulating the g-side, rather than the h-side. To tackle this, we must modify the inner workings of $\mathsf{Twinkle}$ and make adjustments to $\mathsf{Twinkle}$'s proof technique.

Eliminating The Commitment Round. A promising approach to reducing the number of rounds is by eliminating the first round of $\mathsf{Twinkle}$, where signers send their commitments com_i. However, doing so without further modifications would allow malicious signers to bias the combined nonce $R^{(1)}$, leading to insecure schemes [DEF+19, NRS21]. Interestingly, recent works on multi-signatures have securely removed such an additional commitment round by utilizing homomorphic trapdoor commitments [DOTT21, PW23, PW24]. Unfortunately, multi-signatures are designed for non-adaptive security models, where there is a *single* honest signer, and we find that this technique is incompatible with adaptive security. Specifically, in these constructions, the signing protocol for a message m involves a commitment key ck_m, which is derived using a random oracle. To simulate the honest signer, the reduction must embed a trapdoor into ck_m. Crucially, the generation of this trapdoor depends on the public key of the sole honest signer. In threshold signatures, however, there are multiple honest signers, and more critically, when generating ck_m, there is no way to predict which signers will still be honest when ck_m is eventually used. While the joint public key could be used to generate the trapdoor, this would only allow to simulate final signatures, but not signature shares of honest signers.

Merging Rounds – Naively. Rather than eliminating an entire round, we could consider merging our additional ρ_i-round into the first round of $\mathsf{Twinkle}$. The challenge with this approach is that the first round of $\mathsf{Twinkle}$ already depends on h. Specifically, signers commit to both the g-side *and the h-side* via $\mathsf{com}_i = \tilde{\mathsf{H}}(R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$, where $R_i^{(2)} = h^{r_i}$ and $\mathsf{pk}_i^{(2)} = h^{\mathsf{sk}_i}$. To compute h, however, signers first need to know all the ρ_i values. To explore a potential solution, it is helpful to understand what would go wrong if we simply omitted the h-side from the commitments, i.e., defined $\mathsf{com}_i = \tilde{\mathsf{H}}(R_i^{(1)})$. This would allow us to merge the two rounds, with signers computing and revealing their h-side after the merged first round. Intuitively, this naive approach introduces a security flaw: while malicious signers would be unable to bias the combined nonce $R^{(1)}$ due to the commitments, they would gain *full control* over the combined nonce $R^{(2)}$. Crucially, the adversary would know the random oracle input for the challenge $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$ *before* the reduction, As a consequence, the reduction cannot use any random oracle programming techniques to simulate honest signers.

[5]A trivial threshold signature would just be the concatenation of single-signer signatures from at least $t+1$ signers.

Twinkle's Proof. To better understand the issues caused by naively merging the first two rounds, we must first examine how the reduction in $\mathsf{Twinkle}$ uses the commitments. For simplicity, consider a scenario with one honest signer i and one malicious signer j, while also ignoring $\mathsf{pk}_i^{(2)}$ for now[6]. The reduction begins by sending a random commitment com_i. Upon receiving com_j from the malicious signer, it extracts $R_j^{(1)}$ and $R_j^{(2)}$ by searching through the random oracle queries. If no preimage is found, the malicious party will almost certainly fail to open the commitment in the next round. Once $R_j^{(1)}$ and $R_j^{(2)}$ are extracted, the reduction chooses a challenge c, samples a random response s_i, and generates $R_i^{(1)}$ and $R_i^{(2)}$ using honest-verifier zero-knowledge. It then calculates the combined nonces $R^{(1)}$ and $R^{(2)}$, and programs the random oracles so that $\mathsf{com}_i = \tilde{\mathsf{H}}(R_i^{(1)}, R_i^{(2)})$ and $c = \bar{\mathsf{H}}(\mathsf{pk}, R^{(1)}, R^{(2)}, \mathsf{m})$. The entropy in $R^{(1)}$ ensures that the random oracles remain unprogrammed until this point. The reduction finally reveals $R_i^{(1)}$ and $R_i^{(2)}$ to the adversary and continues the protocol, relying on honest-verifier zero-knowledge to send s_i as the final response. This strategy hinges on the reduction knowing the random oracle input to $\bar{\mathsf{H}}$ before the adversary does. If the h-side were omitted from the commitments, the reduction could not extract $R_j^{(2)}$ and could not compute the combined nonce $R^{(2)}$. As a consequence, the reduction could not program $\bar{\mathsf{H}}$ accordingly, making it impossible for the reduction to generate the g-side $(R_i^{(1)}, s_i)$ without the secret key sk_i.

Our Intuition. To address this issue, we make a key observation: if $R_j^{(2)}$ is computed honestly as $R_j^{(2)} = h^{r_j}$, then it is *uniquely determined* by $R_j^{(1)} = g^{r_j}$. Even better: we know that messages we have to sign are in the g-space (cf. Section 2.1), i.e., $h = g^{\delta_\mathsf{m}}$ for a δ_m known to the reduction. This implies that $R_j^{(2)} = h^{r_j} = (R_j^{(1)})^{\delta_\mathsf{m}}$, allowing the reduction to compute $R_j^{(2)}$ directly from $R_j^{(1)}$. In other words, having the adversary commit to the g-side is as good as committing to both sides. Of course, this intuition relies on $R_j^{(1)}$ and $R_j^{(2)}$ sharing the same discrete logarithm with respect to g and h. To ensure the reduction only deals with this case, we need an efficient way for honest signers to verify this relationship and abort if it fails. Luckily, we can design a very efficient non-interactive zero-knowledge argument for that[7]. Concretely, signers would now only commit to the g-side, but when they open the commitments and reveal the h-side, they provide a non-interactive argument that the g-side and h-side are consistent[8] – i.e., that $R_j^{(1)}$ and $R_j^{(2)}$ share the same discrete logarithm r_j, and pk_j and $\mathsf{pk}_j^{(2)}$ share the same discrete logarithm sk_j. In the security proof, the reduction would extract the g-side from the commitments, precompute the h-side using δ_m, and then program $\bar{\mathsf{H}}$ as done in the $\mathsf{Twinkle}$ proof. Soundness guarantees that the precomputed h-side will match whatever the malicious signer sends, while zero-knowledge allows the reduction to simulate the argument without needing the discrete logarithms r_i and sk_i.

Adaptive Security Madness. We have made significant progress, and it may seem like we are finished. However, a final technical issue arises when considering adaptive corruptions. To illustrate this, consider an honest signer i. Suppose the reduction has simulated a non-interactive argument π for this signer during a signing interaction. If the adversary later corrupts signer i, the reduction must query its one-more CDH oracle to obtain sk_i and provide it to the adversary, as outlined in Section 2.1. However, the reduction also needs to provide the entire internal state of that signer. In particular, this includes the random coins that signer i would have used to generate π in the protocol. Since the proof was not computed honestly by the reduction, there is no straightforward way to retrieve these coins. This issue has already been pointed out informally in recent work [BLSW24]. Their solution involves relying on secure erasures, requiring each party to erase their random coins after computing π. As we want to avoid secure erasures, we instead rely on a non-standard notion of zero-knowledge inspired by explainable arguments [HK22]. Specifically, we require that the zero-knowledge simulator can provide well-distributed random coins for any simulated proof as soon as it learns the witness. We show that our non-interactive argument achieves this notion.

[6]The complexity of $\mathsf{Twinkle}$'s proof largely stems from the more intricate scenarios where multiple honest signers receive inconsistent commitments from the adversary. To handle this, the authors introduced the use of random oracles on equivalence classes, a technique we will also adopt in our proof. For the purposes of this overview, however, we can illustrate the key challenges and our solution by considering a simplified setting.

[7]Since we later aim for an instantiation from DDH, we need an efficient non-interactive argument system for general tagged linear functions, which we show can be designed both generically and efficiently.

[8]Note that this non-interactive argument does not have to be included in the signature.

3 Preliminaries

We denote the security parameter by λ and assume that all algorithms get λ in unary as input. For a finite set S, we write $x \stackrel{\$}{\leftarrow} S$ to denote that x is sampled uniformly at random from S. For a probabilistic algorithm $\mathcal{A}$, we write $s := \mathcal{A}(x; \varrho)$ to denote output assignment when $\mathcal{A}$ is run on input x with random coins ϱ. If ϱ is sampled uniformly at random, we also write $s \leftarrow \mathcal{A}(x)$. Further, we write $s \in \mathcal{A}(x)$ to denote that s is a possible output of $\mathcal{A}$ on input x (i.e., there exist random coins ϱ such that $s = \mathcal{A}(x; \rho)$). For an integer $x \in \mathbb{N}$, we define $[\![x]\!] := \{0, \dots, x\}$ and $[x] := \{1, \dots, x\}$. In all our games, if not specified otherwise, numerical variables are implicitly initialized with 0, and lists and sets are initialized with $\emptyset$. We use standard cryptographic terminology such as negligible, overwhelming, and PPT.

3.1 Threshold Signatures

We define the syntax and security of (three-round) threshold signatures with trusted key generation, following previous works [BLT+24].

Syntax. A (t, n)-threshold signature scheme is a tuple of PPT algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ defined as follows. The setup algorithm $\mathsf{Setup}(1^\lambda)$ outputs system parameters par, and the key generation algorithm $\mathsf{Gen}(\mathsf{par})$ outputs a public key pk and secret key shares $\mathsf{sk}_1, \dots, \mathsf{sk}_n$. Further, the signing protocol Sig is split into four algorithms $(\mathsf{Sig}_0, \mathsf{Sig}_1, \mathsf{Sig}_2, \mathsf{Combine})$. Roughly, each algorithm Sig_j specifies how a signer locally computes its protocol message pm_{j+1} for the subsequent round and updates its internal state. In more detail, $\mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m})$ takes as input the signer set S, the index of the signer $i \in [n]$, its secret key share sk_i, and the message m to be signed, and it outputs a protocol message pm_1 and a state St_1. And Sig_j for $j \in [2]$ takes as input the signer's current state and the list $\mathcal{M}_j$ of all protocol messages from the previous round, and it outputs a protocol message pm_{j+1} and an updated state St_{j+1}. Finally, the combine algorithm $\mathsf{Combine}(S, \mathsf{m}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3)$ allows to convert the transcript of all protocol messages into a compact signature σ, which can then be verified using the verification algorithm $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$. Completeness of the scheme then requires for all such parameters and keys, a signature generated from a signing protocol among $t + 1$ honestly behaving parties outputs a signature for which Ver outputs 1.

Definition 1 (Threshold Signature Scheme)**.** Let $t < n$ be natural numbers. A (three-round) (t, n)-threshold signature scheme is a tuple of PPT algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$ takes as input the security parameter 1^λ and outputs public system parameters par, where par implicitly defines sets of public keys, secret keys, messages and signatures. We assume that all other algorithms implicitly take par as input.
- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk}_1, \dots, \mathsf{sk}_n)$ takes as input system parameters par, and outputs a public key pk and secret key shares $\mathsf{sk}_1, \dots, \mathsf{sk}_n$.
- $\mathsf{Sig} = (\mathsf{Sig}_0, \mathsf{Sig}_1, \mathsf{Sig}_2, \mathsf{Combine})$ is split into four algorithms:
  - $\mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m}) \to (\mathsf{pm}_1, St_1)$ takes as input a signer set $S \subseteq [n]$, an index $i \in [n]$, a secret key share sk_i, and a message m, and outputs a protocol message pm_1 and a state St_1.
  - $\mathsf{Sig}_1(St_1, \mathcal{M}_1) \to (\mathsf{pm}_2, St_2)$ takes as input a state St_1 and a tuple $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \dots, \mathsf{pm}_{1,l})$ of protocol messages, and outputs a protocol message pm_2 and a state St_2.
  - $\mathsf{Sig}_2(St_2, \mathcal{M}_2) \to \mathsf{pm}_3$ takes as input a state St_2 and a tuple $\mathcal{M}_2 = (\mathsf{pm}_{2,1}, \dots, \mathsf{pm}_{2,l})$ of protocol messages, and outputs a protocol message pm_3.
  - $\mathsf{Combine}(S, \mathsf{m}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3) \to \sigma$ takes as input a signer set $S \subseteq [n]$, a message m, tuples $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \dots, \mathsf{pm}_{1,l}), \mathcal{M}_2 = (\mathsf{pm}_{2,1}, \dots, \mathsf{pm}_{2,l})$, and $\mathcal{M}_3 = (\mathsf{pm}_{3,1}, \dots, \mathsf{pm}_{3,l})$ of protocol messages, and outputs a signature σ.
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma) \to b$ is deterministic, takes as input a public key pk, a message m, and a signature σ, and outputs a decision bit $b \in \{0, 1\}$.

We require that TS is complete in the following sense. For all $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \in \mathsf{Gen}(\mathsf{par})$, all messages m, and all $S \subseteq [n]$ with $|S| = t+1$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma) = 1 \;\middle|\; \sigma \leftarrow \mathsf{TS.Exec}(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, S, \mathsf{m})\;\right] = 1,$$

where algorithm $\mathsf{TS.Exec}$ is defined in Figure 1.

```
Alg $\mathsf{TS.Exec}(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n, S, \mathsf{m})$
01 if $|S| \neq t+1 \vee S \not\subseteq [n]$ : return $\bot$
02 parse $\{i_1, \ldots, i_{t+1}\} := S$ s.t. $i_1 < \cdots < i_{t+1}$
03 for $j \in [t+1]$ : $(\mathsf{pm}_{1,i_j}, St_{1,i_j}) \leftarrow \mathsf{Sig}_0(S, i_j, \mathsf{sk}_{i_j}, \mathsf{m})$
04 $\mathcal{M}_1 := (\mathsf{pm}_{1,i_1}, \ldots, \mathsf{pm}_{1,i_{t+1}})$
05 for $j \in [t+1]$ : $(\mathsf{pm}_{2,i_j}, St_{2,i_j}) \leftarrow \mathsf{Sig}_1(St_{1,i_j}, \mathcal{M}_1)$
06 $\mathcal{M}_2 := (\mathsf{pm}_{2,i_1}, \ldots, \mathsf{pm}_{2,i_{t+1}})$
07 for $j \in [t+1]$ : $\mathsf{pm}_{3,i_j} \leftarrow \mathsf{Sig}_2(St_{2,i_j}, \mathcal{M}_2)$
08 $\mathcal{M}_3 := (\mathsf{pm}_{3,i_1}, \ldots, \mathsf{pm}_{3,i_{t+1}})$
09 return $\sigma \leftarrow \mathsf{Combine}(S, \mathsf{m}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3)$
```

Figure 1: Algorithm $\mathsf{TS.Exec}$ for a (three-round) (t, n)-threshold signature scheme $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. The algorithm models an honest execution of the signing protocol.

Security. We define our security game following [BLT+24], which is an interactive version of the $\mathsf{TS\text{-}SUF\text{-}0}$ unforgeability notion put forth by Bellare et al. [BTZ22, BCK+22]. We formally define the security game in Figure 2 and give a verbal description next. The adversary gets system parameters par and an honestly generated public key pk as input. At any point in time throughout the game, the adversary can corrupt an honest party i by calling the oracle $\mathtt{Corrupt}(i)$, for up to t parties. Upon corruption, the adversary obtains party i's secret key sk_i and the internal state for all signing sessions party i participated in, both ongoing and completed ones. Further, the adversary can initiate a new signing session sid with some specified signer set S and message m by calling the oracle $\mathtt{Next}(sid, S, \mathsf{m})$. After initiating a new signing session, the adversary can interact with honest signers in these signing sessions. We model this for each signing protocol round via the signing oracles $\mathtt{Sig}_0, \mathtt{Sig}_1, \mathtt{Sig}_2$. For each oracle $\mathtt{Sig}_j$, the adversary can specify an honest signer i and a session identifier sid, conditioned on this signer is already in the respective round for this session sid (this is checked by an algorithm $\mathsf{Allowed}$). For $\mathtt{Sig}_j$ with $j \in [2]$, the adversary can further specify the messages of the previous round of other signers. In particular, the adversary could send different messages to two different honest signers within the same session, and as such, we do not assume a broadcast channel. Also, the adversary could send messages to an honest signer i on behalf of another honest signer j that deviate from what signer j actually sent, and as such, we also do not assume authenticated channels. In the end, the adversary outputs a forgery (m^*, σ^*) and wins the security game if it never started a signing session for message m^* and the signature σ^* is valid. We note that an interesting research direction is to achieve tight security with a stronger variant of unforgeability, in which the adversary is allowed to start signing sessions for the forgery m^*.

On Erasures. In this security model, the private state of a signer i for a signing session sid is maintained in a map as $\mathsf{state}[sid, i]$, which is updated after each signing round. Consequently, schemes that rely on secure erasures could satisfy this security definition. For instance, the scheme could be proven secure by requiring signers to erase part of the state from an earlier round before it gets corrupted. An example of that is the threshold signature scheme by Makriyannis [Mak22], which is proven secure using erasures. However, we emphasize that in our scheme, any state from earlier rounds can be efficiently computed from the state in the current round and the secret key share. In particular, our scheme does not rely on secure erasures.

Definition 2 ($\mathsf{TS\text{-}EUF\text{-}CMA}$ Security)**.** Let $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a (t, n)-threshold signature scheme. Consider the game **TS-EUF-CMA** defined in Figure 2. We say that TS is $\mathsf{TS\text{-}EUF\text{-}CMA}$ secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

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

```
**Game** $\textbf{TS-EUF-CMA}^{\mathcal{A}}_{\mathsf{TS}}(\lambda)$
$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Gen}(\mathsf{par})$
$\mathtt{Sig} := (\mathtt{Next}, \mathtt{Sig}_0, \mathtt{Sig}_1, \mathtt{Sig}_2)$
$(\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{Sig},\mathtt{Corrupt}}(\mathsf{par}, \mathsf{pk})$
if $\mathsf{m}^* \in \mathsf{Queried}$ : return $0$
return $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}^*, \sigma^*)$

**Oracle** $\mathtt{Corrupt}(i)$
if $|\mathsf{Corrupted}| \geq t$ : return $\perp$
$\mathsf{Corrupted} := \mathsf{Corrupted} \cup \{i\}$
return $(\mathsf{sk}_i, \mathsf{state}[\cdot, i])$

**Oracle** $\mathtt{Next}(sid, S, \mathsf{m})$
if $|S| \neq t+1 \vee S \not\subseteq [n]$ : return $\perp$
if $sid \in \mathsf{Sessions}$ : return $\perp$
$\mathsf{Sessions} := \mathsf{Sessions} \cup \{sid\}$
$\mathsf{message}[sid] := \mathsf{m}, \ \mathsf{signers}[sid] := S$
$\mathsf{Queried} := \mathsf{Queried} \cup \{\mathsf{m}\}$
for $i \in S$ : $\mathsf{round}[sid, i] := 0$

**Oracle** $\mathtt{Sig}_0(sid, i)$
if $\mathsf{Allowed}(sid, i, 0, \perp) = 0$ :
    return $\perp$
$S := \mathsf{signers}[sid], \ \mathsf{m} := \mathsf{message}[sid]$
$(\mathsf{pm}, St) \leftarrow \mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m})$
$\mathsf{pm}_1[sid, i] := \mathsf{pm}, \ \mathsf{state}[sid, i] := St$
$\mathsf{round}[sid, i] := 1$
return $\mathsf{pm}$

**Oracle** $\mathtt{Sig}_1(sid, i, \mathcal{M}_1)$
if $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 0$ :
    return $\perp$
$(\mathsf{pm}, St) \leftarrow \mathsf{Sig}_1(\mathsf{state}[sid, i], \mathcal{M}_1)$
$\mathsf{pm}_2[sid, i] := \mathsf{pm}, \ \mathsf{state}[sid, i] := St$
$\mathsf{round}[sid, i] := 2$
return $\mathsf{pm}$

**Oracle** $\mathtt{Sig}_2(sid, i, \mathcal{M}_2)$
if $\mathsf{Allowed}(sid, i, 2, \mathcal{M}_2) = 0$ :
    return $\perp$
$\mathsf{pm} \leftarrow \mathsf{Sig}_2(\mathsf{state}[sid, i], \mathcal{M}_2)$
$\mathsf{round}[sid, i] := 3$
return $\mathsf{pm}$

**Alg** $\mathsf{Allowed}(sid, i, r, \mathcal{M})$
if $sid \notin \mathsf{Sessions}$ : return $0$
$S := \mathsf{signers}[sid], \ H := S \setminus \mathsf{Corrupted}$
if $i \notin H$ : return $0$
if $\mathsf{round}[sid, i] \neq r$ : return $0$
if $r > 0$ :
    parse $(\mathsf{pm}_i)_{i \in S} := \mathcal{M}$
    if $\mathsf{pm}_i \neq \mathsf{pm}_r[sid, i]$ : return $0$
return $1$
```

Figure 2: The game **TS-EUF-CMA** for a (three-round) (t, n)-threshold signature scheme $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ and an adversary $\mathcal{A}$.

3.2 Tagged Linear Function Families

Following [BLT+24], we make use of the abstraction of tagged linear function families. Consider a field $\mathcal{S}$ (the scalars), a set $\mathcal{T}$ (the tags), and vector spaces $\mathcal{D}$ (the domain) and $\mathcal{R}$ (the range) over $\mathcal{S}$, all parameterized by some public parameters par. Then, a tagged linear function family is a tuple $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ of PPT algorithms, where $\mathsf{Gen}(1^\lambda)$ generates such parameters and for given parameters, T realizes a function $\mathsf{T}\colon \mathcal{T} \times \mathcal{D} \to \mathcal{R}$, such that for any fixed $g \in \mathcal{T}$, $\mathsf{T}(g, \cdot)$ is a vector space homomorphism.

Definition 3 (Tagged Linear Function Family)**.** A tagged linear function family (TLFF) is a tuple of PPT algorithms $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{par}$ takes as input the security parameter 1^λ and outputs parameters par. We assume that par implicitly defines the following sets: a set of scalars $\mathcal{S}_{\mathsf{par}}$, which forms a field; a set of tags $\mathcal{T}_{\mathsf{par}}$; a domain $\mathcal{D}_{\mathsf{par}}$ and a range $\mathcal{R}_{\mathsf{par}}$, where each forms a vector space over $\mathcal{S}_{\mathsf{par}}$. If par is clear from the context, we omit the subscript par. We naturally denote the operations of these fields and vector spaces by $+$ and $\cdot$, and assume that these operations can be evaluated efficiently.

- $\mathsf{T}(\mathsf{par}, g, x) \to X$ is deterministic, takes as input parameters par, a tag $g \in \mathcal{T}$, and a domain element $x \in \mathcal{D}$, and outputs a range element $X \in \mathcal{R}$. For all parameters par, and for all tags $g \in \mathcal{T}$, the function $\mathsf{T}(\mathsf{par}, g, \cdot)$ realizes a homomorphism, i.e.,

$$\forall s \in \mathcal{S}, x, y \in \mathcal{D} : \ \mathsf{T}(\mathsf{par}, g, s \cdot x + y) = s \cdot \mathsf{T}(\mathsf{par}, g, x) + \mathsf{T}(\mathsf{par}, g, y).$$

  For T, we also omit the input par if it is clear from the context.

A *regular* tagged linear function family additionally satisfies that for an overwhelming fraction of pairs (par, g), the images of random domain elements under $\mathsf{TLF}(g, \cdot)$ are uniform over $\mathcal{R}$. We usually denote the set of such pairs by Reg and call this the regularity set.

Definition 4 (Regular TLFF)**.** Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family. We say that TLF is ε_{r}-regular, if there is a set Reg (called the regularity set) such that the following two properties hold:

- We have
$$\Pr\left[(\mathsf{par}, g) \notin \mathsf{Reg} \mid \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda),\ g \xleftarrow{\$} \mathcal{T}\right] \leq \varepsilon_{\mathsf{r}}.$$
- For any fixed $(\mathsf{par}, g) \in \mathsf{Reg}$, the following distributions are the same:
$$\{(\mathsf{par}, g, X) \mid X \xleftarrow{\$} \mathcal{R}\} \text{ and } \{(\mathsf{par}, g, X) \mid x \xleftarrow{\$} \mathcal{D},\ X := \mathsf{T}(\mathsf{par}, g, x)\}.$$

A second property that we require is *translatability.* It means that there is an efficient way of setting up a tag h with a trapdoor from a given tag g, formally $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}, g)$, such that (1) for a random g the distribution of (g, h) is statistically close to uniform over $\mathcal{T}^2$, and (2) there are deterministic polynomial-time algorithms $\mathsf{Translate}, \mathsf{InvTranslate}$ with $\mathsf{Translate}(\mathsf{td}, \mathsf{T}(g, x)) = \mathsf{T}(h, x)$ and $\mathsf{InvTranslate}(\mathsf{td}, \mathsf{T}(h, x)) = \mathsf{T}(g, x)$ for all $x \in \mathcal{D}$. That is, images under g can be translated to images under h and vice versa.

Definition 5 (Translatability)**.** Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family. We say that TLF is ε_{t}-translatable, if there is a PPT algorithm Shift and a deterministic polynomial-time algorithms $\mathsf{Translate}, \mathsf{InvTranslate}$, such that the following properties hold:

- **Well-Distributed Tags.** The statistical distance between the following distributions $\mathcal{X}_0$ and $\mathcal{X}_1$ is at most ε_{t}:
$$\mathcal{X}_0 := \left\{(\mathsf{par}, g, h) \;\middle|\; \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda),\ g \xleftarrow{\$} \mathcal{T},\ h \xleftarrow{\$} \mathcal{T}\right\},$$
$$\mathcal{X}_1 := \left\{(\mathsf{par}, g, h) \;\middle|\; \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda),\ g \xleftarrow{\$} \mathcal{T},\ (h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}, g)\right\}.$$
- **Translation Completeness.** For every $\mathsf{par} \in \mathsf{Gen}(1^\lambda)$, for any $g \in \mathcal{T}$, any $x \in \mathcal{D}$, and any $(h, \mathsf{td}) \in \mathsf{Shift}(\mathsf{par}, g)$, we have
$$\mathsf{Translate}(\mathsf{td}, \mathsf{T}(g, x)) = \mathsf{T}(h, x) \text{ and } \mathsf{InvTranslate}(\mathsf{td}, \mathsf{T}(h, x)) = \mathsf{T}(g, x).$$

The central security property that tagged linear function families need to satisfy is called *t-algebraic translation resistance.* Intuitively, 0-algebraic translation resistance states that no efficient adversary that is given two uniform tags $g, h \in \mathcal{T}$ and an image $X_0 = \mathsf{T}(g, x_0)$ can compute $\mathsf{T}(h, x_0)$. Even more, it asks that this even holds in an interactive one-more fashion: the adversary gets $X_i = \mathsf{T}(g, x_i)$ for $i \in [\![t]\!]$ and uniform $x_i \in \mathcal{D}$, and gets t-time oracle access to an oracle $\mathtt{Inv}(\alpha_0, \ldots, \alpha_t)$ that outputs the linear combination $\sum_{i=0}^{t} \alpha_i x_i$. The adversary wins if it outputs $t+1$ elements X_i', $i \in [\![t]\!]$, such that X_i' and X_i have the same preimage with respect to tags h and g, respectively.

Definition 6 (Algebraic Translation Resistance)**.** Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family, and $t \in \mathbb{N}$ be a number. Consider the game **A-TRAN-RES** defined in Figure 3. We say that TLF is t-algebraic translation resistant, if for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:
$$\mathsf{Adv}^{t\text{-}\mathsf{A\text{-}TRAN\text{-}RES}}_{\mathcal{A},\mathsf{TLF}}(\lambda) := \Pr\left[t\text{-}\mathbf{A\text{-}TRAN\text{-}RES}^{\mathcal{A}}_{\mathsf{TLF}}(\lambda) \Rightarrow 1\right].$$

4 Non-Interactive Arguments for Tagged Linear Functions

In our construction, we use a non-interactive zero-knowledge argument system [BFM88]. The relations we consider are defined for any given tagged linear function family, and are a natural analogue of discrete logarithm equality.In this section, we sketch the definition of non-interactive arguments and our

```
**Game** $t$-**A-TRAN-RES**$^{\mathcal{A}}_{\mathsf{TLF}}(\lambda)$
$\mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda)$
$g, h \xleftarrow{\$} \mathcal{T},\ x_0, \ldots, x_t \xleftarrow{\$} \mathcal{D}$
**for** $i \in [\![t]\!] : X_i := \mathsf{T}(g, x_i)$
$(X'_i)_{i=0}^t \leftarrow \mathcal{A}^{\mathtt{Inv}}(\mathsf{par}, g, h, (X_i)_{i=0}^t)$
**if** $\forall i \in [\![t]\!] (X_i, X'_i) \in \mathsf{Im}(\mathsf{par}, g, h)$ :
    **return** $1$
**return** $0$
```

```
**Oracle** $\mathtt{Inv}(\alpha_0, \ldots, \alpha_t)$
**if** $q \geq t$ : **return** $\bot$
$q := q + 1$
$x := \sum_{i=0}^t \alpha_i x_i$
**return** $x$
```

Figure 3: Game **A-TRAN-RES** for a tagged linear function family $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ and adversary $\mathcal{A}$, where $\mathsf{Im}(\mathsf{par}, g, h)$ is defined as the set of pairs $(X, X') \in \mathcal{R}^2$ such that there is a $x \in \mathcal{D}$ with $\mathsf{T}(g, x) = X$ and $\mathsf{T}(h, x) = X'$.

construction. We provide formal details in Appendix A. Notably, our construction is both generic and concretely efficient.

Explainable Non-Interactive Arguments. Consider any **NP** relation $\mathcal{R}$ of statement-witness pairs $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$. Informally, a non-interactive argument system for $\mathcal{R}$, with respect to a random oracle H, is a pair $\mathsf{AS} = (\mathsf{Prove}, \mathsf{VerProof})$ of PPT algorithms. The algorithm $\mathsf{Prove}^{\mathsf{H}}$ takes a pair $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$ as input and outputs a proof π. The algorithm $\mathsf{VerProof}^{\mathsf{H}}$, on input the statement stmt and the proof π, decides whether to accept or reject the proof. Completeness ensures that honestly generated proofs for $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$ are accepted. We require our argument system to satisfy zero-knowledge, meaning there exists an efficient simulator that can generate simulated proofs without knowledge of the witness witn, by appropriately programming the random oracle H. Additionally, to handle adaptive corruptions while avoiding the need for secure erasures, we introduce a more nuanced extension of zero-knowledge. In this notion, the simulator, after having simulated a proof, must be able to generate valid random coins that lead to this proof after learning the witness. This property is critical for our security reduction, where the simulator may need to produce proofs on behalf of honest parties and later reconstruct their entire internal state upon corruption. Without erasures, this internal state includes the random coins used in proof generation. Following [HK22], we call this notion *explainable zero-knowledge*, see Definition 9. Lastly, we require the argument system to be weakly simulation-sound, meaning no efficient adversary can produce valid proofs for false statements, even after observing simulated proofs. In our application, we do not need to simulate proofs for invalid statements, which is why we call this notion *weak* simulation-soundness. In particular, weak simulation-soundness is implied by explainable zero-knowledge and soundness. Detailed definitions can be found in Appendix A.1.

The TLF Relation. Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family. As in Section 3.2, we denote the scalars, domain, range, and set of tags by $\mathcal{S}, \mathcal{D}, \mathcal{R}$, and $\mathcal{T}$, respectively. For fixed parameters par and a tag $g \in \mathcal{T}$, we consider the relation $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$, which is defined as

$$\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g] := \left\{ (\mathsf{stmt}, \mathsf{witn}) \in (\mathcal{T} \times \mathcal{R}^4) \times \mathcal{D}^2 \;\middle|\; \begin{array}{l} R_1 = \mathsf{T}(g, r),\ R_2 = \mathsf{T}(h, r), \\ X_1 = \mathsf{T}(g, x),\ X_2 = \mathsf{T}(h, x) \end{array} \right\}.$$

Here, $\mathsf{stmt} = (h, R_1, R_2, X_1, X_2)$ is the statement and $\mathsf{witn} = (r, x)$ is the witness. Intuitively, for two tags g, h, valid statements correspond to pairs of images (R_1, R_2) and (X_1, X_2) of the same domain element x and r, respectively. For our analysis (concretely, for zero-knowledge), we will assume that TLF is ε_{r}-regular with regularity set Reg and that $(\mathsf{par}, g) \in \mathsf{Reg}$.

Construction. We construct a non-interactive argument system $\mathsf{AS}[\mathsf{TLF}] = (\mathsf{Prove}, \mathsf{VerProof})$ for relation $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$ with respect to a random oracle $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$. Our starting point is the observation that the mapping $\mathcal{D} \to \mathcal{R}^2$ with $x \mapsto (X_1, X_2)$ and $r \mapsto (R_1, R_2)$ as in the relation is linear. Therefore, we can easily design a Σ-protocol via a generic template for linear functions. Via the Fiat-Shamir paradigm [FS87], we get a non-interactive argument system for the sub-statements (h, X_1, X_2) and (h, R_1, R_2). To improve efficiency, we additionally apply a batching step to combine the two sub-statements into one, namely, into $(h, X_1 + \gamma R_1, X_2 + \gamma R_2)$ for a random $\gamma \in \mathcal{S}$. We present the construction in Figure 4. Completeness follows by inspection. The proofs of explainable zero-knowledge (Lemma 1) and soundness (Lemma 2) are postponed to Appendix A.2. Combining Lemmata 1 and 2 and Lemma 3, we also obtain weak simulation-soundness of $\mathsf{AS}[\mathsf{TLF}]$.

```
Alg Prove^H(stmt, witn)
parse $(h, R_1, R_2, X_1, X_2) := \mathsf{stmt}$
parse $(r, x) := \mathsf{witn}$
$\gamma := \mathsf{H}(0, \mathsf{stmt})$
$\bar{X} := (R_1 + \gamma X_1, R_2 + \gamma X_2)$
$\overline{\mathsf{stmt}} := (h, \bar{X})$
$\overline{\mathsf{witn}} := \bar{x} := r + \gamma x \in \mathcal{D}$
$w \xleftarrow{\$} \mathcal{D},\ W := \varphi_{g,h}(w) \in \mathcal{R}^2$
$c := \mathsf{H}(1, \overline{\mathsf{stmt}}, W)$
$z := c\bar{x} + w \in \mathcal{D}$
return $\pi := (c, z) \in \mathcal{S} \times \mathcal{D}$

Alg VerProof^H(stmt, π)
parse $(h, R_1, R_2, X_1, X_2) := \mathsf{stmt}$
parse $(c, z) := \pi$
$\gamma := \mathsf{H}(0, \mathsf{stmt})$
$\bar{X} := (R_1 + \gamma X_1, R_2 + \gamma X_2)$
$\overline{\mathsf{stmt}} := (h, \bar{X})$
$W := \varphi_{g,h}(z) - c\bar{X}$
if $\mathsf{H}(1, \overline{\mathsf{stmt}}, W) = c$ : return 1
return 0
```

Figure 4: The non-interactive argument system $\mathsf{AS}[\mathsf{TLF}] = (\mathsf{Prove}, \mathsf{VerProof})$ for relation $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$ with respect to a random oracle $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$, where $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ is a tagged linear function family with scalars, domain, range, and set of tags by $\mathcal{S}, \mathcal{D}, \mathcal{R}$, and $\mathcal{T}$, respectively. The function $\varphi_{g,h}\colon \mathcal{D} \to \mathcal{R}^2$ is defined as $w \mapsto (\mathsf{T}(g, w), \mathsf{T}(h, w))$.

Lemma 1. *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family with set of scalars* $\mathcal{S}$*, and let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$ *be a random oracle. Further, assume that* TLF *is* ε_{r}*-regular with regularity set* Reg *and that* $(\mathsf{par}, g) \in \mathsf{Reg}$*. Then,* $\mathsf{AS}[\mathsf{TLF}]$ *satisfies* $\varepsilon_{\mathsf{xzk}}$*-explainable zero-knowledge, with* $\varepsilon_{\mathsf{xzk}} \leq QQ_{\mathsf{H}}/|\mathcal{R}|$*, where* Q_{H} *denotes the number of random oracle queries and* Q *denotes the number of queries to oracles* `GetProof`, `GetCoins`.

Lemma 2. *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family with set of scalars* $\mathcal{S}$*, and let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$ *be a random oracle. Then,* $\mathsf{AS}[\mathsf{TLF}]$ *satisfies satisfies* $\varepsilon_{\mathsf{snd}}$*-soundness, with* $\varepsilon_{\mathsf{snd}} \leq 2Q_{\mathsf{H}}/|\mathcal{S}|$*, where* Q_{H} *denotes the number of random oracle queries.*

5 Our Construction

In this section, we present our construction of three-round threshold signature using the abstraction of tagged linear function families. For concrete instantiations, we refer to Section 6.

5.1 Construction

Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family with set of scalars $\mathcal{S}$, domain $\mathcal{D}$, range $\mathcal{R}$, and set of tags $\mathcal{T}$. Let $\mathsf{H}\colon \{0,1\}^* \to \mathcal{T}$, $\bar{\mathsf{H}}\colon \{0,1\}^* \to \mathcal{S}$, and $\tilde{\mathsf{H}}, \hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$ be random oracles. Further, let $\mathsf{AS}[\mathsf{TLF}] = (\mathsf{Prove}, \mathsf{VerProof})$ be a non-interactive argument system for relation $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$ as specified in Section 4. For readability, we omit the random oracle associated to AS in our description. We construct a tightly secure three-round (t, n)-threshold signature scheme $\mathsf{Twinkle\text{-}T}[\mathsf{TLF}] = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. We assume that there is an implicit injection from $[n]$ into $\mathcal{S}$. Further, let $\ell_{i,S}(x) := \prod_{j \in S \setminus \{i\}} (j - x)/(j - i) \in \mathcal{S}$ denote the i-th Lagrange coefficient for all $i \in [n]$ and $S \subseteq [n]$, and let $\ell_{i,S} := \ell_{i,S}(0)$. We give a verbal description of our scheme and present it formally as pseudocode in Figure 8.

Setup and Key Generation. All parties have access to the public parameters $\mathsf{par} \leftarrow \mathsf{TLF.Gen}(1^\lambda)$ which define the function T, and sets $\mathcal{S}, \mathcal{T}, \mathcal{D}$, and $\mathcal{R}$, and to a random tag $g \xleftarrow{\$} \mathcal{T}$. To generate keys, elements $a_j \xleftarrow{\$} \mathcal{D}$ for $j \in [\![t]\!]$ are sampled. These elements form the coefficients of a polynomial of degree t. For each $i \in [n]$, we define the key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for the i-th signer as

$$\mathsf{sk}_i := \sum_{j=0}^{t} a_j i^j, \quad \mathsf{pk}_i := \mathsf{T}(g, \mathsf{sk}_i).$$

The joint public key is defined as $\mathsf{pk} := \mathsf{pk}_0 := \mathsf{T}(g, a_0)$.

Signing Protocol. Let $S \subseteq [n]$ be a set of signers of size $t + 1$. We assume that all signers are aware of the set S and a message $\mathsf{m} \in \{0,1\}^*$ to be signed. Then, they run the following protocol phases to compute the signature:

1. *Randomness Generation and Commitment Phase.* Each signer $i \in S$ samples $\varrho_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ and $r_i \xleftarrow{\$} \mathcal{D}$. Then, the signer computes
$$R_i^{(1)} := \mathsf{T}(g, r_i), \quad \mathsf{com}_i := \tilde{\mathsf{H}}(S, i, R_i^{(1)}).$$
It sends $\mathsf{pm}_1 := (\varrho_i, \mathsf{com}_i)$ to the other signers.

2. *Opening and Translation Phase.* Let $\mathcal{M}_1 = (\mathsf{pm}_{1,j})_{j \in S}$ be the list of messages output in the first round, where $\mathsf{pm}_{1,j} = (\varrho_j, \mathsf{com}_j)$ is sent by signer $j \in S$. First, each signer $i \in S$ computes the joint random string ϱ and derives a tag h from it. This is done by computing
$$\varrho := \hat{\mathsf{H}}(S, \mathsf{m}, (\varrho_j)_{j \in S}), \quad h := \mathsf{H}(\mathsf{m}, \varrho).$$
Then, the signer computes
$$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i), \quad R_i^{(2)} := \mathsf{T}(h, r_i).$$
Further, it computes a proof $\pi_i := \mathsf{Prove}(\mathsf{stmt}, \mathsf{witn}_i; \rho)$ for statement $\mathsf{stmt} := (h, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i, \mathsf{pk}_i^{(2)})$ with witness $\mathsf{witn}_i := (r_i, \mathsf{sk}_i)$ and uniform random coins $\rho \xleftarrow{\$} \mathcal{D}$. It sends $\mathsf{pm}_2 := (\mathsf{pk}_i^{(2)}, R_i^{(2)}, R_i^{(1)}, \pi_i)$ to the other signers.

3. *Response Phase.* Let $\mathcal{M}_2 = (\mathsf{pm}_{2,j})_{j \in S}$ be the list of messages output in the second round, where $\mathsf{pm}_{2,j} = (\mathsf{pk}_j^{(2)}, R_j^{(2)}, R_j^{(1)}, \pi_j)$ is sent by signer $j \in S$. For $j \in S$, let $\mathsf{stmt}_j := (h, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j, \mathsf{pk}_j^{(2)})$. Each signer $i \in S$ checks that $\mathsf{com}_j = \tilde{\mathsf{H}}(S, j, R_j^{(1)})$ and $\mathsf{VerProof}(\mathsf{stmt}_j, \pi_j) = 1$ holds for all $j \in S$. If one of these equations does not hold, the signer aborts. Otherwise, the signer defines
$$R^{(1)} := \sum_{j \in S} R_j^{(1)}, \quad R^{(2)} := \sum_{j \in S} R_j^{(2)}, \quad \mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \cdot \mathsf{pk}_j^{(2)}.$$
Then, the signer computes $c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$ and
$$s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i.$$
It sends $\mathsf{pm}_3 := s_i$ to the other signers.

The signature is $\sigma := (\mathsf{pk}^{(2)}, c, s, \varrho)$ for $s := \sum_{j \in S} s_j$ and $\varrho := \hat{\mathsf{H}}(S, \mathsf{m}, (\varrho_j)_{j \in S})$.

Signature Verification. Let pk be a public key, let $\mathsf{m} \in \{0,1\}^*$ be a message, and let $\sigma = (\mathsf{pk}^{(2)}, c, s, \varrho)$ be a signature. To verify σ with respect to public key pk and message m, one first computes $h := \mathsf{H}(\mathsf{m}, \varrho)$, and $R^{(1)} := \mathsf{T}(g, s) - c \cdot \mathsf{pk}$, $R^{(2)} := \mathsf{T}(h, s) - c \cdot \mathsf{pk}^{(2)}$. Then, one accepts the signature (i.e., outputs 1) if and only if $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$.

5.2 Security Analysis

Completeness follows by inspection. We now turn to the security analysis.

Theorem 1. *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family with set of scalars* $\mathcal{S}$*, range* $\mathcal{R}$*, and set of tags* $\mathcal{T}$*. Let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{T}$*,* $\bar{\mathsf{H}}\colon \{0,1\}^* \to \mathcal{S}$*, and* $\tilde{\mathsf{H}}, \hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$ *be random oracles. Further, let* $\mathsf{AS}[\mathsf{TLF}] = (\mathsf{Prove}, \mathsf{VerProof})$ *be a non-interactive argument system for* $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$*. Assume that* TLF *is* ε_{r}*-regular,* ε_{t}*-translatable, and* t*-algebraic translation resistant. Further, assume that* $\mathsf{AS}[\mathsf{TLF}]$ *satisfies* $\varepsilon_{\mathsf{xzk}}$*-explainable zero-knowledge and* $\varepsilon_{\mathsf{snd}}$*-soundness. Then, the scheme* $\mathsf{Twinkle\text{-}T}[\mathsf{TLF}]$ *is* $\mathsf{TS\text{-}EUF\text{-}CMA}$ *secure.*

Concretely, for any PPT algorithm $\mathcal{A}$ *that makes at most* Q_S *queries in total to oracles* $\mathtt{Sig}_0, \mathtt{Sig}_1, \mathtt{Sig}_2$ *and at most* $Q_{\mathsf{H}}, Q_{\bar{\mathsf{H}}}, Q_{\tilde{\mathsf{H}}}, Q_{\hat{\mathsf{H}}}$ *queries to oracles* $\mathsf{H}, \bar{\mathsf{H}}, \tilde{\mathsf{H}}, \hat{\mathsf{H}}$*, respectively, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*
$$\begin{aligned} \mathsf{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Twinkle\text{-}T}[\mathsf{TLF}]}(\lambda) \quad \leq \quad & \frac{Q_S^2 + Q_S Q_{\tilde{\mathsf{H}}}(t+1) + Q_{\hat{\mathsf{H}}}(Q_S + Q_{\mathsf{H}}) + Q_{\tilde{\mathsf{H}}}^2 + Q_{\hat{\mathsf{H}}}^2}{2^{2\lambda}} \\ & + \frac{Q_S^2(Q_S + t) + 2Q_S Q_{\bar{\mathsf{H}}}}{|\mathcal{R}|} + \frac{Q_{\bar{\mathsf{H}}}}{|\mathcal{S}|} + Q_{\mathsf{H}}\varepsilon_{\mathsf{t}} + 2\varepsilon_{\mathsf{r}} \\ & + 2\varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}} + \mathsf{Adv}^{t\text{-}\mathsf{A\text{-}TRAN\text{-}RES}}_{\mathcal{B},\mathsf{TLF}}(\lambda). \end{aligned}$$

Proof. Let $\mathcal{A}$ be an adversary against the security of $\mathsf{TS} := \mathsf{Twinkle\text{-}T}[\mathsf{TLF}]$. We structure our proof as a sequence of games $\mathbf{G}_0$ to $\mathbf{G}_{11}$ and a reduction to t-algebraic translation resistance.

Game $\mathbf{G}_0$: This game is the real security game $\mathbf{TS\text{-}EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ for threshold signatures: the game samples parameters par' for TLF and a random tag $g \xleftarrow{\$} \mathcal{T}$. It also samples random coefficients $a_0, \dots, a_t \xleftarrow{\$} \mathcal{D}$ and computes the public key $\mathsf{pk} := \mathsf{pk}_0 := \mathsf{T}(g, a_0)$ and secret key shares $\mathsf{sk}_i := \sum_{j=0}^{t} a_j i^j$ for each $i \in [n]$. Denote the corresponding public key shares by $\mathsf{pk}_i := \mathsf{T}(g, \mathsf{sk}_i)$. Then, the game runs $\mathcal{A}$ on input $\mathsf{par} := (\mathsf{par}', g)$ and pk with access to signing oracles, corruption oracles, and random oracles. Concretely, it gets access to random oracles $\mathsf{H}, \bar{\mathsf{H}}, \tilde{\mathsf{H}}$, and $\hat{\mathsf{H}}$, which the game provides by standard lazy sampling using maps $h[\cdot], \bar{h}[\cdot], \tilde{h}[\cdot]$, and $\hat{h}[\cdot]$, respectively. The set of corrupted parties is denoted by $\mathsf{Corrupted}$. Whenever $\mathcal{A}$ calls the signing session oracle $\mathtt{Next}$ on some valid input (sid, S, m) (i.e., $|S| = t+1$, $S \subseteq [n]$, and $sid \notin \mathsf{Sessions}$), the message m is added to the set $\mathsf{Queried}$. Finally, the adversary outputs a forgery (m^*, σ^*) and the game outputs 1 if $\mathsf{m}^* \notin \mathsf{Queried}$, $|\mathsf{Corrupted}| \leq t$, and $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}^*, \sigma^*) = 1$. By definition, we have

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

Before we proceed with our analysis, we assume that the adversary always makes exactly t distinct corruption queries. This is without loss of generality, since one could build a wrapper adversary that internally runs $\mathcal{A}$, but never issues a corruption query twice, and makes enough corruption queries before outputting its forgery. Clearly, the wrapper adversary has the same advantage and running time as $\mathcal{A}$.

Game $\mathbf{G}_1$: We rule out collisions for $\tilde{\mathsf{H}}$ (used for commitments) and $\hat{\mathsf{H}}$ (used for joint randomness). Concretely, the game aborts if there are $x \neq x'$ such that $\tilde{h}[x] = \tilde{h}[x'] \neq \bot$ or $\hat{h}[x] = \hat{h}[x'] \neq \bot$. By the birthday bound, we have

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \frac{Q^2_{\tilde{\mathsf{H}}} + Q^2_{\hat{\mathsf{H}}}}{2^{2\lambda}}.$$

Subsequent games will make use of an algorithm $\tilde{\mathsf{H}}^{-1}$ that on input y searches for an x such that $\tilde{h}[x] = y$. If no such x is found, the algorithm returns $\bot$. Note that the game guarantees the existence of at most one such preimage of y.

Game $\mathbf{G}_2$: In this game, we introduce two initially empty lists $\hat{\mathcal{L}}$ and $\mathcal{L}$ and change the game as follows:

- Whenever $\mathcal{A}$ calls the signing oracle $\mathtt{Sig}_0$ on some valid input (sid, i) (i.e., $\mathsf{Allowed}(sid, i, 0, \bot) = 1$) and the honest signer i would return its message $(\varrho_i, \mathsf{com}_i)$, the game aborts if the random oracle $\hat{\mathsf{H}}$ has been queried before on an input of the form $(S, \mathsf{m}, (\varrho'_j)_{j \in S})$ such that $\varrho'_i = \varrho_i$. Here, the signer set S and message m are defined by the session identifier sid. Otherwise, it inserts $(S, \mathsf{m}, \varrho_i, i)$ into $\hat{\mathcal{L}}$. Since ϱ_i is sampled uniformly at random from $\{0,1\}^{2\lambda}$, the probability of abort for a fixed signing query is bounded by $Q_{\hat{\mathsf{H}}}/2^{2\lambda}$. Thus, by a union bound over all signing queries, this abort happens with probability at most $Q_S Q_{\hat{\mathsf{H}}}/2^{2\lambda}$.

- Whenever the random oracle $\hat{\mathsf{H}}$ is queried on a fresh input $(S, \mathsf{m}, (\varrho_i)_{i \in S})$ such that there is a corresponding entry $(S, \mathsf{m}, \varrho_i, i) \in \hat{\mathcal{L}}$, the game samples a random $\varrho \xleftarrow{\$} \{0,1\}^{2\lambda}$ to program $\hat{\mathsf{H}}$ as before. Then, the game aborts if the random oracle H has been queried before with (m, ϱ). Otherwise, the game inserts (m, ϱ) into $\mathcal{L}$. Since ϱ is sampled uniformly at random from $\{0,1\}^{2\lambda}$, the probability of this abort for any fixed query to $\hat{\mathsf{H}}$ is at most $Q_{\mathsf{H}}/2^{2\lambda}$. Thus, by a union bound over all such queries, this abort happens with probability at most $Q_{\hat{\mathsf{H}}} Q_{\mathsf{H}}/2^{2\lambda}$.

Overall, we get

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \frac{Q_{\hat{\mathsf{H}}}(Q_S + Q_{\mathsf{H}})}{2^{2\lambda}}.$$

Game $\mathbf{G}_3$: In this game, we change how the random oracle H is programmed. Namely, we let the game sample a random tag $h^\diamond \xleftarrow{\$} \mathcal{T}$ at the beginning, and implement H as follows:

$$\mathsf{H}(\mathsf{m}, \varrho) := \begin{cases} \text{via } \mathsf{Shift}(\mathsf{par}', g), & \text{if } (\mathsf{m}, \varrho) \in \mathcal{L}, \\ \text{via } \mathsf{Shift}(\mathsf{par}', h^\diamond), & \text{otherwise.} \end{cases}$$

In more detail, the game does the following. For a query $\mathsf{H}(\mathsf{m}, \varrho)$ for which the hash value $h[\mathsf{m}, \varrho]$ is not yet defined and $(\mathsf{m}, \varrho) \in \mathcal{L}$, the game samples $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', g)$ and sets $h[\mathsf{m}, \varrho] := h$. Further, it stores td

in a map tr as $tr[\mathsf{m},\varrho] := \mathsf{td}$. On the other hand, if $(\mathsf{m},\varrho) \notin \mathcal{L}$, the game samples $(h,\mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', h^{\diamond})$, and updates the maps $tr[\mathsf{m},\varrho] := \mathsf{td}$ and $h[\mathsf{m},\varrho] := h$. Clearly, $\mathbf{G}_2$ and $\mathbf{G}_3$ are indistinguishable by the ε_{t}-translatability of TLF, applied to every random oracle query. Concretely, we have

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq Q_{\mathsf{H}}\varepsilon_{\mathsf{t}}.$$

Before we proceed with our sequence of games, we point out the following crucial observations: first, whenever an honest party computes the tag h during the signing protocol, we know that it has been generated as a shift of g and we know the corresponding trapdoor. As a result, we will be able to ensure that the game no longer needs secret key shares sk_i to compute secondary public key shares $\mathsf{pk}_i^{(2)} := \mathsf{T}(h,\mathsf{sk}_i)$ and nonces $R_i^{(2)} := \mathsf{T}(h,r_i)$ for signing queries. Instead, it can simulate using the g-side transcript $(\mathsf{pk}_i, R_i^{(1)})$ and the trapdoor of the shifted tags. In the remainder of our proof, we use careful delayed random oracle programming, observability of the random oracle, and an honest-verifier zero-knowledge-style programming to simulate the remaining parts of the signing queries without sk_i. As a result, the secret key sk_i will only be needed when the adversary corrupts parties.

The second observation is that for the forgery message m^*, no honest signer has ever been queried with m^*. Thus, m^* never occurs in $\mathcal{L}$. In particular, the tag in the forgery is a shift of $h^{\diamond}$.

<u>**Game $\mathbf{G}_4$:**</u> We let the game abort if $(\mathsf{par}', g) \notin \mathsf{Reg}$, where Reg is the regularity set of TLF. By ε_{r}-regularity of TLF, we have

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \varepsilon_{\mathsf{r}}.$$

<u>**Game $\mathbf{G}_5$:**</u> In this game, we change how the non-interactive zero-knowledge proofs π_i are computed by the signing oracle $\mathtt{Sig}_1$ for honest signers $i \in S$. Namely, we switch from honestly generated proofs π_i to simulated proofs $\tilde{\pi}_i$. Upon an adaptive corruption, we would now have to provide the random coins used for generating the proofs, as already observed in [BLSW24]. To do this, we rely on the explainable zero-knowledge notion. Concretely, whenever the game runs the zero-knowledge simulator Sim to simulate a proof on behalf of an honest signer i, the simulator would now also output a state St associated to this signer and the session, and the game would store this state (along with signer index i and session identifier sid). Then, if signer i gets corrupted later, the game would run Sim again using the state St to obtain random coins $\tilde{\rho}_i$ explaining the proof $\tilde{\pi}_i$, and it does so for every session sid for which it has simulated a proof on behalf of signer i. It then includes those random coins in the internal state that it outputs for the signer i. Clearly, $\mathbf{G}_4$ and $\mathbf{G}_5$ are indistinguishable by explainable zero-knowledge of AS, and we get

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \varepsilon_{\mathsf{xzk}}.$$

<u>**Game $\mathbf{G}_6$:**</u> In this game, we change how the secondary elements (i.e., the h-side) $\mathsf{pk}_i^{(2)}$ and $R_i^{(2)}$ are computed by the signing oracle $\mathtt{Sig}_1$. To recall, in the opening phase of the signing protocol, the signing oracle for honest party $i \in S$ in $\mathbf{G}_5$ would compute $\mathsf{pk}_i^{(2)} := \mathsf{T}(h,\mathsf{sk}_i)$ and $R_i^{(2)} := \mathsf{T}(h,r_i)$, where $h := \mathsf{H}(\mathsf{m},\varrho)$ is obtained from $\varrho := \hat{\mathsf{H}}(S,\mathsf{m},(\varrho_j)_{j\in S})$ in the commitment phase. From this game on, $\mathsf{pk}_i^{(2)}$ and $R_i^{(2)}$ are computed via translation from pk_i and $R_i^{(1)}$ using the trapdoor output by $\mathsf{Shift}(\mathsf{par}, g)$. Concretely:

$$\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m},\varrho],\mathsf{pk}_i), \quad R_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m},\varrho], R_i^{(1)}).$$

Note that $\mathbf{G}_3$ guarantees knowledge of the trapdoor $\mathsf{td} = tr[\mathsf{m},\varrho]$. In particular, this is a trapdoor for g, following the observation made after $\mathbf{G}_3$. Thus, it follows from the translation completeness of TLF that the view of $\mathcal{A}$ remains unchanged, and we get

$$\Pr[\mathbf{G}_5 \Rightarrow 1] = \Pr[\mathbf{G}_6 \Rightarrow 1].$$

<u>**Game $\mathbf{G}_7$:**</u> In this game, we change the signing oracle again. Concretely, we change $\mathtt{Sig}_0$ for the commitment phase and $\mathtt{Sig}_1$ for the opening phase. To recall, in the commitment phase of the signing protocol, the signing oracle for honest party $i \in S$ in $\mathbf{G}_6$ would sample a random element $r_i \xleftarrow{\$} \mathcal{D}$ and send the commitment $\mathsf{com}_i := \tilde{\mathsf{H}}(S,i,R_i^{(1)})$ for $R_i^{(1)} := \mathsf{T}(g,r_i)$. Later, in the opening phase, the oracle would compute $\mathsf{pk}_i^{(2)}$ and $R_i^{(2)}$ via translation as explained in $\mathbf{G}_6$, and send $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$ along with a simulated proof $\tilde{\pi}$. We change this as follows: Instead of sampling an element $r_i \xleftarrow{\$} \mathcal{D}$ and computing $R_i^{(1)}$ honestly, we let signer i send a random commitment $\mathsf{com}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in the first round. Additionally, the

game inserts an entry (S, i, com_i) into a list SimCom that keeps track of these simulated commitments. If the same commitment has been sampled by the game twice, then the game aborts: i.e., if there is already an entry $(S,', i', \mathsf{com}_i) \in \mathsf{SimCom}$ such that $(S,', i') \neq (S, i)$. By the birthday bound, this event occurs only with probability $Q_S^2/2^{2\lambda}$. We identify two situations where the preimage of com_i has to be revealed: First, the game has to output $R_i^{(1)}$ in the opening phase, and second, when party i gets corrupted, the value r_i has to be given to the adversary. To make the game's behavior clear in both situations, we make a case distinction.

- Consider the opening phase or the case where party i gets corrupted before it reaches the opening phase. In that case, we let the game sample a random $r_i \xleftarrow{\$} \mathcal{D}$, compute $R_i^{(1)} := \mathsf{T}(g, r_i)$, and then check if $\tilde{\mathsf{H}}(S, i, R_i^{(1)})$ is already defined. If it is already defined, then the game aborts. Otherwise, it defines $\tilde{h}[S, i, R_i^{(1)}] := \mathsf{com}_i$ and proceeds as before: i.e., in the opening phase, it would output $R_i^{(1)}$ (along with $R_i^{(2)}, \mathsf{pk}_i^{(2)}, \tilde{\pi}$ computed as in $\mathbf{G}_6$), and upon a corruption, it would output r_i as part of its state. We reiterate that upon a corruption, random coins for simulated proofs $\tilde{\pi}_i$ are generated by running the zero-knowledge simulator Sim as described in $\mathbf{G}_5$.
- Consider the case where party i gets corrupted *after* the opening phase. In that case, the game has already defined r_i (see the previous case). The game handles the corruption as before (i.e., using this particular r_i).

To bound the event of abort in the above, we use the regularity of TLF, which implies that $R_i^{(1)}$ is uniform over the range $\mathcal{R}$. For a fixed signing query, the probability that $(S, i, R_i^{(1)})$ matches a previous query of $\mathcal{A}$ is bounded by $Q_{\tilde{\mathsf{H}}}/|\mathcal{R}|$. Thus, by a union bound over all signing queries, the abort happens with probability at most $Q_S Q_{\tilde{\mathsf{H}}}/|\mathcal{R}|$. Overall, we get

$$|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \frac{Q_S Q_{\tilde{\mathsf{H}}}}{|\mathcal{R}|} + \frac{Q_S^2}{2^{2\lambda}}.$$

In the next part of our proof, we use the technique of random oracle programming on equivalence classes from [BLT+24]. This technique allows to simulate signing queries later using honest-verifier zero-knowledge, even when the adversary sends inconsistent sets of commitments to different honest parties. At a high level, this technique allows to identify whether two queries $q = (sid, i, \mathcal{M}_1)$ and $q' = (sid', i', \mathcal{M}_1')$ to $\mathtt{Sig}_1$ will result in the same combined nonce *before* all commitments in $\mathcal{M}_1$ and $\mathcal{M}_1'$ have preimages. Recall that in our case $\mathcal{M}_1 := ((\varrho_j, \mathsf{com}_j))_{j \in S}$, which defines $\varrho = \hat{\mathsf{H}}(S, \mathsf{m}, (\varrho_j)_{j \in S})$ and $h = \mathsf{H}(\mathsf{m}, \varrho)$. And the combined nonce is computed as $R^{(1)} = \sum_{j \in S} R_j^{(1)}$ where $\mathsf{com}_j = \tilde{\mathsf{H}}(S, j, R_j^{(1)})$. To establish this, the authors define an equivalence relation $\sim$ on such queries with the following two properties. First, the equivalence relation is preserved over time. Second, as soon as all commitments in $\mathcal{M}_1$ and $\mathcal{M}_1'$ have preimages, the equivalence relation defines identical resulting combined nonces, i.e., the resulting combined nonces are the same if and only if $q \sim q'$. Assuming such an equivalence relation, the simulation is done as follows. When the reduction has to reveal the nonce $R_i^{(1)}$ of an honest party $i \in S$, it first defines $c := \mathsf{Cl}(q)$, where $\mathsf{Cl} := \mathsf{H}/\sim$ is an internal random oracle modulo the relation $\sim$. Concretely, this means two equivalent triples are mapped to the same output, and this is well-defined as the relation stays consistent over time. Then, it defines $R_i^{(1)} := \mathsf{T}(g, s_i) - c \cdot \mathsf{pk}_i^{\ell_{i,S}}$ for a randomly sampled $s_i \xleftarrow{\$} \mathcal{D}$. On the other hand, it does not define nonces of any other honest parties at that point and thus the combined nonce may not be known yet. Instead, the random oracle programming is delayed until the combined nonce is known. We now proceed with a slightly simplified formalization of this technique and refer to the original work [BLT+24] for more details.

<u>Game $\mathbf{G}_8$:</u> In this game, we introduce a list $\mathsf{Pending}$ which keeps track of honest parties i and signing sessions sid for which the game cannot yet extract preimages of all commitments sent to party i in the commitment phase. In more detail, the list contains an entry $(sid, i, \mathcal{M}_1)$ if and only if:

(i) The signing oracle $\mathtt{Sig}_1$ has been called with valid input $(sid, i, \mathcal{M}_1)$, i.e., for this query the game did not output $\perp$ because of $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 0$. Further, at that point in time, we have: For every commitment com_j in $\mathcal{M}_1$ such that $(S, j, \mathsf{com}_j) \notin \mathsf{SimCom}$, the preimage $(S, j, R_j^{(1)}) := \tilde{\mathsf{H}}^{-1}(\mathsf{com}_j)$ is defined (and thus can be extracted), and

(ii) there is a commitment com_j in $\mathcal{M}_1$ such that $\tilde{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot$.

Further, the list is dynamically updated in the following two situations: First, whenever the adversary calls the signing oracle $\mathtt{Sig}_1$ with some valid input $(sid, i, \mathcal{M}_1)$ and the first condition is satisfied. In that case, the tuple $(sid, i, \mathcal{M}_1)$ is added to the list $\mathsf{Pending}$. Second, whenever the map $\tilde{h}[\cdot]$ is changed (i.e., during queries to $\tilde{\mathsf{H}}$ and queries to the corruption and signing oracles). In that case, an existing entry is removed from the list $\mathsf{Pending}$ when the second condition above is not satisfied anymore. In more detail, whenever the map $\tilde{h}[\cdot]$ is changed, the following is done: Initialize an empty list New and run through all entries $(sid, i, \mathcal{M}_1) \in \mathsf{Pending}$ with the following steps:

- Check if the entry still satisfies the condition (ii) above. If this is the case, keep it in $\mathsf{Pending}$.
- Otherwise, remove it from $\mathsf{Pending}$ and proceed as follows. Let $(sid, i, \mathcal{M}_1)$ be this removed tuple. In particular, all preimages $(S, j, R_j^{(1)}) := \tilde{\mathsf{H}}^{-1}(\mathsf{com}_j)$ are defined and can be extracted, where S is the signer set associated with the session sid. Determine the combined nonces and secondary public key
$$R^{(1)} = \sum_{j \in S} R_j^{(1)}, \quad \bar{R}^{(2)} = \sum_{j \in S} R_j^{(2)}, \quad \bar{\mathsf{pk}}^{(2)} = \sum_{j \in S} \ell_{j,S} \mathsf{pk}_j^{(2)},$$
where the secondary combined nonce and public key are as
$$\bar{\mathsf{pk}}^{(2)} := \mathsf{Translate}(tr[\mathsf{m}, \varrho], \mathsf{pk}), \quad \bar{R}^{(2)} := \mathsf{Translate}(tr[\mathsf{m}, \varrho], R^{(1)}).$$
Recall that $(sid, i, \mathcal{M}_1)$ defines (m, ϱ) and that $\mathbf{G}_3$ guarantees knowledge of the trapdoor $\mathsf{td} = tr[\mathsf{m}, \varrho]$. And following the observation made after $\mathbf{G}_3$, this is a trapdoor for g.
- If $(S, R^{(1)}, \bar{R}^{(2)}, \bar{\mathsf{pk}}^{(2)}, \mathsf{m}, \varrho) \notin \mathsf{New}$ but the value $\bar{\mathsf{H}}(\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho)$ is already defined, where the message m and randomness ϱ are defined by $(sid, i, \mathcal{M}_1)$, then the game aborts.
- Otherwise, sample $\bar{h}[\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho] \xleftarrow{\$} \mathcal{S}$ and insert the tuple $(S, R^{(1)}, \bar{R}^{(2)}, \bar{\mathsf{pk}}^{(2)}, \mathsf{m}, \varrho)$ into the list New.

Further, we introduce an additional abort condition. Namely, the game aborts if the following event happens: Upon a random oracle query to $\tilde{\mathsf{H}}$ for which the hash value is yet undefined and freshly sampled as $\mathsf{com} \xleftarrow{\$} \{0,1\}^{2\lambda}$, there is already an existing entry $(sid, i, \mathcal{M}_1) \in \mathsf{Pending}$ such that com is in $\mathcal{M}_1$. Having said that, the game change is now fully defined. We highlight that the only difference to how [BLT+24] defines its game change (concretely, the way $\mathsf{Pending}$ and New are defined and updated) is how the secondary combined nonce $\bar{R}^{(2)}$ and public key $\bar{\mathsf{pk}}^{(2)}$ are computed. While [BLT+24] obtains these values via extraction from $\tilde{\mathsf{H}}$, we compute them via translation from pk and $R^{(1)}$ using the trapdoor $tr[\mathsf{m}, \varrho]$ for g. Clearly, this does not affect the abort conditions defined in this game. Thus, we can directly apply the probability analysis for abort from [BLT+24], and get
$$|\Pr[\mathbf{G}_7 \Rightarrow 1] - \Pr[\mathbf{G}_8 \Rightarrow 1]| \le \frac{Q_S Q_{\tilde{\mathsf{H}}}(t+1)}{2^{2\lambda}} + \frac{Q_S Q_{\tilde{\mathsf{H}}}}{|\mathcal{R}|}.$$

Game $\mathbf{G}_9$: In this game, we introduce two algorithms Equiv and $\mathsf{GetChal}$. The first algorithm allows to group tuples $(sid, i, \mathcal{M}_1)$ that have been inserted into list $\mathsf{Pending}$ into equivalence classes, while the second algorithm behaves as a random oracle on these equivalence classes. To clarify again, the equivalence relation is defined on the set of all triples in $\mathsf{Pending}$ and all triples that already have been removed from $\mathsf{Pending}$, but not on any other tuples. In more detail, two tuples $q = (sid, i, \mathcal{M}_1)$ and $q' = (sid', i', \mathcal{M}_1')$ are equivalent if and only if the following conditions are satisfied simultaneously:

- Let S, S' and m, m' be the signer sets and messages associated with sid and sid', respectively. And let ϱ, ϱ' be the joint randomness obtained from $\mathcal{M}_1 := ((\varrho_j, \mathsf{com}_j))_{j \in S}, \mathcal{M}_1' := ((\varrho_j', \mathsf{com}_j'))_{j \in S'}$, respectively. Then, it holds that $S = S'$, $\mathsf{m} = \mathsf{m}'$, and $\varrho = \varrho'$.
- Let $F \subseteq S$ be the set of indices $j \in S$ such that $\tilde{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot$. Analogously, let $F' \subseteq S'$ be the set of indices $j \in S'$ such that $\tilde{\mathsf{H}}^{-1}(\mathsf{com}_j') = \bot$. Then, it holds that $(\mathsf{com}_j)_{j \in F} = (\mathsf{com}_j')_{j \in F'}$.

- Let $G := S \setminus F$ and $G' := S' \setminus F'$, where F, F' are defined as above. We know $(S, j, R_j^{(1)}) := \tilde{\mathsf{H}}^{-1}(\mathsf{com}_j)$ exists for all $j \in G$. Similarly, $(S', j, R_j'^{(1)}) := \tilde{\mathsf{H}}^{-1}(\mathsf{com}'_j)$ exists for all $j \in G'$. We define partially combined nonces and secondary public key for $(sid, i, \mathcal{M}_1)$ as
$$\tilde{R}^{(1)} = \sum_{j \in G} R_j^{(1)}, \quad \tilde{R}^{(2)} = \sum_{j \in G} R_j^{(2)}, \quad \tilde{\mathsf{pk}}^{(2)} = \sum_{j \in G} \ell_{j,S} \cdot \mathsf{pk}_j^{(2)},$$
where $R_j^{(2)} = \mathsf{Translate}(tr[\mathsf{m}, \varrho], R_j^{(1)})$ and $\mathsf{pk}_j^{(2)} = \mathsf{Translate}(tr[\mathsf{m}, \varrho], \mathsf{pk}_j)$ for all $j \in G$ are computed via translation. Analogously, we define partially combined nonces $\tilde{R}'^{(1)}, \tilde{R}'^{(2)}$ and secondary public key $\tilde{\mathsf{pk}}'^{(2)}$ for $(sid', i', \mathcal{M}'_1)$ through G'. Then, it holds that $(\tilde{R}^{(1)}, \tilde{R}^{(2)}, \tilde{\mathsf{pk}}^{(2)}) = (\tilde{R}'^{(1)}, \tilde{R}'^{(2)}, \tilde{\mathsf{pk}}'^{(2)})$.

To summarize, two triples $q = (sid, i, \mathcal{M}_1)$ and $q' = (sid', i', \mathcal{M}'_1)$ are equivalent if and only their signer sets, messages, joint randomness, partially combined first nonce, partially combined translated nonce and public key, and remaining commitments match. Again, we refer to the original work [BLT+24] for a proof that this indeed defines an equivalence relation. With this observation, we define an algorithm $\mathsf{GetChal}$ which assigns each equivalence class a random challenge $c \xleftarrow{\$} \mathcal{S}$ in a lazy manner. In more detail, it takes as input a tuple $(sid, i, \mathcal{M}_1)$ and checks if there is a tuple $(sid', i', \mathcal{M}'_1)$ in the same equivalence class (using algorithm Equiv) that is already assigned a challenge c. In that case, it returns this value c. Otherwise, it assigns $c \xleftarrow{\$} \mathcal{S}$ to $(sid, i, \mathcal{M}_1)$. With these two algorithms, we change the game as follows. Instead of programming the random oracle $\bar{\mathsf{H}}$ as $\bar{h}[\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho] \xleftarrow{\$} \mathcal{S}$ whenever an entry $(sid, i, \mathcal{M}_1)$ is removed from the list $\mathsf{Pending}$ and no abort occurs, we define $\bar{h}[\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho] := \mathsf{GetChal}(sid, i, \mathcal{M}_1)$. We claim that this programming does not change the view of the adversary. For this, we need to show that two different inputs $x \neq x'$ to $\bar{\mathsf{H}}$ give independently sampled outputs. Let
$$x = (\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho), \quad x' = (\mathsf{pk}, \bar{\mathsf{pk}}'^{(2)}, R'^{(1)}, \bar{R}'^{(2)}, \mathsf{m}', \varrho'),$$
and let $q := (sid, i, \mathcal{M}_1)$ and $q' := (sid', i', \mathcal{M}'_1)$ be the associated entries in $\mathsf{Pending}$ that were removed. We recall that the overline symbol in $\bar{\mathsf{pk}}^{(2)}, \bar{R}^{(2)}$ indicates that these values were obtained via translation from the g-side as introduced in $\mathbf{G}_8$. Clearly, if q and q' were not equivalent at the time of removal of the later one, then the outputs $\bar{\mathsf{H}}(x)$ and $\bar{\mathsf{H}}(x')$ are independent. On the other hand, if q and q' were indeed equivalent, then we have that $\mathsf{m} = \mathsf{m}'$, $\varrho = \varrho'$, and $(\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}) = (\mathsf{pk}'^{(2)}, R'^{(1)}, R'^{(2)})$, and thus $x = x'$. As a result, we get
$$\Pr[\mathbf{G}_8 \Rightarrow 1] = \Pr[\mathbf{G}_9 \Rightarrow 1].$$

At this point, we are ready to use an honest-verifier zero-knowledge-style simulation to simulate the remaining parts of signing without secret key shares. In particular, sk_i will only be needed upon corruptions. Intuitively, we can do that because now we know the challenge (using algorithm $\mathsf{GetChal}$) already in the opening phase before fixing honest party's nonces.

Game $\mathbf{G}_{10}$: In this game, we change the signing oracle and corruption oracle. To recall, in the opening phase of the signing protocol, the signing oracle $\mathtt{Sig}_1$ for honest party $i \in S$ would sample a random element $r_i \xleftarrow{\$} \mathcal{D}$, compute $R_i^{(1)} := \mathsf{T}(g, r_i)$, and derive $\mathsf{pk}_i^{(2)}$ and $R_i^{(2)}$ via translation from pk_i and $R_i^{(1)}$, respectively. Later, in the response phase, the party sends $s_i := c\ell_{i,S}\mathsf{sk}_i + r_i$ where $c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$ and $\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}$ are the combined secondary public key and nonces. Further, when the party gets corrupted, it has to send r_i as part of its state. We change this as follows. Instead of computing the challenge c as above, we let signer i derive c after the commitment phase (upon receiving $(sid, i, \mathcal{M}_1)$) as follows, considering two cases. First, if $(sid, i, \mathcal{M}_1)$ has not been added to the list $\mathsf{Pending}$, then the party sets $c := 0$. Note that in this case, there is by definition a commitment com_j in $\mathcal{M}_1$ such that $(S, j, \mathsf{com}_j) \notin \mathsf{SimCom}$ and the preimage $\tilde{\mathsf{H}}^{-1}(\mathsf{com}_j)$ is not defined yet. Thus, the adversary will not be able to open the commitment com_j in the opening phase and party i will never reach the response phase for this session (see abort conditions in $\mathbf{G}_8$). Otherwise, the party sets $\bar{c} := \mathsf{GetChal}(sid, i, \mathcal{M}_1)$. In either case, it samples an $s_i \xleftarrow{\$} \mathcal{D}$ and computes $R_i^{(1)} := \mathsf{T}(g, s_i) - \bar{c} \cdot \ell_{i,S} \cdot \mathsf{pk}_i$. Then, it derives $\mathsf{pk}_i^{(2)}$ and $R_i^{(2)}$ via translation from pk_i and $R_i^{(1)}$ as before. Later, in the response phase, it outputs s_i as its signature share. Further, when the party gets corrupted after the opening phase, it sets $r_i := s_i - \bar{c} \cdot \ell_{i,S} \cdot \mathsf{sk}_i$. Finally, we let the game abort if $\bar{c} \neq \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$, where $\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}$ are the combined secondary public key and combined nonces computed after the opening phase. We identify two events where the abort happens. To establish these, we assume that the signer reaches the response phase. In particular, we know that the entry $(sid, i, \mathcal{M}_1)$ has been removed from the list $\mathsf{Pending}$ at some point in time.

(1) In the opening phase, $\mathcal{A}$ sends a message $\mathsf{pm}_{2,j} := (\mathsf{pk}_j^{(2)}, R_j^{(2)}, R_j^{(1)}, \pi_j)$ such that $\mathsf{pk}_j^{(2)} \neq \bar{\mathsf{pk}}_j^{(2)}$ or $R_j^{(2)} \neq \bar{R}_j^{(2)}$ for

$$\bar{\mathsf{pk}}_j^{(2)} := \mathsf{Translate}(tr[\mathsf{m}, \varrho], \mathsf{pk}_j), \quad \bar{R}_j^{(2)} := \mathsf{Translate}(tr[\mathsf{m}, \varrho], R_j^{(1)})$$

yielded, but the proof π_j still verified. Recall that $\bar{\mathsf{pk}}_j^{(2)}$ and $\bar{R}_j^{(2)}$ were used to define the equivalence classes for $\mathsf{GetChal}$ and thus $\bar{h}[\mathsf{pk}, \bar{\mathsf{pk}}^{(2)}, R^{(1)}, \bar{R}^{(2)}, \mathsf{m}, \varrho]$ (see $\mathbf{G}_9$ and $\mathbf{G}_8$). We now argue that this event can only happen with negligible probability. For that, we first observe that in this case the statement is no longer in the language. To see this, recall our relation for the argument system in Section 4. Without loss of generality, we assume that $\mathsf{pk}_j^{(2)} \neq \bar{\mathsf{pk}}_j^{(2)}$. The other case (i.e., $R_j^{(2)} \neq \bar{R}_j^{(2)}$) can be handled analogously. By translation completeness of TLF, we know $\bar{\mathsf{pk}}_j^{(2)} = \mathsf{T}(h, \mathsf{sk}_j)$ from $\mathsf{pk}_j = \mathsf{T}(g, \mathsf{sk}_j)$, where h is the tag defined by $(sid, i, \mathcal{M}_1)$. At the same time, a valid proof π_j for $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$ tells us that there exists an $x \in \mathcal{D}$ such that $\mathsf{pk}_j = \mathsf{T}(g, x)$ and $\mathsf{pk}_j^{(2)} = \mathsf{T}(h, x)$. From this, we obtain

$$\mathsf{T}(g, \mathsf{sk}_j) = \mathsf{pk}_j = \mathsf{T}(g, x) \implies \mathsf{T}(h, \mathsf{sk}_j) = \mathsf{Translate}(tr[\mathsf{m}, \varrho], \mathsf{pk}_j) = \mathsf{T}(h, x).$$

But this clearly contradicts the assumption that $\mathsf{pk}_j^{(2)} \neq \bar{\mathsf{pk}}_j^{(2)}$, which shows that the statement is not in the language. Now, we can bound the probability of this event happening using a straightforward reduction to weak simulation-soundness of $\mathsf{AS}[\mathsf{TLF}]$. By Lemma 3, weak simulation-soundness is implied by soundness in conjunction with explainable zero-knowledge as $\varepsilon_{\mathsf{wssnd}} \leq \varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}}$. Thus, this event happens with probability at most $\varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}}$. And if this event does not happen, then $(\bar{\mathsf{pk}}^{(2)}, \bar{R}^{(2)}) = (\mathsf{pk}^{(2)}, R^{(2)})$, i.e., which implies that $\bar{h}$ was programmed as $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho] := \mathsf{GetChal}(sid, i, \mathcal{M}_1)$ when $(sid, i, \mathcal{M}_1)$ was removed from $\mathsf{Pending}$ (see $\mathbf{G}_8$ and $\mathbf{G}_9$).

(2) The value $\mathsf{GetChal}(sid, i, \mathcal{M}_1)$ has changed over time and will not match $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho]$ anymore. However, this event happens only with probability at most $Q_S^2(Q_S + t)/|\mathcal{R}|$ [BLT+24].

Overall, we get

$$|\Pr[\mathbf{G}_9 \Rightarrow 1] - \Pr[\mathbf{G}_{10} \Rightarrow 1]| \leq \frac{Q_S^2(Q_S + t)}{|\mathcal{R}|} + \varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}}.$$

Game $\mathbf{G}_{11}$: We no longer assume that $(\mathsf{par}', g) \in \mathsf{Reg}$. Clearly, we have

$$|\Pr[\mathbf{G}_{10} \Rightarrow 1] - \Pr[\mathbf{G}_{11} \Rightarrow 1]| \leq \varepsilon_{\mathsf{r}}.$$

It remains to bound the probability that the final game $\mathbf{G}_{11}$ outputs 1. At this stage, observe that we no longer need secret key shares sk_i to simulate signing for honest parties, and sk_i is only needed upon corruptions.

Final Claim. We now conclude the proof by claiming that

$$\Pr[\mathbf{G}_{11} \Rightarrow 1] \leq \frac{Q_{\bar{\mathsf{H}}}}{|\mathcal{S}|} + \mathsf{Adv}_{\mathcal{B},\mathsf{TLF}}^{t\text{-}\mathsf{A}\text{-}\mathsf{TRAN}\text{-}\mathsf{RES}}(\lambda).$$

To prove the claim, we make a case distinction regarding the final forgery (m^*, σ^*) where $\sigma^* = (\mathsf{pk}^{*(2)}, c^*, s^*, \varrho^*)$. For this, denote $h^* := \mathsf{H}(\mathsf{m}^*, \varrho^*)$.

- There is no $x_0 \in \mathcal{D}$ such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$. We can bound the probability of this event using Lemma 5. For this, we build a reduction $\mathcal{I}$ that runs in the game defined in Lemma 5 and succeeds (if some guess of it was correct). The reduction $\mathcal{I}$ gets as input parameters

par' for TLF. Then, it samples a random $i^* \xleftarrow{\$} [Q_{\bar{\mathsf{H}}}]$ and simulates game $\mathbf{G}_{11}$ for $\mathcal{A}$ with the following change: Upon the i^*-th query $x^* := (\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$ to random oracle $\bar{\mathsf{H}}$, $\mathcal{I}$ outputs its state along with

$$g,\ h := \mathsf{H}(\mathsf{m}, \varrho),\ X_1 := \mathsf{pk},\ X_2 := \mathsf{pk}^{(2)},\ R_1 := R^{(1)},\ R_2 := R^{(2)}$$

to the game and obtains a challenge $c \in \mathcal{S}$. If the hash value $\bar{\mathsf{H}}(x^*)$ is already defined, the reduction aborts. Otherwise, it programs $\bar{\mathsf{H}}(x^*) := c$ and continues the simulation. At a later stage, when $\mathcal{A}$ outputs its forgery, the reduction aborts if the query defining c^* was not x^*. Otherwise, it outputs s^* to the game. Note that the random oracle is never reprogrammed at that position, as $\mathcal{A}$ is not allowed to make a signing query for m^*. Further, it is clear that the reduction is successful against the game from Lemma 5 if the guess i^* was correct. As the view of $\mathcal{A}$ in its interaction with the reduction is independent of i^*, we can bound the probability of this case by $Q_{\bar{\mathsf{H}}}/|\mathcal{S}|$.

- There is an $x_0 \in \mathcal{D}$ such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$. We can bound the probability of this event using an efficient reduction $\mathcal{B}$ against the t-algebraic translation resistance of TLF. For this, $\mathcal{B}$ gets as input parameters par' for TLF, tags g, h, and images $X_0, \ldots, X_t \in \mathcal{R}$. Then, it simulates game $\mathbf{G}_{11}$ for $\mathcal{A}$ with the following changes:
  - **Key Setup.** It sets $\mathsf{pk}_i := X_i$ for all $i \in [\![t]\!]$. In particular, the joint public key is $\mathsf{pk} = X_0$. Further, it sets $\mathsf{pk}_i := \sum_{j \in [\![t]\!]} \ell_{j,S_0}(i)\mathsf{pk}_j$ for all $i \in [n] \setminus [\![t]\!]$. Clearly, the public keys have the same distribution as in $\mathbf{G}_{11}$.
  - **Target Tag.** It sets $h^\diamond := h$ for the randomly sampled tag introduced from game $\mathbf{G}_3$ on.
  - **Corruptions.** Whenever $\mathcal{A}$ queries the corruption oracle on some honest party i, $\mathcal{B}$ queries $x_i := \mathtt{Inv}(\ell_{0,S_0}(i), \ldots, \ell_{t,S_0}(i))$ and returns $\mathsf{sk}_i := x_i$. Clearly, the secret key sk_i is correctly distributed. Further, $\mathcal{B}$ queries the inversion oracle exactly t times, as $\mathcal{A}$ corrupts exactly t parties during the interaction. We reiterate that this allows to compute the other values in the corruption oracle as well. Concretely, the nonce r_i for a session sid can be computed as $r_i := s_i - \bar{c} \cdot \ell_{i,S} \cdot \mathsf{sk}_i$, where s_i is the signature share and $\bar{c}$ is derived as in $\mathbf{G}_{10}$. To obtain the random coins for simulated proofs, the reduction runs the zero-knowledge simulator Sim on the witness $\mathsf{witn}_i := (r_i, \mathsf{sk}_i)$ and a previously by Sim output state St when it was invoked to simulate proofs (as described in $\mathbf{G}_5$).

Finally, when $\mathcal{A}$ outputs its forgery (m^*, σ^*) and game $\mathbf{G}_{11}$ outputs 1 (which $\mathcal{B}$ can check efficiently), the reduction $\mathcal{B}$ takes $\mathsf{td}^* := tr[\mathsf{m}^*, \varrho^*]$ and computes the inverse $X_0' := \mathsf{InvTranslate}(\mathsf{td}^*, \mathsf{pk}^{*(2)})$ for the tag $h^* := \mathsf{H}(\mathsf{m}^*, \varrho^*)$. Recall that $(h^*, \mathsf{td}^*) \leftarrow \mathsf{Shift}(\mathsf{par}', h)$ because of the changes in $\mathbf{G}_3$. Then, $\mathcal{B}$ computes

$$X_i' := \mathsf{T}(h, x_i)\ \forall i \in \mathsf{Corrupted}, \quad X_i' := \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot X_j'\ \forall i \in [n] \setminus \mathsf{Corrupted},$$

where $C^* := \mathsf{Corrupted} \cup \{0\}$ (note that this set has size $t+1$ by game $\mathbf{G}_0$), and outputs $(X_0', \ldots, X_t')$ to the t-algebraic translation resistance game. We argue that this indeed defines a valid solution to the game: i.e., for all $i \in [\![t]\!]$, there is a $z_i \in \mathcal{D}$ such that $\mathsf{T}(g, z_i) = X_i$ and $\mathsf{T}(h, z_i) = X_i'$. Clearly, this is true for all $i \in \mathsf{Corrupted}$ by construction. Further, we know by assumption that there is an $x_0 \in \mathcal{D}$ such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$. Thus, by translation completeness of TLF, it follows that $X_0' = \mathsf{InvTranslate}(\mathsf{td}^*, \mathsf{pk}^{*(2)}) = \mathsf{T}(h, x_0)$. Further, for all $i \in [t] \setminus C^*$, we have

$$X_i' = \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot X_j' = \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot \mathsf{T}(h, x_i) = \mathsf{T}\left(h, \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot x_i\right).$$

On the other hand, we have

$$\mathsf{T}\left(g, \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot x_i\right) = \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot \mathsf{T}(g, x_i) = \sum_{j \in C^*} \ell_{j,C^*}(i) \cdot \mathsf{pk}_i' = X_i'.$$

This shows our claim and we conclude the proof.

□

6 Instantiations and Efficiency

We have presented our scheme using the abstraction of tagged linear function families introduced in [BLT+24]. In particular, we can use the instantiations of tagged linear functions provided in [BLT+24].

Instantiations. We can either instantiate linear functions based on an interactive assumption, or with minimal overhead from a non-interactive assumption. Both instantiations rely on a cyclic group $\mathbb{G}$ of prime order p and have the following characteristics:

- *Based On* AOMCDH. One can instantiate the function based on an algebraic variant of one-more CDH. In this case, we have $\mathcal{S} = \mathbb{Z}_p$, $\mathcal{T} = \mathbb{G}$, $\mathcal{D} = \mathbb{Z}_p$, and $\mathcal{R} = \mathbb{G}$. We denote the resulting threshold signature scheme by $\mathsf{Twinkle\text{-}T}_{\mathsf{AOMCDH}}$.

- *Based On* DDH. One can instantiate the function based on the DDH assumption, which is *non-interactive*. In this case, we have $\mathcal{S} = \mathbb{Z}_p$, $\mathcal{T} = \mathbb{G}^{2\times 2}$, $\mathcal{D} = \mathbb{Z}_p^2$, and $\mathcal{R} = \mathbb{G}^2$. We denote the resulting threshold signature scheme by $\mathsf{Twinkle\text{-}T}_{\mathsf{DDH}}$.

It has been shown in [BLT+24] that these instantiations are correct and satisfy regularity, translatability, and algebraic translation resistance. Importantly, algebraic translation resistance is *tightly* reduced to AOMCDH and DDH, respectively. We provide more details in Appendix B.

Efficiency. In Table 2, we compare the efficiency and concrete security level of our schemes with previous

Scheme	PK	SK	Sig	Comm	Sec
Frost [KG20]	33	33	64	98	43
Frost2 [BCK+22]/Frost3 [RRJ+22]	33	33	64	98	43
TZ [TZ23]	33	33	97	130	42
Classic-S [Mak22]	33	33	64	97	43
Sparkle [CKM23b]	33	33	64	97	43
Sparkle+ [CKM23a]	33	66	64	161	43
Zero-S [Mak22]	33	330	64	129	43
KRT [KRT24]	33	660	64	194	3
$\mathsf{Twinkle}_{\mathsf{AOMCDH}}$ [BLT+24]	33	33	97	163	85
$\mathsf{Twinkle}_{\mathsf{DDH}}$ [BLT+24]	66	66	162	294	85
$\mathsf{Twinkle\text{-}T}_{\mathsf{AOMCDH}}$ (ours)	33	33	129	259	127
$\mathsf{Twinkle\text{-}T}_{\mathsf{DDH}}$ (ours)	66	66	194	422	127

Table 2: Concrete efficiency and security of threshold signature schemes in pairing-free cyclic groups. We compare the size of public keys, size of secret keys per signer, signature size, communication cost per signer, and concrete security - all sizes are given in bytes. In KRT, each signer has $2n + 2$ secret keys where n is the total number of signers. In Zero-S, each signer has $n + 1$ secret keys, since it assumes point-to-point secret channels. We take $n = 9$ for our comparison. In Sparkle+, each signer has two secret keys, since it makes use of an additional standard signature scheme (we take standard Schnorr for that), which KRT also does. We omit Lindell from our comparison, since it uses online-extractable NIZK proofs [Fis05]. To compute the concrete security, we take the security bounds stated in the papers, assume $Q = 2^{40}$ for the number of hash and signing queries each, and assume that the underlying assumption is 128-bit hard.

constructions. For the comparison, we assume that all schemes are instantiated with the secp256k1 curve and the SHA-256 hash function. Further, we assume that challenges c_i are sampled uniformly from $\mathbb{Z}_p$ and have 256 bit-length, although some implementations may use challenges of 128 bit-length. For concrete security, we assume the underlying assumption is 128-bit hard and that the number of hash and signing queries each is $Q = 2^{40}$. Overall, our comparison indicates that our constructions only add a slight overhead in the communication cost and signature size compared to the state-of-the-art fully adaptive threshold signature Twinkle. As such, our schemes still remain highly practical and simple. In particular, given the strong security guarantees that our schemes provide, this minor efficiency overhead is acceptable.

References

[AB21] Handan Kilinç Alper and Jeffrey Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, August 2021. Springer, Cham. (Cited on Page 5.)

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

[AF04] Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Berlin, Heidelberg, August 2004. (Cited on Page 4, 5.)

[AFLT12] Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly-secure signatures from lossy identification schemes. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 572–590. Springer, Berlin, Heidelberg, April 2012. (Cited on Page 5.)

[AHN$^+$23] Masayuki Abe, Dennis Hofheinz, Ryo Nishimaki, Miyako Ohkubo, and Pan Jiaxin. Compact structure-preserving signatures with almost tight security. *Journal of Cryptology*, 36, 08 2023. (Cited on Page 5.)

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008. (Cited on Page 5.)

[BCK$^+$22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022. (Cited on Page 3, 4, 11, 24.)

[BD21] Mihir Bellare and Wei Dai. Chain reductions for multi-signatures and the HBMS scheme. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 650–678. Springer, Cham, December 2021. (Cited on Page 5.)

[BDL19] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 607–636. Springer, Cham, December 2019. (Cited on Page 3.)

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Cham, December 2018. (Cited on Page 5.)

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. (Cited on Page 13.)

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, Cham, December 2021. (Cited on Page 38.)

[BHK$^+$24] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: High-throughput robust distributed Schnorr signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 62–91. Springer, Cham, May 2024. (Cited on Page 4.)

[BJ08] Ali Bagherzandi and Stanislaw Jarecki. Multisignatures using proofs of secret key possession, as secure as the Diffie-Hellman problem. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 218–235. Springer, Berlin, Heidelberg, September 2008. (Cited on Page 5.)

[BJ10] Ali Bagherzandi and Stanislaw Jarecki. Identity-based aggregate and multi-signature schemes based on RSA. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 480–498. Springer, Berlin, Heidelberg, May 2010. (Cited on Page 5.)

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

[BLSW24] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. HARTS: High-threshold, adaptively secure, and robust threshold schnorr signatures. Cryptology ePrint Archive, Paper 2024/280, 2024. (Cited on Page 5, 9, 18.)

[BLT$^+$24] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 429–459. Springer, Cham, May 2024. (Cited on Page 3, 4, 5, 6, 7, 10, 11, 12, 19, 20, 21, 22, 24, 35, 38, 39.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006. (Cited on Page 3, 5.)

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422. Springer, Berlin, Heidelberg, July 2007. (Cited on Page 5.)

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003. (Cited on Page 5.)

[BP22] 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`, 2022. Accessed: 2024-09-17. (Cited on Page 3.)

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Berlin, Heidelberg, May 1996. (Cited on Page 7.)

[BTT22] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 276–305. Springer, Cham, August 2022. (Cited on Page 5.)

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. (Cited on Page 3, 4, 11.)

[BW24] Renas Bacho and Benedikt Wagner. Tightly secure non-interactive BLS multi-signatures. Cryptology ePrint Archive, Paper 2024/1368, 2024. (Cited on Page 5.)

[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. (Cited on Page 5.)

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

[CGJ$^+$99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Berlin, Heidelberg, August 1999. (Cited on Page 4, 5.)

[CGRS23] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023. (Cited on Page 4.)

[CKM21] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. (Cited on Page 3, 4.)

[CKM23a] Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive schnorr threshold signatures. Cryptology ePrint Archive, Paper 2023/445, 2023. (Cited on Page 3, 4, 5, 24.)

[CKM23b] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023. (Cited on Page 3, 4, 5, 24.)

[CKM$^+$23c] Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 710–742. Springer, Cham, August 2023. (Cited on Page 5.)

[CKP$^+$23] Elizabeth C. Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 348–382. Springer, Singapore, December 2023. (Cited on Page 5.)

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Berlin, Heidelberg, August 2000. (Cited on Page 5.)

[DEF$^+$19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019. (Cited on Page 5, 8.)

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Berlin, Heidelberg, August 1988. (Cited on Page 3.)

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990. (Cited on Page 3.)

[DOTT21] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 99–130. Springer, Cham, May 2021. (Cited on Page 8.)

[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. (Cited on Page 5.)

[DR24] Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 251–284. Springer, Cham, August 2024. (Cited on Page 5.)

[EHK$^+$13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Berlin, Heidelberg, August 2013. (Cited on Page 39.)

[EKP20] Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy CSI-FiSh: Efficient signature scheme with tight reduction to decisional CSIDH-512. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 157–186. Springer, Cham, May 2020. (Cited on Page 5.)

[EKT24] Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 387–424. Springer, Cham, August 2024. (Cited on Page 5.)

[FH20] Masayuki Fukumitsu and Shingo Hasegawa. A tightly secure ddh-based multisignature with public-key aggregation. In *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 321–327, 2020. (Cited on Page 5.)

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005. (Cited on Page 24.)

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. (Cited on Page 3.)

[FMY99] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure distributed public-key systems. In Jaroslav Nešetřil, editor, *Algorithms - ESA' 99*, pages 4–27, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. (Cited on Page 5.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. (Cited on Page 14.)

[FSZ22] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1109–1123. ACM Press, November 2022. (Cited on Page 5.)

[GHKP18] Romain Gay, Dennis Hofheinz, Lisa Kohl, and Jiaxin Pan. More efficient (almost) tightly secure structure-preserving signatures. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 230–258. Springer, Cham, April / May 2018. (Cited on Page 5.)

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007. (Cited on Page 4.)

[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. (Cited on Page 5.)

[GS24] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 370–400. Springer, Cham, May 2024. (Cited on Page 4.)

[HJ12] Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 590–607. Springer, Berlin, Heidelberg, August 2012. (Cited on Page 5.)

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 66–83. Springer, Berlin, Heidelberg, May 2012. (Cited on Page 5.)

[HJP18] Dennis Hofheinz, Dingding Jia, and Jiaxin Pan. Identity-based encryption tightly secure under chosen-ciphertext attacks. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 190–220. Springer, Cham, December 2018. (Cited on Page 5.)

[HK22] Lucjan Hanzlik and Kamil Kluczniak. Explainable arguments. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 59–79. Springer, Cham, May 2022. (Cited on Page 9, 14, 32.)

[HKK$^+$24] Goichiro Hanaoka, Shuichi Katsumata, Kei Kimura, Kaoru Takemure, and Shota Yamada. Tighter adaptive IBEs and VRFs: Revisiting waters' artificial abort. Cryptology ePrint Archive, Paper 2024/1481, 2024. (Cited on Page 5.)

[HKS15] Dennis Hofheinz, Jessica Koch, and Christoph Striecks. Identity-based encryption with (almost) tight security in the multi-instance, multi-ciphertext setting. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 799–822. Springer, Berlin, Heidelberg, March / April 2015. (Cited on Page 5.)

[IN83] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983. (Cited on Page 5.)

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

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020. (Cited on Page 3, 4, 24.)

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 537–553. Springer, Berlin, Heidelberg, April 2012. (Cited on Page 5.)

[KRB$^+$24] Ioanna Karantaidou, Omar Renawi, Foteini Baldimtsi, Nikolaos Kamarinakis, Jonathan Katz, and Julian Loss. Blind multisignatures for anonymous tokens with decentralized issuance. Cryptology ePrint Archive, Paper 2024/1406, 2024. (Cited on Page 5.)

[KRT24] Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024. (Cited on Page 3, 4, 5, 24.)

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, October 2003. (Cited on Page 5, 7.)

[KYY18] Shuichi Katsumata, Shota Yamada, and Takashi Yamakawa. Tighter security proofs for GPV-IBE in the quantum random oracle model. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 253–282. Springer, Cham, December 2018. (Cited on Page 5.)

[Lin24] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *CiC*, 1(1):25, 2024. (Cited on Page 3, 4.)

[LJY14] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014. (Cited on Page 5.)

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Berlin, Heidelberg, May / June 2006. (Cited on Page 5.)

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

[Mak22] Nikolaos Makriyannis. On the classic protocol for MPC schnorr signatures. Cryptology ePrint Archive, Paper 2022/1332, 2022. (Cited on Page 4, 5, 11, 24.)

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001. (Cited on Page 5.)

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptography*, 87(9):2139–2164, September 2019. (Cited on Page 5.)

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Cham. (Cited on Page 5, 8, 38.)

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020. (Cited on Page 5.)

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Berlin, Heidelberg, April 1991. (Cited on Page 3.)

[PW21] Jiaxin Pan and Benedikt Wagner. Short identity-based signatures with tight security from lattices. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 360–379, Cham, 2021. Springer International Publishing. (Cited on Page 5.)

[PW22] Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 347–378. Springer, Cham, March 2022. (Cited on Page 5.)

[PW23] Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 597–627. Springer, Cham, April 2023. (Cited on Page 5, 7, 8.)

[PW24] Jiaxin Pan and Benedikt Wagner. Toothpicks: More efficient fork-free two-round multi-signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 460–489. Springer, Cham, May 2024. (Cited on Page 5, 7, 8.)

[QLH12] Haifeng Qian, Xiangxue Li, and Xinli Huang. Tightly secure non-interactive multisignatures in the plain public key model. *Informatica (Vilnius)*, 3, 01 2012. (Cited on Page 5.)

[RRJ$^+$22] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022. (Cited on Page 4, 24.)

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Berlin, Heidelberg, May 2007. (Cited on Page 5.)

[Seu12] Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, April 2012. (Cited on Page 4, 5.)

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997. (Cited on Page 38.)

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Berlin, Heidelberg, July 2001. (Cited on Page 4.)

[TZ22] Stefano Tessaro and Chenzhi Zhu. Short pairing-free blind signatures with exponential security. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 782–811. Springer, Cham, May / June 2022. (Cited on Page 3.)

[TZ23] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023. (Cited on Page 3, 4, 5, 24.)

[WQL09] Zecheng Wang, Haifeng Qian, and Zhibin Li. Adaptively secure threshold signature scheme in the standard model. *Informatica*, 20(4):591–612, dec 2009. (Cited on Page 5.)

Appendix

A More Details on Non-Interactive Argument Systems

In this section, we give formal definitions and proofs for non-interactive argument systems.

A.1 Formal Definitions

Here, we define non-interactive argument systems for **NP** relations. We focus on argument systems in the random oracle model. For all definitions related to argument systems, we do not explicitly define an advantage of the form $\mathsf{Adv}^{\cdot}_{\cdot}(\lambda)$, but rather use $\varepsilon_{\mathsf{xzk}}$ to convey that for our instantiation this will hold for *any potentially unbounded* adversary, assuming only *polynomially many* random oracle queries. Before we can give the definition of an argument systems, we first formally define what we mean by an **NP** relation.

Definition 7 (**NP** Relation)**.** Let $\mathcal{R}$ be a relation containing pairs $(\mathsf{stmt}, \mathsf{witn})$, which is implicitly parameterized by the security parameter λ. We say that $\mathcal{R}$ is an **NP** relation and call stmt the statement and witn the witness, if the following hold:

- *Size Constraints.* There are polynomials poly_1 and poly_2 such that for all $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$, we have $|\mathsf{stmt}| \leq \mathsf{poly}_1(\lambda)$ and $|\mathsf{witn}| \leq \mathsf{poly}_2(|\mathsf{stmt}|)$.
- *Efficiently Decidable.* The relation can be decided deterministically in polynomial time, i.e., there is a deterministic polynomial time algorithm that takes as input a pair $(\mathsf{stmt}, \mathsf{witn})$ and outputs 1 if and only if $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$.

For such a relation, we define the induced language $\mathcal{L}_{\mathcal{R}} \subseteq \{0,1\}^*$ by

$$\mathcal{L}_{\mathcal{R}} := \left\{\mathsf{stmt} \in \{0,1\}^* \mid \exists \mathsf{witn} \in \{0,1\}^* : (\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}\right\}.$$

Definition 8 (Non-Interactive Argument System)**.** Let $\mathcal{R}$ be an **NP** relation and H be a random oracle. A non-interactive argument system for $\mathcal{R}$ with respect to $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ is defined to be a pair $\mathsf{AS} = (\mathsf{Prove}, \mathsf{VerProof})$ of PPT algorithms with oracle access to H and the following syntax:

- $\mathsf{Prove}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn}) \to \pi$ takes as input a statement stmt and a witness witn, and outputs a proof π. We assume that the random coins are uniform over $\{0,1\}^{\ell(\lambda)}$, i.e., we can write $\pi := \mathsf{Prove}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn}; \rho)$ for $\rho \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$.
- $\mathsf{VerProof}^{\mathsf{H}}(\mathsf{stmt}, \pi) \to b$ is deterministic, takes as input a statement stmt and a proof π, and outputs a decision bit $b \in \{0,1\}$.

Further, we require that the argument system is complete in the following sense: For any pair $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}$, we have

$$\Pr\left[\mathsf{VerProof}^{\mathsf{H}}(\mathsf{stmt}, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn})\right] = 1,$$

where the probability is taken over the random coins of Prove and the random oracle H, which may be queried by external algorithms in an arbitrary way.

We need non-interactive arguments that are zero-knowledge, i.e., there is an efficient simulator that can generate proofs without access to the witness, which is typically accomplished by programming the random oracle. Note, however, that standard zero-knowledge is not enough in the context of adaptive corruptions: say the reduction simulates a proof on behalf of some honest party. When the adversary later corrupts this honest party, it would expect to learn the witness and well-distributed random coins that have been used to generate the proof from the witness. Therefore, we need a stronger form of zero-knowledge, in which the simulator can create appropriate random coins once it learns the witness. A similar notion called *explainable arguments* has been introduced by Hanzlik and Kluczniak [HK22]. We call our definition explainable zero-knowledge. In our definition, the adversary runs in one of two games, and has to say in which game it runs. In the first game, it gets access to honestly generated proofs, for which it can later ask the game to reveal the random coins that have been used. In the second game, proofs are provided by the simulator without access to the witness. When the adversary wants to learn

```
Game X-ZK$^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim},b}(\lambda)$
01 $b' \leftarrow \mathcal{A}^{\texttt{GetProof}_b,\texttt{GetCoins}_b,\mathsf{H}}(1^\lambda)$
02 return $b'$

Oracle $\texttt{GetProof}_0(pid, \mathsf{stmt}, \mathsf{witn})$
03 if $St[pid] \neq \bot$ : return $\bot$
04 if $(\mathsf{stmt}, \mathsf{witn}) \notin \mathcal{R}$ : return $\bot$
05 $\rho \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$, $St[pid] := \rho$
06 $\pi := \mathsf{Prove}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn}; \rho)$
07 return $\pi$

Oracle $\texttt{GetCoins}_0(pid)$
08 $\rho := St[pid]$
09 return $\rho$

Oracle $\mathsf{H}(x)$
10 if $h[x] = \bot$ : $h \xleftarrow{\$} \mathcal{Y}$
11 return $h[x]$

Oracle $\texttt{GetProof}_1(pid, \mathsf{stmt}, \mathsf{witn})$
12 if $St[pid] \neq \bot$ : return $\bot$
13 if $(\mathsf{stmt}, \mathsf{witn}) \notin \mathcal{R}$ : return $\bot$
14 $(\pi, St, h'[\cdot]) \leftarrow \mathsf{Sim}(0, h[\cdot], \mathsf{stmt})$
15 $St[pid] := (St, \mathsf{witn})$, $h[\cdot] := h'[\cdot]$
16 return $\pi$

Oracle $\texttt{GetCoins}_1(pid)$
17 $(St, \mathsf{witn}) := St[pid]$
18 return $\rho := \mathsf{Sim}(1, h[\cdot], St, \mathsf{witn})$
```

Figure 5: The games **X-ZK** modeling explainable zero-knowledge for a non-interactive argument system $\mathsf{AS} = (\mathsf{Prove}, \mathsf{VerProof})$ for an **NP** relation $\mathcal{R}$ with respect to a random oracle $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ and an adversary $\mathcal{A}$. Note that the simulator Sim can update the random oracle in oracle $\texttt{GetProof}_1$ by outputting a new map h'.

random coins, the witness is given to the simulator, and the simulator is expected to generate the random coins. To formalize that the simulator can program the random oracle, we allow the simulator to output a new map that lazily implements the random oracle.

Definition 9 (Explainable Zero-Knowledge)**.** Let $\mathcal{R}$ be an **NP** relation and $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ be a random oracle. Let AS be a non-interactive argument system for $\mathcal{R}$. We say that AS satisfies $\varepsilon_{\mathsf{xzk}}$-explainable zero-knowledge, if there is a PPT algorithm Sim (called the simulator) such that for any algorithm $\mathcal{A}$, we have

$$\left|\Pr\left[\textbf{X-ZK}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim},0}(\lambda) \Rightarrow 1\right] - \Pr\left[\textbf{X-ZK}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim},1}(\lambda) \Rightarrow 1\right]\right| \leq \varepsilon_{\mathsf{xzk}},$$

where the games $\textbf{X-ZK}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim},b}(\lambda)$ are defined in Figure 5. The term $\varepsilon_{\mathsf{xzk}}$ is allowed to depend on the number of random oracle queries $\mathcal{A}$ makes.

Next, we define a weak form of simulation-soundness in presence of explained proofs. That is, in our simulation-soundness game, the adversary gets access to simulated proofs and random coins that are output by the simulator, similar to the second game of explainable zero-knowledge. Then, the goal of the adversary is to output a proof that verifies for a wrong statement. In our definition, the adversary does not get access to simulated proofs for invalid statements.

```
Game W-SIM-SND$^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda)$
01 $\mathcal{A}^{\texttt{GetProof}_1,\texttt{GetCoins}_1,\mathsf{H},\texttt{ForgeProof}}(1^\lambda)$
02 return win

Oracle $\texttt{ForgeProof}(\mathsf{stmt}, \pi)$
03 if $\mathsf{stmt} \in \mathcal{L}_{\mathcal{R}}$ : return
04 if $\mathsf{VerProof}^{\mathsf{H}}(\mathsf{stmt}, \pi) = 1$ : win := 1
```

Figure 6: The game **W-SIM-SND** modeling weak simulation-soundness for a non-interactive argument system $\mathsf{AS} = (\mathsf{Prove}, \mathsf{VerProof})$ for an **NP** relation $\mathcal{R}$ with respect to a random oracle $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ and an adversary $\mathcal{A}$. Oracles $\texttt{GetProof}_1, \texttt{GetCoins}_1, \mathsf{H}$ are as in Figure 5.

Definition 10 (Weak Simulation-Soundness)**.** Let $\mathcal{R}$ be an **NP** relation and $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ be a random oracle. Let AS be a non-interactive argument system for $\mathcal{R}$ and Sim be a PPT algorithm. Then, we say that AS satisfies $\varepsilon_{\mathsf{wssnd}}$-weak simulation-soundness with respect to Sim, if for any algorithm $\mathcal{A}$, we have

$$\Pr\left[\textbf{W-SIM-SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda) \Rightarrow 1\right] \leq \varepsilon_{\mathsf{wssnd}},$$

where the game $\textbf{W-SIM-SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda)$ is defined in Figure 6. The term $\varepsilon_{\mathsf{wssnd}}$ is allowed to depend on the number of random oracle queries $\mathcal{A}$ makes.

As said before, we do not give the adversary access to simulated proofs for invalid statements. It is well-known that in this case, simulation-soundness follows from standard soundness, as defined next, and zero-knowledge. We show this for completeness in Lemma 3.

Definition 11 (Soundness)**.** Let $\mathcal{R}$ be an **NP** relation and $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ be a random oracle. Let AS be a non-interactive argument system for **NP**. Then, we say that AS satisfies $\varepsilon_{\mathsf{snd}}$-soundness, if for any algorithm $\mathcal{A}$, we have

$$\Pr\left[\mathsf{VerProof}^{\mathsf{H}}(\mathsf{stmt}, \pi) = 1 \wedge \mathsf{stmt} \notin \mathcal{L}_{\mathcal{R}} \mid (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(1^{\lambda})\right] \leq \varepsilon_{\mathsf{snd}}.$$

The term $\varepsilon_{\mathsf{snd}}$ is allowed to depend on the number of random oracle queries $\mathcal{A}$ makes.

Lemma 3. *Let $\mathcal{R}$ be an* **NP** *relation and $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$ be a random oracle. Let AS be a non-interactive argument system for $\mathcal{R}$ and Sim be a PPT algorithm. Assume that AS satisfies $\varepsilon_{\mathsf{snd}}$-soundness and that it satisfies $\varepsilon_{\mathsf{xzk}}$-explainable zero-knowledge with simulator Sim. Then, it satisfies $\varepsilon_{\mathsf{wssnd}}$-weak simulation-soundness with respect to Sim, where*

$$\varepsilon_{\mathsf{wssnd}} \leq \varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}}.$$

Proof. Let $\mathcal{A}$ be any algorithm. Our goal is to show that

$$\Pr\left[\mathbf{W\text{-}SIM\text{-}SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda) \Rightarrow 1\right] \leq \varepsilon_{\mathsf{xzk}} + \varepsilon_{\mathsf{snd}}.$$

Recall that in $\mathbf{W\text{-}SIM\text{-}SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda)$, the adversary $\mathcal{A}$ gets access to oracles $\mathtt{GetProof}_1, \mathtt{GetCoins}_1, \mathsf{H}$, $\mathtt{ForgeProof}$, and the game outputs the variable win (see Figure 6). Now, consider a game $\mathbf{G}$, which is exactly as experiment $\mathbf{W\text{-}SIM\text{-}SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda)$, but the oracles $\mathtt{GetProof}_1, \mathtt{GetCoins}_1$ are replaced by the oracles $\mathtt{GetProof}_0, \mathtt{GetCoins}_0$ given in Figure 5. We claim that it follows from $\varepsilon_{\mathsf{xzk}}$-explainable zero-knowledge that

$$\left|\Pr\left[\mathbf{W\text{-}SIM\text{-}SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda) \Rightarrow 1\right] - \Pr\left[\mathbf{G} \Rightarrow 1\right]\right| \leq \varepsilon_{\mathsf{xzk}}.$$

To see why this holds, consider the following (potentially inefficient reduction), running in $\mathbf{X\text{-}ZK}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim},b}(\lambda)$ for some $b \in \{0, 1\}$:

- The reduction gets access to oracles $\mathtt{GetProof}_b, \mathtt{GetCoins}_b, \mathsf{H}$.
- The reduction runs $\mathcal{A}$ on oracles $\mathtt{GetProof}_b, \mathtt{GetCoins}_b, \mathsf{H}, \mathtt{ForgeProof}$. It simulates $\mathtt{ForgeProof}$ and sets win accordingly via brute-force, but without making any additional queries to H.
- Finally, the reduction outputs win.

Clearly, if $b = 0$, the reduction perfectly simulates $\mathbf{G}$ for $\mathcal{A}$, whereas if $b = 1$, it perfectly simulates $\mathbf{W\text{-}SIM\text{-}SND}^{\mathcal{A}}_{\mathsf{AS},\mathsf{Sim}}(\lambda)$. As the number of random oracle queries does not increase, the claim follows. Hence, it remains to bound the probability that $\mathbf{G}$ outputs 1. This can easily be done via an (again, inefficient) reduction breaking soundness. This reduction is as follows:

- The reduction runs in the soundness game, i.e., it has access to a random oracle H.
- The reduction simulates game $\mathbf{G}$ honestly for $\mathcal{A}$. Note that in $\mathbf{G}$, the random oracle is never programmed.
- Simulation of oracle $\mathtt{ForgeProof}$ is potentially inefficient but does not increase the number of random oracle queries.
- When $\mathcal{A}$ makes the first query $\mathtt{ForgeProof}(\mathsf{stmt}, \pi)$ for which $\mathsf{win} = 1$ is set, the reduction terminates with output (stmt, π).

As the reduction makes as many random oracle queries as $\mathcal{A}$ and breaks soundness whenever $\mathbf{G}$ outputs 1, we get

$$\Pr\left[\mathbf{G} \Rightarrow 1\right] \leq \varepsilon_{\mathsf{snd}}.$$

The lemma follows. □

A.2 Proofs for Our Construction

Here, we prove Lemmata 1 and 2. Before we do that, we show a generic statistical lemma that will turn out to be useful. Intuitively, it states that the aggregation step of our argument system is sound. We also recall a useful statistical lemma taken from [BLT+24].

Lemma 4. *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family. For every fixed parameters* par *and tags* $g, h \in \mathcal{T}$*, define the set*

$$\mathsf{Im}(\mathsf{par}, g, h) := \left\{(X_1, X_2) \in \mathcal{R}^2 \;\middle|\; \exists x \in \mathcal{D}: \; \mathsf{T}(g, x) = X_1 \wedge \mathsf{T}(h, x) = X_2\right\}.$$

Then, for any (even unbounded) algorithm $\mathcal{A}$*, we have*

$$\Pr\left[(X \notin \mathcal{I} \vee R \notin \mathcal{I}) \wedge \bar{X} \in \mathcal{I} \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda), \\ (g, h, R, X) \leftarrow \mathcal{A}(\mathsf{par}), \\ \gamma \xleftarrow{\$} \mathcal{S}, \;\; \bar{X} := R + \gamma X \end{array}\right] \leq \frac{1}{|\mathcal{S}|},$$

where $\mathcal{I} := \mathsf{Im}(\mathsf{par}, g, h)$ *and* $R, X, \bar{X} \in \mathcal{R}^2$.

Proof. Consider the experiment in the statement, and define $\mathcal{I} := \mathsf{Im}(\mathsf{par}, g, h) \subseteq \mathcal{R}^2$. Then, it is easy to see that $\mathcal{I}$ is a subspace of the $\mathcal{S}$-vector space $\mathcal{R}^2$. Hence, there is a subspace $\mathcal{U} \subseteq \mathcal{R}^2$ such that $\mathcal{U} \oplus \mathcal{I} = \mathcal{R}^2$, i.e., any $W \in \mathcal{R}^2$ can be written as $W = U_W + I_W$ for some uniquely defined $U_W \in \mathcal{U}$ and $I_W \in \mathcal{I}$. Now, let $X, R, \bar{X}$ be as in the experiment. Then, write $X = U_X + I_X$ and $R = U_R + I_R$ with $U_X, U_R \in \mathcal{U}$ and $V_X, V_R \in \mathcal{I}$. With that, note that

$$\bar{X} = R + \gamma X = \underbrace{(U_R + \gamma U_X)}_{\in \mathcal{U}} + \underbrace{(I_R + \gamma I_X)}_{\in \mathcal{I}}.$$

In particular, the probability that we want to bound becomes

$$\Pr\left[\bar{X} \in \mathcal{I} \mid R \notin \mathcal{I} \vee X \notin \mathcal{I}\right] \leq \Pr_{\gamma}\left[U_R + \gamma U_X = 0 \mid U_R \neq 0 \vee U_X \neq 0\right] \leq \frac{1}{|\mathcal{S}|}.$$

□

Lemma 5 ([BLT+24])**.** *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family. For every fixed parameters* par *and tags* $g, h \in \mathcal{T}$*, define the set*

$$\mathsf{Im}(\mathsf{par}, g, h) := \left\{(X_1, X_2) \in \mathcal{R}^2 \;\middle|\; \exists x \in \mathcal{D}: \; \mathsf{T}(g, x) = X_1 \wedge \mathsf{T}(h, x) = X_2\right\}.$$

Then, for any (even unbounded) algorithm $\mathcal{A}$*, we have*

$$\Pr\left[\begin{array}{ll} & (X_1, X_2) \notin \mathsf{Im}(\mathsf{par}, g, h) \\ \wedge & \mathsf{T}(g, s) = c \cdot X_1 + R_1 \\ \wedge & \mathsf{T}(h, s) = c \cdot X_2 + R_2 \end{array} \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda), \\ (St, g, h, X_1, X_2, R_1, R_2) \leftarrow \mathcal{A}(\mathsf{par}), \\ c \xleftarrow{\$} \mathcal{S}, \;\; s \leftarrow \mathcal{A}(St, c) \end{array}\right] \leq \frac{1}{|\mathcal{S}|}.$$

Proof of Lemma 1. We prove explainable zero-knowledge of $\mathsf{AS}[\mathsf{TLF}]$. To this end, we give an efficient simulator Sim, which has two jobs:

- *Simulating Proofs.* On input $(0, h[\cdot], \mathsf{stmt})$, where $h[\cdot]$ is the map defining the current state of the random oracle $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$ and stmt is a valid statement, the simulator needs to simulate a proof π for this statement. It is allowed to program the random oracle by outputting a new map $h'[\cdot]$ that represents the new state of the random oracle.
- *Simulating Random Coins.* At some later point in time, the simulator may get input $(1, h[\cdot], St, \mathsf{witn})$, where again $h[\cdot]$ represents the random oracle, St is a state the simulator can output when simulating a proof, and witn is a witness for the statement for which it simulated the proof. The task of the simulator is to output well-distributed random coins ρ such that π could have been generated using the witness and these coins. In the case of $\mathsf{AS}[\mathsf{TLF}]$, the random coins are simply $w \xleftarrow{\$} \mathcal{D}$.

We now describe how the simulator works for our scheme $\mathsf{AS}[\mathsf{TLF}]$:

- $\mathsf{Sim}(0, h[\cdot], \mathsf{stmt}) \to (\pi, St, h'[\cdot])$:
    1. Parse the statement as $(h, R_1, R_2, X_1, X_2) := \mathsf{stmt}$.
    2. Set $\gamma := \mathsf{H}(0, \mathsf{stmt})$, which potentially updates map $h[\cdot]$.
    3. Define $\bar{X} := (R_1 + \gamma X_1, R_2 + \gamma X_2)$ and $\overline{\mathsf{stmt}} := (h, \bar{X})$.
    4. Sample $c \xleftarrow{\$} \mathcal{S}$ and $z \xleftarrow{\$} \mathcal{D}$ and set $W := \varphi_{g,h}(z) - c\bar{X}$.
    5. If $h[1, \overline{\mathsf{stmt}}, W] \neq \bot$, abort. Otherwise, set $h[1, \overline{\mathsf{stmt}}, W] := c$.
    6. Set $\pi := (c, z)$ and $St := (\gamma, c, z)$.
    7. Define $h'[\cdot]$ to be the map $h[\cdot]$ with all the changes so far.
    8. Return $(\pi, St, h'[\cdot])$.
- $\mathsf{Sim}(1, h[\cdot], St, \mathsf{witn}) \to \rho$:
    1. Parse $(\gamma, z, c) := St$ and $(r, x) := \mathsf{witn}$.
    2. Set $\overline{\mathsf{witn}} := \bar{x} := r + \gamma x$ and return $\rho := w := z - c\bar{x}$.

We now turn to the analysis of our simulator. First, it is clear that Sim is PPT. Second, we need to argue that for any algorithm $\mathcal{A}$, making at most Q_{H} queries to H and Q queries to the oracles `GetProof` and `GetCoins` and running either in the game with honestly generated proofs or with the simulator, the distinguishing advantage is bounded by $\varepsilon_{\mathsf{xzk}}$ as in the lemma. To this end, we first bound the probability that the simulator will ever abort. Namely, by regularity of TLF and the assumption that (par, g) is in the regularity set, we know that the first component of $\varphi_{g,h}(z)$ for $z \xleftarrow{\$} \mathcal{D}$ is uniformly distributed over $\mathcal{R}$, and so is the first component of $W = \varphi_{g,h}(z) - c\bar{X}$. Therefore, for any fixed invocation of the simulator, the probability that it aborts because $h[1, \overline{\mathsf{stmt}}, W]$ is already defined is at most $Q_{\mathsf{H}}/|\mathcal{R}|$. With a union bound over the at most Q invocations of the simulator, the probability of an abort is therefore bounded by $QQ_{\mathsf{H}}/|\mathcal{R}|$. Now, condition on no abort happening. In this case, note that the simulator performs the aggregation step (computing $\bar{X}$ from (R_1, R_2) and (X_1, X_2)) exactly as the honest prover algorithm would do. Therefore, it is sufficient to observe that for any $\bar{x} \in \mathcal{D}$, the following distributions are the same:

- *Real Distribution.* Output $(\bar{x}, w, c, z)$, where $w \xleftarrow{\$} \mathcal{D}, c \xleftarrow{\$} \mathcal{S}$, and $z := c\bar{x} + w$.
- *Simulated Distribution.* Output $(\bar{x}, w, c, z)$, where $z \xleftarrow{\$} \mathcal{D}, c \xleftarrow{\$} \mathcal{S}$, and $w := z - c\bar{x}$.

This is clear, concluding the proof. $\square$

Proof of Lemma 2. We prove soundness of $\mathsf{AS}[\mathsf{TLF}]$. To this end, consider the soundness experiment for an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ is assumed to have access to the parameters par, g specifying the relation $\mathcal{R}_{\mathsf{TLF}}[\mathsf{par}, g]$. It gets access to the random oracle $\mathsf{H}\colon \{0,1\}^* \to \mathcal{S}$ and the security parameter 1^λ as input. The adversary outputs a statement $\mathsf{stmt} = (h, R_1, R_2, X_1, X_2) \in \mathcal{T} \times \mathcal{R}^4$ and a proof $\pi = (c, z) \in \mathcal{S} \times \mathcal{D}$. The adversary wins the game if the statement is invalid, but the proof verifies, i.e., if the following hold:

1. We have $(X_1, X_2) \notin \mathsf{Im}(\mathsf{par}, g, h)$ or $(R_1, R_2) \notin \mathsf{Im}(\mathsf{par}, g, h)$, using the notation of Lemmata 4 and 5.
2. We have $\mathsf{H}(1, \overline{\mathsf{stmt}}, W) = c$ for $\gamma := \mathsf{H}(0, \mathsf{stmt})$, $\bar{X} := (R_1 + \gamma X_1, R_2 + \gamma X_2)$, $\overline{\mathsf{stmt}} := (h, \bar{X})$, and $W := \varphi_{g,h}(z) - c\bar{X}$.

We now consider the following events $\mathsf{BatchBreak}$:

- Event $\mathsf{BatchBreak}$: This event occurs, if $\bar{X} \in \mathsf{Im}(\mathsf{par}, g, h)$.
- Event Win: This event occurs, if $\mathcal{A}$ wins, as specified above. Our goal is to bound the probability of this event.

We can use Lemma 4 and a union bound over all Q_{H} random oracle queries to get

$$\Pr\left[\mathsf{BatchBreak}\right] \leq \frac{Q_{\mathsf{H}}}{|\mathcal{S}|}.$$

Now, condition on $\mathsf{BatchBreak}$ not occurring. Then in particular, we have that $\bar{X} \notin \mathsf{Im}(\mathsf{par}, g, h)$. We can therefore easily build a reduction that runs in the experiment in Lemma 5: it would guess the random oracle query $\mathsf{H}(1, \overline{\mathsf{stmt}}, W)$ associated with the adversaries output. When this query occurs, it would output its state and $g, h, \bar{X}, W$ to the experiment in Lemma 5 and program the random oracle to the challenge that it receives back. Finally, when $\mathcal{A}$ terminates and the guess was correct (which happens with probability $1/Q_{\mathsf{H}}$), it would output z to the experiment. By Lemma 5, we have

$$\Pr\left[\mathsf{Win}\right] \leq \Pr\left[\mathsf{BatchBreak}\right] + \Pr\left[\mathsf{Win} \mid \neg\mathsf{BatchBreak}\right] \leq \frac{Q_{\mathsf{H}}}{|\mathcal{S}|} + \frac{Q_{\mathsf{H}}}{|\mathcal{S}|} = \frac{2Q_{\mathsf{H}}}{|\mathcal{S}|}.$$

$\square$

B Details on Instantiations

Here, we provide more details on the instantiations of tagged linear function families, which are taken from [BLT+24].

B.1 Instantiation from Algebraic One-More CDH

We use the tagged linear function family $\mathsf{TLF}_{\mathsf{AOMCDH}} = (\mathsf{Gen}_{\mathsf{AOMCDH}}, \mathsf{T}_{\mathsf{AOMCDH}})$ from [BLT+24] based on the AOMCDH assumption. We first recall the AOMCDH assumption.

Definition 12 (AOMCDH Assumption)**.** Let GGen be a group generation algorithm. That is, on input 1^λ, it outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p and a generator g of $\mathbb{G}$, and a description of the group operation. Consider the game **AOMCDH** in Figure 7. We say that the t-AOMCDH assumption holds relative to GGen, if for all PPT algorithms $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{t\text{-}\mathsf{AOMCDH}}_{\mathcal{A},\mathsf{GGen}}(\lambda) := \Pr\left[t\text{-}\mathbf{AOMCDH}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda) \Rightarrow 1\right].$$

```
Game $t$-AOMCDH$^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$
$h \xleftarrow{\$} \mathbb{G},\ x_0, \dots, x_t \xleftarrow{\$} \mathbb{Z}_p$
for $i \in [\![t]\!] : X_i := g^{x_i}$
$(X'_i)_{i=0}^t \leftarrow \mathcal{A}^{\mathtt{Inv}}(\mathbb{G}, g, p, h, (X_i)_{i=0}^t)$
if $\forall i \in [\![t]\!]\ X'_i = h^{x_i}$ : return 1
return 0

Oracle $\mathtt{Inv}(\alpha_0, \dots, \alpha_t)$
if $q \geq t$ : return $\perp$
$q := q + 1$
$x := \sum_{i=0}^t \alpha_i x_i$
return $x$
```

Figure 7: The game **AOMCDH** from the definition of the AOMCDH assumption for an adversary $\mathcal{A}$.

The authors [BLT+24] show that this assumption is implied by the algebraic one-more discrete logarithm (AOMDL) assumption [NRS21] in the algebraic group model. Further, Bauer et al. [BFP21] give a bound for AOMDL in the generic group model (GGM) [Sho97]. As a direct implication, the advantage of any adversary against AOMCDH is bounded by the same probability, namely $\Theta\left(t^2/(p-t^2) + 1/p\right)$.

The tagged linear function family $\mathsf{TLF}_{\mathsf{AOMCDH}} = (\mathsf{Gen}_{\mathsf{AOMCDH}}, \mathsf{T}_{\mathsf{AOMCDH}})$ based on AOMCDH is defined as follows. Let GGen be an algorithm that takes as input 1^λ and outputs the description of a group $\mathbb{G}$ of prime order p with generator $g \in \mathbb{G}$. Algorithm $\mathsf{Gen}_{\mathsf{AOMCDH}}$ runs GGen and outputs parameters $\mathsf{par} = (\mathbb{G}, g, p)$. These parameters define the following sets of scalars, tags, and the domain and range, respectively:

$$\mathcal{S} := \mathbb{Z}_p, \quad \mathcal{T} := \mathbb{G}, \quad \mathcal{D} := \mathbb{Z}_p, \quad \mathcal{R} := \mathbb{G}.$$

Given a tag $h \in \mathbb{G}$ and an input element $x \in \mathbb{Z}_p$, the tagged linear function $\mathsf{T}_{\mathsf{AOMCDH}}$ is defined as

$$\mathsf{T}_{\mathsf{AOMCDH}}(h, x) := h^x \in \mathbb{G}.$$

Lemma 6 ([BLT+24])**.** *The tagged linear function family $\mathsf{TLF}_{\mathsf{AOMCDH}}$ is ε_{r}-regular and ε_{t}-translatable with $\varepsilon_{\mathsf{r}} \leq 1/p$ and $\varepsilon_{\mathsf{t}} \leq 2/p$. Further, let $t \in \mathbb{N}$ be polynomial in λ, and assume that the t-AOMCDH assumption holds relative to GGen. Then, $\mathsf{TLF}_{\mathsf{AOMCDH}}$ is t-algebraic translation resistant. Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{t\text{-}\mathsf{A}\text{-}\mathsf{TRAN}\text{-}\mathsf{RES}}_{\mathcal{A},\mathsf{TLF}_{\mathsf{AOMCDH}}}(\lambda) \leq \mathsf{Adv}^{t\text{-}\mathsf{AOMCDH}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

B.2 Instantiation from DDH

We use the tagged linear function family $\mathsf{TLF}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{T}_{\mathsf{DDH}})$ from [BLT+24] based on the DDH assumption. We first recall the DDH assumption.

Definition 13 (DDH Assumption)**.** Let GGen be a group generation algorithm. That is, on input 1^λ, it outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p and a generator g of $\mathbb{G}$, and a description of the group operation. We say that the DDH assumption holds relative to GGen, if for all PPT algorithms $\mathcal{A}$, the following advantage is negligible:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A},\mathsf{GGen}}(\lambda) := \Bigg| &\Pr\left[\mathcal{A}(\mathbb{G},p,g,h,g^a,h^a)=1 \;\middle|\; \begin{array}{l}(\mathbb{G},g,p)\leftarrow \mathsf{GGen}(1^\lambda),\\ h \xleftarrow{\$} \mathbb{G}, a \xleftarrow{\$} \mathbb{Z}_p\end{array}\right] \\ &- \Pr\left[\mathcal{A}(\mathbb{G},p,g,h,u,v)=1 \;\middle|\; \begin{array}{l}(\mathbb{G},g,p)\leftarrow \mathsf{GGen}(1^\lambda),\\ h,u,v \xleftarrow{\$} \mathbb{G}\end{array}\right]\Bigg|.\end{aligned}$$

We make use of the implicit notation for group elements from [EHK$^+$13]. That is, we write $[\mathbf{A}] \in \mathbb{G}^{a\times b}$ for the matrix of group elements with exponents given by the matrix $\mathbf{A} \in \mathbb{Z}_p^{a\times b}$, i.e., if $\mathbf{A} = (A_{i,j})_{i\in[a],j\in[b]}$, then $[\mathbf{A}] := (g^{A_{i,j}})_{i\in[a],j\in[b]}$. The tagged linear function family $\mathsf{TLF}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{T}_{\mathsf{DDH}})$ based on DDH is defined as follows. Let GGen be an algorithm that takes as input 1^λ and outputs the description of a group $\mathbb{G}$ of prime order p with generator $g \in \mathbb{G}$. Algorithm $\mathsf{Gen}_{\mathsf{DDH}}$ runs GGen and outputs parameters $\mathsf{par} = (\mathbb{G}, g, p)$. These parameters define the following sets of scalars, tags, and the domain and range, respectively:

$$\mathcal{S} := \mathbb{Z}_p, \quad \mathcal{T} := \mathbb{G}^{2\times 2}, \quad \mathcal{D} := \mathbb{Z}_p^2, \quad \mathcal{R} := \mathbb{G}^2.$$

Given a tag $[\mathbf{G}] \in \mathbb{G}^{2\times 2}$ and an input element $\mathbf{x} \in \mathbb{Z}_p^2$, the tagged linear function $\mathsf{T}_{\mathsf{DDH}}$ is defined as

$$\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{x}) := [\mathbf{G}\mathbf{x}] \in \mathbb{G}^2.$$

Lemma 7 ([BLT$^+$24])**.** *The tagged linear function family* $\mathsf{TLF}_{\mathsf{DDH}}$ *is* ε_{r}*-regular and* ε_{t}*-translatable with* $\varepsilon_{\mathsf{r}} \le (p+1)/p^2$ *and* $\varepsilon_{\mathsf{t}} \le (3+3p)/p^2$*. Further, let* $t \in \mathbb{N}$ *be polynomial in* λ*, and assume that the* DDH *assumption holds relative to* GGen*. Then,* $\mathsf{TLF}_{\mathsf{DDH}}$ *is* t*-algebraic translation resistant. Concretely, for any PPT algorithm* $\mathcal{A}$*, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*

$$\mathsf{Adv}^{t\text{-}\mathsf{A}\text{-}\mathsf{TRAN}\text{-}\mathsf{RES}}_{\mathcal{A},\mathsf{TLF}_{\mathsf{DDH}}}(\lambda) \le \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{4}{p} + \frac{2}{p^2}.$$

C Pseudocode

```
Alg $\mathsf{Setup}(1^\lambda)$
$\mathsf{par}' \leftarrow \mathsf{TLF.Gen}(1^\lambda)$, $g \xleftarrow{\$} \mathcal{T}$
return $\mathsf{par} := (\mathsf{par}', g)$

Alg $\mathsf{Gen}(\mathsf{par})$
$a_0, \ldots, a_t \xleftarrow{\$} \mathcal{D}$
for $i \in [n] : \mathsf{sk}_i := \sum_{j=0}^{t} a_j i^j$
$\mathsf{pk} := \mathsf{pk}_0 := \mathsf{T}(g, a_0)$
return $(\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$

Alg $\mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m})$
$\varrho_i \xleftarrow{\$} \{0,1\}^{2\lambda}$, $r_i \xleftarrow{\$} \mathcal{D}$
$R_i^{(1)} := \mathsf{T}(g, r_i)$
$\mathsf{com}_i := \tilde{\mathsf{H}}(S, i, R_i^{(1)})$
$\mathsf{pm}_1 := (\varrho_i, \mathsf{com}_i)$
$St_1 := (S, i, \mathsf{sk}_i, \mathsf{m}, r_i, \mathsf{pm}_1)$
return $(\mathsf{pm}_1, St_1)$

Alg $\mathsf{Sig}_1(St_1, \mathcal{M}_1)$
parse $(S, i, \mathsf{sk}_i, \mathsf{m}, r_i, \mathsf{pm}_1) := St_1$
parse $((\varrho_j, \mathsf{com}_j))_{j \in S} := \mathcal{M}_1$
if $(\varrho_i, \mathsf{com}_i) \neq \mathsf{pm}_1$ : return $\perp$
$\varrho := \hat{\mathsf{H}}(S, \mathsf{m}, (\varrho_j)_{j \in S})$
$h := \mathsf{H}(\mathsf{m}, \varrho)$
$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$
$R_i^{(2)} := \mathsf{T}(h, r_i)$
$\mathsf{stmt} := (h, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i, \mathsf{pk}_i^{(2)})$
$\mathsf{witn} := (r_i, \mathsf{sk}_i)$
$\rho \xleftarrow{\$} \mathcal{D}$
$\pi_i := \mathsf{Prove}(\mathsf{stmt}, \mathsf{witn}; \rho)$
$\mathsf{pm}_2 := (\mathsf{pk}_i^{(2)}, R_i^{(2)}, R_i^{(1)}, \pi_i)$
$St_2 := (\mathcal{M}_1, St_1, h, \varrho, \rho, \mathsf{pm}_2)$
return $(\mathsf{pm}_2, St_2)$

Alg $\mathsf{Sig}_2(St_2, \mathcal{M}_2)$
parse $(\mathcal{M}_1, St_1, h, \varrho, \rho, \mathsf{pm}_2) := St_2$
parse $(S, i, \mathsf{sk}_i, \mathsf{m}, r_i, \mathsf{pm}_1) := St_1$
parse $((\varrho_j, \mathsf{com}_j))_{j \in S} := \mathcal{M}_1$
parse $((\mathsf{pk}_j^{(2)}, R_j^{(2)}, R_j^{(1)}, \pi_j))_{j \in S} := \mathcal{M}_2$
if $(\mathsf{pk}_i^{(2)}, R_i^{(2)}, R_i^{(1)}, \pi_i) \neq \mathsf{pm}_2$ :
    return $\perp$
for $j \in S$ :
    $\mathsf{stmt}_j := (h, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j, \mathsf{pk}_j^{(2)})$
if $\exists j \in S$ s.t. $\tilde{\mathsf{H}}(S, j, R_j^{(1)}) \neq \mathsf{com}_j$
    $\vee\ \mathsf{VerProof}(\mathsf{stmt}_j, \pi_j) = 0$ : return $\perp$
$R^{(1)} := \sum_{j \in S} R_j^{(1)}$
$R^{(2)} := \sum_{j \in S} R_j^{(2)}$
$\mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \cdot \mathsf{pk}_j^{(2)}$
$c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$
$s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$
return $\mathsf{pm}_3 := s_i$

Alg $\mathsf{Combine}(S, \mathsf{m}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3)$
parse $((\varrho_j, \mathsf{com}_j))_{j \in S} := \mathcal{M}_1$
parse $((\mathsf{pk}_j^{(2)}, R_j^{(2)}, R_j^{(1)}, \pi_j))_{j \in S} := \mathcal{M}_2$
parse $(s_j)_{j \in S} := \mathcal{M}_3$
$R^{(1)} := \sum_{j \in S} R_j^{(1)}$
$R^{(2)} := \sum_{j \in S} R_j^{(2)}$
$\mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \cdot \mathsf{pk}_j^{(2)}$
$\varrho := \hat{\mathsf{H}}(S, \mathsf{m}, (\varrho_j)_{j \in S})$
$c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$
$s := \sum_{j \in S} s_j$
return $\sigma := (\mathsf{pk}^{(2)}, c, s, \varrho)$

Alg $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$
parse $(\mathsf{pk}^{(2)}, c, s, \varrho) := \sigma$
$h := \mathsf{H}(\mathsf{m}, \varrho)$
$R^{(1)} := \mathsf{T}(g, s) - c \cdot \mathsf{pk}$
$R^{(2)} := \mathsf{T}(h, s) - c \cdot \mathsf{pk}^{(2)}$
if $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}, \varrho)$ :
    return 1
return 0
```

Figure 8: The three-round (t, n)-threshold signature scheme $\mathsf{Twinkle\text{-}T[TLF]} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ for a tagged linear function family TLF. The scheme $\mathsf{AS[TLF]} = (\mathsf{Prove}, \mathsf{VerProof})$ is as in Section 4.