Twinkle: Threshold Signatures from DDH with Full Adaptive Security

Renas Bacho [1,3] Julian Loss [1] Stefano Tessaro [2]
Benedikt Wagner [1,3] Chenzhi Zhu [2]

February 26, 2024

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{renas.bacho,loss,benedikt.wagner}@cispa.de
[2] Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle, USA
{tessaro,zhucz20}@cs.washington.edu
[3] Saarland University, Saarbrücken, Germany

Abstract

Sparkle is the first threshold signature scheme in the pairing-free discrete logarithm setting (Crites, Komlo, Maller, Crypto 2023) to be proven secure under adaptive corruptions. However, without using the algebraic group model, Sparkle's proof imposes an undesirable restriction on the adversary. Namely, for a signing threshold $t < n$, the adversary is restricted to corrupt at most $t/2$ parties. In addition, Sparkle's proof relies on a strong one-more assumption.

In this work, we propose Twinkle, a new threshold signature scheme in the pairing-free setting which overcomes these limitations. Twinkle is the first pairing-free scheme to have a security proof under up to t adaptive corruptions without relying on the algebraic group model. It is also the first such scheme with a security proof under adaptive corruptions from a well-studied non-interactive assumption, namely, the Decisional Diffie-Hellman (DDH) assumption.

We achieve our result in two steps. First, we design a generic scheme based on a linear function that satisfies several abstract properties and prove its adaptive security under a suitable one-more assumption related to this function. In the context of this proof, we also identify a gap in the security proof of Sparkle and develop new techniques to overcome this issue. Second, we give a suitable instantiation of the function for which the corresponding one-more assumption follows from DDH.

Keywords: Threshold Signatures, Adaptive Security, Pairing-Free, Non-Interactive Assumptions

Contents

1 Introduction

A threshold signature scheme [Des88, DF90, Ped91] enables a group of n signers to jointly sign a message as long as more than t of them participate. To this end, each of the n signers holds a share of the secret key associated with the public key of the group. When $t + 1$ of them come together and run a signing protocol for a particular message, they obtain a compact signature (independent in size of t and n) without revealing their secret key shares to each other. On the other hand, no subset of at most t potentially malicious signers can generate a valid signature. Despite being a well-studied cryptographic primitive, threshold signatures have experienced a renaissance due to their use in cryptocurrencies [LN18] and other modern applications [DOK$^+$20]. This new attention has also led to ongoing standardization efforts [BP22]. In this work, we study threshold signatures in the pairing-free discrete logarithm setting. As noted in previous works [TZ22, TZ23, CKM$^+$23b], pairings are not supported in popular libraries and are substantially more expensive to compute, which makes pairing-free solutions appealing.

Static vs. Adaptive Security. When defining security for threshold signatures, the adversary is allowed to concurrently interact with honest signers in the signing protocol. Additionally, it may corrupt up to t out of n parties, thereby learning their secret key material and internal state. Here, we distinguish between *static* corruptions and *adaptive* corruptions. For static corruptions, the adversary declares the set of corrupted parties ahead of time before any messages have been signed. For adaptive corruptions, the adversary can corrupt parties dynamically, depending on previous signatures and corruptions.

Adaptive security is a far stronger notion than static security and matches reality more closely. Unfortunately, proving adaptive security for threshold signatures is highly challenging and previous works in the pairing-free setting rely on strong interactive assumptions to simulate the state of adaptively corrupted parties [CKM23a]. This simulation strategy, however, is at odds with rewinding the adversary as part of a security proof. Roughly, if the adversary is allowed to corrupt up to t_c parties, then in the two runs induced by rewinding, it may corrupt up to $2t_c$ parties in total. Thus, for the reduction to obtain meaningful information from the adversary's forgery, it has to be restricted to corrupt at most $t_c \leq t/2$ parties [CKM23a]. To bypass this unnatural restriction, prior work heavily relies on the algebraic group model (AGM) [FKL18] in order to avoid rewinding[1]. In summary: to support an arbitrary corruption threshold, one has to use the AGM or sacrifice adaptive security.

1.1 Our Contribution

Motivated by this unsatisfactory state of affairs, we construct Twinkle. Twinkle is the first threshold signature scheme in the pairing-free setting which combines all of the following characteristics:

- *Adaptive Security.* We prove Twinkle secure under adaptive corruptions. Notably, we do not rely on secure erasures of private state.
- *Non-Interactive Assumptions.* Our security proof relies on a non-interactive and well-studied assumption, namely, the DDH assumption. As a slightly more efficient alternative, we give an instantiation based on a one-more variant of CDH, for which we provide evidence of its hardness.
- *No AGM.* Our security proof does not rely on the algebraic group model, but only on the random oracle model.
- *Arbitrary Threshold.* Twinkle supports an arbitrary corruption threshold $t < n$ for n parties. Essentially, this is established by giving a proof without rewinding.

For a comparison of schemes in the pairing-free discrete logarithm setting, see Table 1. We also emphasize that we achieve our goal without the use of heavy cryptographic techniques, and our scheme is practical. For example, signatures of Twinkle (from DDH) are at most 3 times as large as regular Schnorr signatures [Sch91], and Twinkle has three rounds. In the context of our proof, we also identify a gap in the analysis of Sparkle [CKM23a] and develop new proof techniques to fix it in the context of our scheme[2].

[1]Other works resort to heavier machinery such as broadcast channels or non-committing encryption resulting in inefficient protocols.

[2]We communicated the gap and our solution to the authors of Sparkle. To be clear, we do not claim that Sparkle is insecure, just that the proof in [CKM23a] has a gap.

Scheme	Rounds	Adaptive	Assumption	Idealization	Corruptions
GJKR [GJKR07]/StiStr [SS01]	≥ 4	✗	DLOG	ROM	$\leq t < n/2$
Lin-UC [Lin22]	3	✗	DLOG	ROM	$\leq t$
Frost [KG20]	2	✗	DLOG	Custom	$\leq t$
Frost [KG20, BTZ22, BCK+22]	2	✗	AOMDL	ROM	$\leq t$
Frost2 [CKM21, BTZ22, BCK+22]	2	✗	AOMDL	ROM	$\leq t$
Frost3 [RRJ+22]/Olaf [CGRS23]	2	✗	AOMDL	ROM	$\leq t$
TZ [TZ23]	2	✗	DLOG	ROM	$\leq t$
Sparkle [CKM23a]	3	✗	DLOG	ROM	$\leq t$
Sparkle [CKM23a]	3	✓	AOMDL	ROM	$\leq t/2$
Sparkle [CKM23a]	3	✓	AOMDL	ROM+AGM	$\leq t$
Twinkle (AOMCDH)	3	✓	AOMCDH	ROM	$\leq t$
Twinkle (DDH)	3	✓	DDH	ROM	$\leq t$

Table 1: Comparison of different threshold signature schemes in the discrete logarithm setting without pairings and the two instantiations of our Twinkle scheme. We compare whether the schemes are proven secure under adaptive corruptions and under which assumption and idealized model they are proven. We also compare the corruption thresholds that they support. For all schemes, we assume that there is a trusted dealer distributing key shares securely. For GJKR [GJKR07]/StiStr [SS01], broadcast channels are assumed, which adds rounds when implemented.

Conceptually, the design of our threshold signature is inspired by five-move identification schemes, which already have found use in the construction of tightly secure signature schemes [Che05, GJKW07, KLP17]. We achieve our result in two main steps:

1. We first phrase our scheme abstractly using (a variant of) linear function families [HKL19, KLR21, CAHL+22, PW23, TZ23]. To prove security under adaptive corruptions, we define a security notion for linear functions resembling a one-more style CDH assumption. This is the step where we identify the gap in the analysis of Sparkle [CKM23a].

2. We then instantiate the linear function family such that this one-more notion follows from the (non-interactive) DDH assumption. Note that Tessaro and Zhu [TZ23] showed a related statement, namely, that a suitable one-more variant of DLOG follows from DLOG. In this sense, our work makes a further step in an agenda aimed at replacing interactive assumptions with non-interactive ones. We are confident that this is interesting in its own right.

1.2 Technical Overview

We keep the technical overview self-contained, but some background on Schnorr signatures [Sch91, KMP16], five-move identification [Che05, GJKW07, KLP17], and Sparkle [CKM23a] is helpful.

Sparkle and The Problem with Rewinding. As our starting point, let us review the main ideas behind Sparkle [CKM23a], and why the use of rewinding limits us to tolerating at most $t/2$ corruptions. For that, we fix a group $\mathbb{G}$ with generator g and prime order p. Each signer $i \in [n]$ holds a secret key share $\mathsf{sk}_i \in \mathbb{Z}_p$ such that $\mathsf{sk}_i = f(i)$ for a polynomial f of degree t. Further, the public key is $\mathsf{pk} = g^{f(0)}$. To sign a message m, a set $S \subseteq [n]$ of signers engage in the following interactive signing protocol, omitting some details:

1. Each party $i \in S$ samples a random $r_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $R_i = g^{r_i}$. It then sends a hash com_i of R_i, S, and m to the other signers to commit to R_i. We call R_i a *preimage* of com_i. The hash function is modeled as a random oracle.

2. Once a party has received all hashes from the first round, it sends R_i to the other signers to open the commitment.

3. If all commitments are correctly opened, each signer computes the combined nonce $R = \prod_i R_i$. Then, it derives a challenge $c \in \mathbb{Z}_p$ from pk, R, and m using another random oracle. Each signer i computes and sends its response share $s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$, where $\ell_{i,S}$ is a Lagrange coefficient. The signature is (c, s), where $s = \sum_i s_i$.

The overall proof strategy adopted in [CKM23a] follows a similar paradigm as that of proving Schnorr signatures, with appropriate twists. Namely, one first takes care of simulating signing queries using honest-verifier zero-knowledge (HVZK) and by suitably programming the random oracle. We will come back to this part of the proof later. Then, via rewinding, one can extract the secret key from a forgery. To simulate adaptive corruption queries, the proof of $\mathsf{Sparkle}$ relies on a DLOG oracle on each corruption query, i.e., security is proven under the one-more version of DLOG (OMDL). Specifically, getting $t+1$ DLOG challenges from the OMDL assumption and t-time access to a DLOG oracle, the reduction defines a degree t polynomial "in the exponent", simulates the game as explained, and uses rewinding to solve the final DLOG challenge. Note that if we allow the adversary to corrupt at most t_c parties throughout the experiment, it may corrupt up to $2t_c$ parties over both runs, meaning that the reduction has to query the DLOG oracle up to $2t_c$ times. Therefore, we have to require that $2t_c \leq t$.

How to Avoid Rewinding. Now it should be clear that the restriction on the corruption threshold is induced by the use of rewinding. If we avoid rewinding, we can also remove the restriction. To do so, it is natural to follow existing approaches from the literature on tightly-secure (and thus rewinding-free) signatures. A common approach is to rely on lossy identification [KW03, AFLT12, KMP16] that has already been used in the closely-related multi-signature setting [PW23]. We find this unsuitable for two reasons. Namely, (a) these schemes rely on the DDH assumption, it is not clear at all what a suitable one-more variant would look like, and (b) the core idea of this technique is to move to a hybrid in which there is no secret key for pk at all. This seems hard to combine with adaptive corruptions. Roughly, this is because if there is no secret key for pk, then at most t of the pk_i can have a secret key, meaning that we would have to guess the set of corruptions. Instead, we take inspiration from five-move identification [Che05, GJKW07, KLP17], for which problems (a) and (b) do not show up. Namely, (a) such schemes rely on the CDH assumption, and (b) there is always a secret key. To explain the idea, we directly focus on our threshold signature scheme. For that, let $h \in \mathbb{G}$ be *derived from the message* via a random oracle. Given h, our signing protocol is as follows:

1. Each signer $i \in S$ samples $r_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and computes $R_i^{(1)} = g^{r_i}, R_i^{(2)} = h^{r_i}$, and $\mathsf{pk}_i^{(2)} = h^{\mathsf{sk}_i}$. It then sends a hash of $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$ to the other signers.

2. Once a party received all hashes from the first round, it sends $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$.

3. If all commitments are correctly opened, each signer computes the combined nonces $R^{(k)}$ for $k \in \{1,2\}$ and secondary public key $\mathsf{pk}^{(2)}$ in a natural way. Then, it derives a challenge c from $R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}$, and m and computes $s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$. The signature is $(\mathsf{pk}^{(2)}, c, s)$ with $s = \sum_i s_i$.

Intuitively, the signers engage in two executions of $\mathsf{Sparkle}$ with generators g and h, respectively, using the same randomness r_i. To understand why we can avoid rewinding with this scheme, let us ignore signing and corruption queries for a moment, and focus on how to turn a forgery $(\mathsf{pk}^{(2)}, c, s)$ into a solution for a hard problem, concretely, CDH. For that, we consider two cases. First, if $\mathsf{pk}^{(2)} = h^{f(0)}$, then $\mathsf{pk}^{(2)}$ is a CDH solution for $\mathsf{pk} = g^{f(0)}$ and h. Indeed, this is what should happen in an honest execution. Second, we can bound the probability that the forgery is valid and $\mathsf{pk}^{(2)} \neq h^{f(0)}$ using a statistical argument. Roughly, (c, s) acts as a statistically sound proof for the statement $\mathsf{pk}^{(2)} = h^{f(0)}$. To simulate adaptive corruptions, for now assume that we can rely on a one-more variant of the CDH assumption, in which we have t-time access to a DLOG oracle. We come back to this later. What remains is to simulate honest parties during the signing. For that, the first trick is to set up h (by programming the random oracle) in a special way. Roughly, we want to be able to translate valid transcripts with respect to g into valid transcripts with respect to h. Once this is established, we can focus on simulating the g-side of the protocol.

A Gap In the Proof of Sparkle. If we only focus on the g-side, our protocol is essentially $\mathsf{Sparkle}$. Therefore, it should be possible to simulate signing exactly as in $\mathsf{Sparkle}$ using HVZK. Unfortunately, when looking at this part of $\mathsf{Sparkle}$'s proof, we discovered that a certain adversarial behavior is not covered. Namely, the proof does not correctly simulate the case in which the adversary sends inconsistent sets of commitments to different honest parties. It turns out that handling this requires fundamentally new techniques. To understand the gap, it is instructive to consider $\mathsf{Sparkle}$'s proof for an example of three signers in a session *sid*, with two of them being honest, say Signer 1 and 2, and the third one being malicious. Let us assume that Signers 1 and 2 are already in the second round of the protocol. That is, both already sent their commitments com_1 and com_2 and now expect a list of commitments $\mathcal{M} = (\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3)$ from the first round as input. In $\mathsf{Sparkle}$'s proof, the reduction sends random commitments com_1 and com_2 on behalf of the honest parties. Later, when Signer 1 (resp. 2) gets $\mathcal{M}$, it has to output its second message R_1 (resp. R_2) and program the random oracle at R_1 (resp. R_2) to be com_1 (resp. com_2). The goal of the reduction is to set up R_1 and R_2 using HVZK such that the responses s_1 and s_2 can be computed without using the secret key. To understand how the reduction proceeds, assume that Signer 1 is asked (by the adversary) to reveal his nonce R_1 first. When this happens, the reduction samples a challenge c and a response s_1. It then defines R_1 as $R_1 := g^{s_1}\mathsf{pk}_1^{-c\ell_{1,S}}$. Ideally, the reduction would now program the random oracle on the combined nonce $R = R_1R_2R_3$ to return c, and output R_1 to the adversary. However, while the reduction can extract R_3 from com_3 by observing the random oracle queries, R_2 is not yet defined at that point. The solution proposed in $\mathsf{Sparkle}$'s proof is as follows. Before returning R_1 to the adversary, the reduction also samples s_2 and defines $R_2 := g^{s_2}\mathsf{pk}_2^{-c\ell_{2,S}}$. Then, the reduction can compute the combined nonce $R = R_1R_2R_3$ and program the random oracle on input R to return c. Later, it can use s_1 and s_2 as responses.

However, as we will argue now, this strategy is flawed[3]. Think about what happens if the first-round messages $\mathcal{M}'$ that Signer 2 sees do not contain com_3, but instead a different[4] commitment com_3' to a nonce $R_3' \neq R_3$. Then, with high probability, the combined nonce R' that Signer 2 will compute is different from R, meaning that its challenge c' will also be different from c, and so s_2 is not a valid response. One naive idea to solve this is to program $R_2 := g^{s_2}\mathsf{pk}_2^{-c'\ell_{2,S}}$ for an independent c' when we reveal R_1. In this case, however, the adversary may just choose to submit $\mathcal{M}' = \mathcal{M}$ to Signer 2, making the simulation fail.

Equivalence Classes to the Rescue. The solution we present is very technical, and we sketch a massively simplified solution here. Abstractly speaking, we want to be able to identify whether two queries $q = (sid, i, \mathcal{M})$ and $q' = (sid', i', \mathcal{M}')$ will result in the same combined nonce *before* all commitments com_j in $\mathcal{M}$ and $\mathcal{M}'$ have preimages R_j. To do so, we define an equivalence relation $\sim$ on such queries for which we show two properties.

1. First, the equivalence relation is consistent over time, namely, (a) if $q \sim q'$ at some point in time, then $q \sim q'$ at any later point, and (b) if $q \not\sim q'$ at some point in time, then $q \not\sim q'$ at any later point.

2. Second, assume that all commitments in $\mathcal{M}$ and $\mathcal{M}'$ have preimages. Then the resulting combined nonces R and R' are the same if and only if $q \sim q'$.

The technical challenge is that $\sim$ has to stay consistent while also adapting to changes in the random oracle over time. Assuming we have such a relation, we can make the simulation work. Namely, when we have to reveal the nonce R_i of an honest signer i, we first define $c := \mathsf{C}(q)$, where C is a *random oracle on equivalence classes* and is only known to the reduction. That is, C is a random oracle with the additional condition that $\mathsf{C}(q) = \mathsf{C}(q')$ if $q \sim q'$. Then, we define $R_i := g^{s_i}\mathsf{pk}_i^{-c\ell_{i,S}}$. We do not define any other $R_{i'}$ of honest parties at that point, meaning that we also may not know the combined nonce yet. Instead, we carefully delay the random oracle programming of the combined nonce until it is completely known.

Cherry on Top: Non-Interactive Assumptions. While the scheme we have so far does its job, we still rely on an interactive assumption, and we are eager to avoid it. For that, it is useful to write our scheme abstractly, replacing every exponentiation with the function $\mathsf{T}(t, x) = t^x$. Note that for almost every $t \in \mathbb{G}$, the function $\mathsf{T}(t, \cdot)$ is a bijection. Our hope is that by instantiating our scheme with a different function with suitable properties, we can show that the corresponding one-more assumption is implied by a non-interactive assumption. Indeed, Tessaro and Zhu [TZ23] recently used a similar strategy to avoid OMDL in certain situations. To do so, they replace the bijective function with a compressing function. In our case, the interactive assumption, written abstractly using T, asks an adversary to win the following game:

- A random g and h are sampled, and random $x_0, \ldots, x_t$ are sampled. Then, g, h, and all $X_i = \mathsf{T}(g, x_i)$ for all $0 \leq i \leq t$ are given to the adversary.

[3]The problem has nothing to do with adaptive security and shows up for a static adversary as well.

[4]Note that in $\mathsf{Sparkle}$, no broadcast channel is assumed, and so this may happen. Also, note that in multi-signatures that follow a similar strategy, e.g. [BN06], this problem does not show up as there is only one honest signer.

- Roughly, the adversary gets t-time access to an algebraic oracle inverting T. More precisely, the oracle outputs $\sum_{i=0}^{t} \alpha_i x_i$ on input $\alpha_0, \ldots, \alpha_t$.
- The adversary outputs X_i' for all $0 \leq i \leq t$. It wins if all solutions are valid, meaning that there is a z_i such that $\mathsf{T}(g, z_i) = X_i \wedge \mathsf{T}(h, z_i) = X_i'$. Intuitively, the adversary has to "shift" the images X_i from g to h.

Under a suitable instantiation of T and a well-studied non-interactive assumption, we want to show that no adversary can win this game. Unfortunately, if we just use a compressing function as in the case of [TZ23], it is not clear how to make use of the winning condition. Instead, our idea is to use a function that can *dynamically* be switched between a bijective and a compressing mode. A bit more precisely, a proof sketch works as follows:

1. We start with the game we introduced above. With overwhelming probability, the functions $\mathsf{T}_g := \mathsf{T}(g, \cdot)$ and $\mathsf{T}_h := \mathsf{T}(h, \cdot)$ should be bijective.
2. Assume that we can efficiently invert T_h using knowledge of h. Then, we can state our winning condition equivalently by requiring that $\mathsf{T}_h^{-1}(X_i') = x_i$ for all i. Roughly, this means that the adversary has to find the x_i to win.
3. We assume that we can indistinguishably switch g to a mode in which T_g is compressing.
4. Finally, we use a statistical argument to show that the adversary can not win. Intuitively, this is because T_g is compressing and the inversion oracle does not leak too much about the x_i's.

It turns out that, choosing T carefully, we find a function that (1) has all the properties we need for our scheme and (2) allows us to follow our proof sketch under the DDH assumption.

1.3 More on Related Work

We discuss further related work, including threshold signatures from other assumptions, and related cryptographic primitives.

Techniques for Adaptive Security. General techniques for achieving adaptive security have been studied [CGJ+99, JL00, LP01]. Unfortunately, these techniques often rely on heavy cryptographic machinery and assumptions, e.g., secure erasures or broadcast channels.

Other Algebraic Structures. In the pairing setting, a natural construction is the (non-interactive) threshold version of the BLS signature scheme [BLS01, Bol03], which has been modified to achieve adaptive security in [LJY14]. Recently, Bacho and Loss [BL22] have proven adaptive security of threshold BLS in the AGM. Das et al. have constructed weighted threshold signatures in the pairing-setting [DCX+23], and Crites et al. have constructed structure-preserving threshold signatures in the pairing-setting [CKP+23]. Threshold signatures have been constructed based on RSA [DDFY94, Rab98, FMY98, Sho00, ADN06, GHKR08, TZ23]. Notably, adaptive security has been considered in [ADN06]. A few works also have constructed threshold signatures from lattices [BKP13, BGG+18, DOTT21, ASY22, GKS23]. Finally, several works have proposed threshold signing protocols for ECDSA signatures [GGN16, LN18, GG18, DKLs19, DJN+20, GG20, CGG+20, CCL+20, GKSŚ20]. Except for [CGG+20], these works focus on static corruptions. For an overview of this line of work, see [AHS20].

Robustness. Recently, there has been renewed interest in robust (Schnorr) threshold signing protocols [RRJ+22, BHK+23, Sho23, GS23]. Such robust protocols additionally ensure that no malicious party can prevent honest parties from signing. Notably, all of these protocols assume static corruptions.

Multi-Signatures. Multi-signatures [IN83, BN06] are threshold signatures with $t = n - 1$, i.e., all n parties need to participate in the signing protocol, with the advantage that parties generate their keys independently and come together to sign spontaneously without setting up a shared key. There is a rich literature on multi-signatures, e.g., [Bol03, BDN18, MPSW19, NRSW20, NRS21, BD21, AB21, BTT22, FSZ22, TZ23]. Closest to our work in spirit are the work by Pan and Wagner [PW23], which avoids rewinding, and the work of Tessaro and Zhu [TZ23], which aims at non-interactive assumptions.

Distributed Key Generation. In principle, one can rely on generic secure multi-party computation to set up key shares for a threshold signature scheme without using a trusted dealer. To get a more efficient solution, dedicated distributed key generation protocols have been studied [Ped92, CGJ+99, JL00, GJKR07, KMS20, DYX+22, KGS23], with some of them being adaptively secure [CGJ+99, JL00, KMS20].

2 Preliminaries

By λ we denote the security parameter. We assume all algorithms get λ in unary as input. If X is a finite set, we write $x \xleftarrow{\$} X$ to indicate that x is sampled uniformly at random from X. If $\mathcal{A}$ is a probabilistic algorithm, we write $y := \mathcal{A}(x;\rho)$ to state that y is assigned to the output of $\mathcal{A}$ on input x with random coins ρ. If ρ is sampled uniformly at random, we simply write $y \leftarrow \mathcal{A}(x)$. Further, the notation $y \in \mathcal{A}(x)$ indicates that y is a possible output of $\mathcal{A}$ on input x, i.e., there are random coins ρ such that $\mathcal{A}(x;\rho)$ outputs y.

Threshold Signatures. We define threshold signatures, assuming a trusted key generation, which can be replaced by a distributed key generation in practice. Our syntax matches the three-round structure of our protocol. Namely, a (t,n)-threshold signature scheme is a tuple of PPT algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$, where $\mathsf{Setup}(1^\lambda)$ outputs system parameters par, and $\mathsf{Gen}(\mathsf{par})$ outputs a public key pk and secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$. Further, Sig specifies a signing protocol, formally split into four algorithms $(\mathsf{Sig}_0, \mathsf{Sig}_1, \mathsf{Sig}_2, \mathsf{Combine})$. Here, algorithm Sig_j models how the signers locally compute their $(j+1)$st protocol message pm_{j+1} and advance their state, where $\mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m})$ 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, and Sig_1 (resp. Sig_2) takes as input the current state of the signer and the list $\mathcal{M}_1$ (resp $\mathcal{M}_2$) of all protocol messages from the previous round. Finally, $\mathsf{Combine}(S, \mathsf{m}, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3)$ can be used to publicly turn the transcript into a signature σ, which can then be verified using $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$. Roughly, we say that the scheme is complete if for any such parameters and keys, a signature generated by a signing protocol among $t+1$ 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 global system parameters par, where par implicitly defines sets of public keys, secret keys, messages and signatures, and all algorithms related to TS implicitly take par as input.

- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ takes as input system parameters par, and outputs a public key pk and secret key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$.

- $\mathsf{Sig} = (\mathsf{Sig}_0, \mathsf{Sig}_1, \mathsf{Sig}_2, \mathsf{Combine})$ 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 $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,l})$ 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 $\mathcal{M}_2 = (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,l})$ 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 $\mathcal{M}_1 = (\mathsf{pm}_{1,1}, \ldots, \mathsf{pm}_{1,l}), \mathcal{M}_2 = (\mathsf{pm}_{2,1}, \ldots, \mathsf{pm}_{2,l})$, and $\mathcal{M}_3 = (\mathsf{pm}_{3,1}, \ldots, \mathsf{pm}_{3,l})$ 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 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.

Our security game is in line with the established template and is presented in Figure 2. First, the adversary gets an honestly generated public key as input. At any point in time, the adversary can start a new signing session with signer set S and message m with session identifier sid by calling an oracle

```
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. $i_1 < i_2 \cdots i_t < i_{t+1}$
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 (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 Sig.

$\textsc{Next}(sid, S, \mathsf{m})$. Additionally, the adversary may adaptively corrupt up to t users via an oracle $\textsc{Corr}$. Thereby, it learns their secret key and private state in all currently open signing sessions. To interact with honest users in signing sessions, the adversary has access to per-round signing oracles $\textsc{Sig}_0, \textsc{Sig}_1, \textsc{Sig}_2$. Roughly, each signing oracle can be called with respect to a specific honest user i and a session identifier sid, given that the user is already in the respective round for that session (modeled by algorithm $\mathsf{Allowed}$). Further, when calling such an oracle, the adversary inputs the vector of all messages of the previous round. In particular, the adversary could send different messages to two different honest parties within the same session, i.e., we assume no broadcast channels. Additionally, this means that the adversary can arbitrarily decide which message to send to an honest party on behalf of another honest party, i.e., we assume no authenticated channels. Finally, the adversary outputs a forgery (m^*, σ^*). It wins the security game, if it never started a signing session for message m^* and the signature σ^* is valid. Therefore, our notion is (an interactive version of) TS-UF-0 using the terminology of [BTZ22, BCK+22], which is similar to recent works [CKM23a, CGRS23].

No Erasures. In our pseudocode, the private state of signer i in session sid is stored in $\mathsf{state}[sid, i]$, where state is a map. After each signing round, this state is updated. We choose to update the state instead of adding a new state to avoid clutter, which is similar to earlier works [CKM23a]. On the downside, this means that potentially, schemes that are secure in our model could rely on erasures, i.e., on safely deleting part of the state of an earlier round before a user gets corrupted. We emphasize that in our scheme, any state in earlier rounds can be computed from the state in the current round and the secret key. This means that our schemes do not rely on erasures.

Definition 2 (TS-EUF-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 TS-EUF-CMA secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

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

3 Our Construction

In this section, we present our new threshold signature scheme. However, before we present it, we first introduce a building block we need, which we call tagged linear function families.

3.1 Tagged Linear Function Families

Similar to what is done in other works [HKL19, KLR21, CAHL+22, PW23, TZ23], we use the abstraction of linear function families to describe our scheme in a generic way. However, we slightly change the notion by introducing tags to cover different functions with the same set of parameters.

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:

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

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

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

Oracle $\textsc{Sig}_1(sid, i, \mathcal{M}_1)$
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 $\textsc{Sig}_2(sid, i, \mathcal{M}_2)$
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 $\mathcal{A}$.

- $\mathsf{Gen}(1^\lambda) \rightarrow \mathsf{par}$ 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.
- $\mathsf{T}(\mathsf{par}, g, x) \rightarrow X$ is deterministic, takes as input parameters par, a tag $g \in \mathcal{T}$, 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.

For our construction, we require that images are uniformly distributed. We formally define this next.

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 such that the following two properties hold:

- We have

$$\Pr\left[(\mathsf{par}, g) \notin \mathsf{Reg} \mid \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda),\ g \stackrel{\$}{\leftarrow} \mathcal{T}\right] \leq \varepsilon_{\mathsf{r}}.$$

- For any fixed $(\mathsf{par}, g) \in \mathsf{Reg}$, the following distributions are the same:

$$\{(\mathsf{par}, g, X) \mid X \stackrel{\$}{\leftarrow} \mathcal{R}\} \text{ and } \{(\mathsf{par}, g, X) \mid x \stackrel{\$}{\leftarrow} \mathcal{D},\ X := \mathsf{T}(\mathsf{par}, g, x)\}.$$

Next, we show that tagged linear function families satisfy a statistical property that turns out to be useful. This property is implicitly present in other works as well, e.g., in [KW03, AFLT12, KLP17, PW23], and can be interpreted in various ways, e.g., as the soundness of a natural proof system.

Lemma 1. *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*

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

Proof. We claim that for each fixed values $\mathsf{par}, g, h, X_1, X_2, R_1, R_2$ such that $(X_1, X_2) \notin \mathsf{Im}(\mathsf{par}, g, h)$, the following set contains at most one element:

$$\mathsf{BadC} := \{c \in \mathcal{S} \mid \exists s \in \mathcal{D}: \; \mathsf{T}(g, s) = c \cdot X_1 + R_1 \wedge \mathsf{T}(h, s) = c \cdot X_2 + R_2\}.$$

The reader may observe that this is indeed sufficient to show the lemma. To show the claim, assume towards contradiction that there are two distinct $c \neq c'$ in BadC and $s, s' \in \mathcal{D}$ were the corresponding witnesses with

$$\begin{aligned} \mathsf{T}(g, s) &= c \cdot X_1 + R_1, & \mathsf{T}(g, s') &= c' \cdot X_1 + R_1 \\ \text{and} \quad \mathsf{T}(h, s) &= c \cdot X_2 + R_2, & \mathsf{T}(h, s') &= c' \cdot X_2 + R_2. \end{aligned}$$

We rearrange these equations and get

$$\begin{aligned} \mathsf{T}(g, s) - c \cdot X_1 &= R_1 = \mathsf{T}(g, s') - c' \cdot X_1 \\ \text{and } \mathsf{T}(h, s) - c \cdot X_2 &= R_2 = \mathsf{T}(h, s') - c' \cdot X_2. \end{aligned}$$

Rearranging again, using the linearity of T for any fixed tag, and solving for (X_1, X_2), we get

$$X_1 = \mathsf{T}\left(g, \frac{s - s'}{c - c'}\right) \text{ and } X_2 = \mathsf{T}\left(h, \frac{s - s'}{c - c'}\right).$$

Hence, $(X_1, X_2) \in \mathsf{Im}(\mathsf{par}, t, h)$. With this contradiction, we conclude. $\square$

As another technical tool in our proof, we need our tagged linear function families to be translatable, a notion we define next. Informally, it means that we can rerandomize a given tag g into a tag h, such that we can efficiently compute $\mathsf{T}(h, x)$ from $\mathsf{T}(g, x)$ without knowing x.

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 algorithm $\mathsf{Translate}$, 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}:

$$\begin{aligned} \mathcal{X}_0 &:= \left\{(\mathsf{par}, g, h) \;\middle|\; \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda), \; g \xleftarrow{\$} \mathcal{T}, \; h \xleftarrow{\$} \mathcal{T}\right\}, \\ \mathcal{X}_1 &:= \left\{(\mathsf{par}, g, h) \;\middle|\; \mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda), \; g \xleftarrow{\$} \mathcal{T}, \; (h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}, g)\right\}. \end{aligned}$$

- **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

$$\mathsf{Translate}(\mathsf{td}, \mathsf{T}(g, x)) = \mathsf{T}(h, x) \text{ and } \mathsf{InvTranslate}(\mathsf{td}, \mathsf{T}(h, x)) = \mathsf{T}(g, x).$$

Next, we define the main security property that we will require for our construction. Intuitively, it should not be possible for an adversary to translate $\mathsf{T}(g, x)$ into $\mathsf{T}(h, x)$ if g, h and x are chosen randomly. Our actual notion is a one-more variant of this intuition.

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:

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

```
Game $t$-A-TRAN-RES$^{\mathcal{A}}_{\mathsf{TLF}}(\lambda)$
01 $\mathsf{par} \leftarrow \mathsf{Gen}(1^\lambda),\quad g,h \xleftarrow{\$} \mathcal{T},\quad x_0,\dots,x_t \xleftarrow{\$} \mathcal{D}$
02 for $i \in \{0\} \cup [t]$ : $X_i := \mathsf{T}(g,x_i)$
03 $(X'_i)_{i=0}^t \leftarrow \mathcal{A}^{\textsc{Inv}}(\mathsf{par}, g, h, (X_i)_{i=0}^t)$
04 if $\forall i \in \{0\} \cup [t]\ \exists z \in \mathcal{D}$
        s.t. $\mathsf{T}(g,z) = X_i \wedge \mathsf{T}(h,z) = X'_i$ :
05          return 1
06 return 0

Oracle $\textsc{Inv}(\alpha_0,\dots,\alpha_t)$
07 if $q \geq t$ : return $\perp$
08 $q := q+1$
09 $x := \sum_{i=0}^t \alpha_i x_i$
10 return $x$
```

Figure 3: Game **A-TRAN-RES** for a tagged linear function family $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ and adversary $\mathcal{A}$.

3.2 Construction

Let $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ be a tagged linear function family. Further, let $\mathsf{H}\colon \{0,1\}^* \to \mathcal{T}$, $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\bar{\mathsf{H}}\colon \{0,1\}^* \to \mathcal{S}$ be random oracles. We construct a (t,n)-treshold signature scheme $\mathsf{Twinkle}[\mathsf{TLF}] = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. 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 ith lagrange coefficient for all $i \in [n]$ and $S \subseteq [n]$, and let $\ell_{i,S} := \ell_{i,S}(0)$. We describe our scheme verbally and provide pseudocode in Figure 5.

Setup and Key Generation. All parties have access to public parameters $\mathsf{par} \leftarrow \mathsf{TLF}.\mathsf{Gen}(1^\lambda)$ 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 $j \in \{0\} \cup [t]$ 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 ith signer as

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

The shared 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 all signers are aware of the set S and a message $\mathsf{m} \in \{0,1\}^*$ to be signed. First, they all compute $h := \mathsf{H}(\mathsf{m})$. Then, they run the following protocol phases to compute the signature:

1. *Commitment Phase.* Each signer $i \in S$ samples $r_i \xleftarrow{\$} \mathcal{D}$ and computes

   $$R_i^{(1)} := \mathsf{T}(g, r_i),\quad R_i^{(2)} := \mathsf{T}(h, r_i),\quad \mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i).$$

   Then, each signer $i \in S$ computes a commitment

   $$\mathsf{com}_i := \hat{\mathsf{H}}(S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$$

   and sends com_i to the other signers.

2. *Opening Phase.* Each signer $i \in S$ sends $R_i^{(1)}, R_i^{(2)}$ and $\mathsf{pk}_i^{(2)}$ to all other signers.

3. *Response Phase.* Each signer $i \in S$ checks that $\mathsf{com}_j = \hat{\mathsf{H}}(S, j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)})$ holds for all $j \in S$. If one of these equations does not hold, the signer aborts. Otherwise, the signer defines

   $$R^{(1)} := \sum_{j\in S} R_j^{(1)},\quad R^{(2)} := \sum_{j\in S} R_j^{(2)},\quad \mathsf{pk}^{(2)} := \sum_{j\in S} \ell_{j,S}\mathsf{pk}_j^{(2)}.$$

   The signer computes $c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$ and $s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$. It sends s_i to all other signers.

The signature is $\sigma := (\mathsf{pk}^{(2)}, c, s)$ for $s := \sum_{j\in S} s_j$.

Verification. Let pk be a public key, let $\mathsf{m} \in \{0,1\}^*$ be a message and let $\sigma = (\mathsf{pk}^{(2)}, c, s)$ be a signature. To verify σ with respect to pk and m, one first computes $h := \mathsf{H}(\mathsf{m})$ 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 $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$.

Theorem 1. *Let* $\mathsf{TLF} = (\mathsf{Gen}, \mathsf{T})$ *be a tagged linear function family and let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{T}$, $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\bar{\mathsf{H}}\colon \{0,1\}^* \to \mathcal{S}$ *be random oracles. Assume that* TLF *is* ε_{r}*-regular and* ε_{t}*-translatable. Further, assume that* TLF *is* t*-algebraic translation resistant. Then,* $\mathsf{Twinkle}[\mathsf{TLF}]$ *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* $\textsc{Sig}_0, \textsc{Sig}_1, \textsc{Sig}_2$ *and at most* $Q_{\mathsf{H}}, Q_{\hat{\mathsf{H}}}, Q_{\bar{\mathsf{H}}}$ *queries to oracles* $\mathsf{H}, \hat{\mathsf{H}}, \bar{\mathsf{H}}$*, respectively, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{TS\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Twinkle}[\mathsf{TLF}]}(\lambda) \leq \quad & 8Q_S Q_{\mathsf{H}}\varepsilon_{\mathsf{t}} + 8Q_S\varepsilon_{\mathsf{r}} + \frac{8Q_S^2 Q_{\hat{\mathsf{H}}} + 4Q_S^3(Q_S+t)}{|\mathcal{R}|} + \frac{4Q_S Q_{\bar{\mathsf{H}}}}{|\mathcal{S}|} \\ & + \frac{4Q_S^3 + 4Q_S Q_{\hat{\mathsf{H}}}^2 + 4Q_{\hat{\mathsf{H}}}Q_S^2(t+1)}{2^{2\lambda}} + 4Q_S \cdot \mathsf{Adv}^{t\text{-}\mathsf{A}\text{-}\mathsf{TRAN}\text{-}\mathsf{RES}}_{\mathcal{B},\mathsf{TLF}}(\lambda).\end{aligned}$$

Proof. Fix an adversary $\mathcal{A}$ against the security of $\mathsf{TS} := \mathsf{Twinkle}[\mathsf{TLF}]$. We prove the statement by presenting a sequence of games $\mathbf{G}_0$-$\mathbf{G}_8$. To accompany the verbal description, all games and associated oracles and algorithms are presented as pseudocode in Figures 6 to 11.

Game $\mathbf{G}_0$: This game is the security game $\mathbf{TS\text{-}EUF\text{-}CMA}^{\mathcal{A}}_{\mathsf{TS}}$ for threshold signatures. We recall the game to fix some notation. First, the game samples parameters par' for TLF and a tag $g \xleftarrow{\$} \mathcal{T}$. It also samples random coefficients $a_0, \dots, a_t \xleftarrow{\$} \mathcal{D}$ and computes a 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 $i \in [n]$. For convenience, 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 $\mathsf{H}, \hat{\mathsf{H}}$, and $\bar{\mathsf{H}}$, which are provided by the game in the standard lazy way using maps $h[\cdot], \hat{h}[\cdot]$, and $\bar{h}[\cdot]$, respectively. The set of corrupted parties is denoted by $\mathsf{Corrupted}$ and the set of queried messages is denoted by $\mathsf{Queried}$. Finally, the adversary outputs a forgery (m^*, σ^*) and the game outputs 1 if $\mathsf{m}^* \notin \mathsf{Queried}$, $|\mathsf{Corrupted}| \leq t$, and σ^* is a valid signature for m^*. We make three purely conceptual changes to the game. First, we will never keep the secret key share sk_i explicitly in the states $\mathsf{state}[sid, i]$ for users i in a session sid, although the scheme description would require this. This is without loss of generality, as the adversary only gets to see the states when it corrupts a user, and in this case it also gets sk_i. Second, we assume the adversary always queried $\mathsf{H}(\mathsf{m}^*)$ before outputting its forgery. Third, we assume that the adversary makes exactly t (distinct) corruption queries. These changes are without loss of generality and do not change the advantage of $\mathcal{A}$. Formally, one could build a wrapper adversary that internally runs $\mathcal{A}$, but makes a query $\mathsf{H}(\mathsf{m}^*)$ and enough corruption queries before terminating, and on every corruption query includes sk_i in the states before passing the result back to $\mathcal{A}$. Clearly, we have

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

The remainder of our proof is split into three parts. In the first part ($\mathbf{G}_1$-$\mathbf{G}_3$), we ensure that the game no longer needs secret key shares sk_i to compute $\mathsf{pk}_i^{(2)}$ in the signing oracle. Roughly, this is done by embedding shifted tags $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', g)$ into random oracle H for signing queries, and keeping random tags h for the query related to the forgery. In the second part ($\mathbf{G}_4$-$\mathbf{G}_{11}$), 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, sk_i is only needed when the adversary corrupts users. In the third part, we analyze $\mathbf{G}_{11}$. This is done by distinguishing two cases. One of the cases is bounded using a statistical argument. The other case is bounded using a reduction breaking the t-algebraic translation resistance of TLF. We now proceed with the details.

Game $\mathbf{G}_1$: In this game, we introduce a map $b[\cdot]$ that maps messages m to bits $b[\mathsf{m}] \in \{0,1\}$. Concretely, whenever a query $\mathsf{H}(\mathsf{m})$ is made for which the hash value is not yet defined, the game samples $b[\mathsf{m}]$ from a Bernoulli distribution $\mathcal{B}_\gamma$ with parameter $\gamma = 1/(Q_S+1)$. That is, $b[\mathsf{m}]$ is set to 1 with probability $1/(Q_S+1)$ and to 0 otherwise. The game aborts if $b[\mathsf{m}] = 1$ for some message m for which the signing oracle is called, or $b[\mathsf{m}^*] = 0$ for the forgery message m^*. Clearly, if no abort occurs, games $\mathbf{G}_0$ and $\mathbf{G}_1$ are the same. Further the view of $\mathcal{A}$ is independent of the map b. We obtain

$$\Pr[\mathbf{G}_1 \Rightarrow 1] = \gamma(1-\gamma)^{Q_S} \cdot \Pr[\mathbf{G}_0 \Rightarrow 1]$$

Now, we can use the fact $(1 - 1/x)^x \geq 1/4$ for all $x \geq 2$ and get

$$\gamma\,(1-\gamma)^{Q_S} = \frac{1}{Q_S+1}\left(1 - \frac{1}{Q_S+1}\right)^{Q_S} = \frac{1}{Q_S}\left(1 - \frac{1}{Q_S+1}\right)^{Q_S+1} \geq \frac{1}{4Q_S},$$

where the second equality follows from

$$\frac{1}{Q_S}\left(1 - \frac{1}{Q_S+1}\right) = \frac{1}{Q_S} - \frac{1}{Q_S(Q_S+1)} = \frac{(Q_S+1)-1}{Q_S(Q_S+1)} = \frac{Q_S}{Q_S(Q_S+1)} = \frac{1}{Q_S+1}.$$

In combination, we get

$$\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] \geq \frac{1}{4Q_S} \cdot \Pr\left[\mathbf{G}_0 \Rightarrow 1\right].$$

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, we change the way queries to random oracle H are answered. Namely, for a query $\mathsf{H}(\mathsf{m})$ for which the hash value $h[\mathsf{m}]$ is not yet defined, the game samples $h[\mathsf{m}] \overset{\$}{\leftarrow} \mathcal{T}$ as a random tag exactly as the previous game did. However, now, if $b[\mathsf{m}] = 0$, the game samples $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', g)$ and sets $h[\mathsf{m}] := h$. Further, it stores td in a map tr as $tr[\mathsf{m}] := \mathsf{td}$. Clearly, $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable by the ε_{t}-translatability of TLF. Concretely, one can easily see that

$$\left|\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \Rightarrow 1\right]\right| \leq Q_{\mathsf{H}}\varepsilon_{\mathsf{t}}.$$

Game $\mathbf{G}_3$: In this game, we change how the values $\mathsf{pk}_i^{(2)}$ are computed by the signing oracle. To recall, in the commitment phase of the signing protocol, the signing oracle for user $i \in [n]$ in $\mathbf{G}_2$ would compute the value $\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$, where $h = \mathsf{H}(\mathsf{m})$ and m is the message to be signed. Also, the value $\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$ is recomputed in the opening phase of the signing protocol and included in the output sent to the adversary. From $\mathbf{G}_3$ on, $\mathsf{pk}_i^{(2)}$ is computed differently, namely, as $\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m}], \mathsf{pk}_i)$. Observe that if the game did not abort, we know that $b[\mathsf{m}] = 0$ (see $\mathbf{G}_1$) and therefore h has been generated as $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', g)$ where $tr[\mathsf{m}] = \mathsf{td}$. Thus, it follows from the translatability of TLF, or more concretely from the translation completeness, that the view of $\mathcal{A}$ is not changed. We get

$$\Pr\left[\mathbf{G}_2 \Rightarrow 1\right] = \Pr\left[\mathbf{G}_3 \Rightarrow 1\right].$$

Game $\mathbf{G}_4$: In this game, we let the game abort if $(\mathsf{par}', g) \notin \mathsf{Reg}$, where Reg is the set from the regularity definition of TLF. By regularity of TLF, we have

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

Game $\mathbf{G}_5$: In this game, we change the signing oracle again. Specifically, we change the commitment and opening phase. Recall that until now, in the commitment phase for an honest party i in a signer set $S \subseteq [n]$ and message m, an element $r_i \overset{\$}{\leftarrow} \mathcal{D}$ is sampled and the party sends a commitment $\mathsf{com}_i := \hat{\mathsf{H}}(S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$ for $R_i^{(1)} := \mathsf{T}(g, r_i)$, $R_i^{(2)} := \mathsf{T}(h, r_i)$, and $\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m}], \mathsf{pk}_i)$. As before, h is defined as $h := \mathsf{H}(\mathsf{m})$. Later, in the opening phase, the party sends $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$. Now, we change this as follows: The signing oracle computes $\mathsf{pk}_i^{(2)}$ as in $\mathbf{G}_4$, but it does not compute $R_i^{(1)}, R_i^{(2)}$ and instead sends a random commitment $\mathsf{com}_i \overset{\$}{\leftarrow} \{0,1\}^{2\lambda}$ on behalf of party i. It also inserts an entry (S, i, com_i) into a list Sim that keeps track of these simulated commitments. If there is already an $(S', i') \neq (S, i)$ such that $(S', i', \mathsf{com}_i) \in \mathsf{Sim}$, then the game aborts. Note that there are two situations where the preimage of com_i has to be revealed. Namely, $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$ has to be given to the adversary in the opening phase, and whenever party i is corrupted the game needs to output r_i. To handle this, consider the opening phase or the case where party i is corrupted before it reaches the opening phase. Here, we let the game sample $r_i \overset{\$}{\leftarrow} \mathcal{D}$ and define $R_i^{(1)} := \mathsf{T}(g, r_i)$ and $R_i^{(2)} := \mathsf{T}(h, r_i)$. Then, the game checks if $\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] = \bot$. If it is not, the game aborts. Otherwise, it programs $\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] := \mathsf{com}_i$ and continues. That is, in the opening phase it would output $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$, and during a corruption, it would output r_i as part of its state. If a corruption occurs after the opening phase, then r_i has already been defined, and corruption is handled as before. Clearly, the view of $\mathcal{A}$ is only affected by this change if $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$ matches a previous query of $\mathcal{A}$ or the same commitment has been sampled by the game twice. The latter event occurs only with probability $Q_S^2/2^{2\lambda}$ by a union bound over all pairs of queries. To bound the former event, we use the regularity of TLF, which implies that $R_i^{(1)}$ is uniform over the range $\mathcal{R}$. Now, for each fixed pair of signing query and random oracle query, the random oracle query matches $R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}$ with probability at most $1/|\mathcal{R}|$. Thus, the event occurs only with probability $Q_S Q_{\hat{\mathsf{H}}}/2^{2\lambda}$. We get

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \frac{Q_S Q_{\hat{\mathsf{H}}}}{|\mathcal{R}|} + \frac{Q_S^2}{2^{2\lambda}}.$$

Game $\mathbf{G}_6$: In this game, we rule out collisions for random oracle $\hat{\mathsf{H}}$. Namely, the game aborts if there are $x \neq x'$ such that $\hat{h}[x] = \hat{h}[x'] \neq \bot$. Clearly, we have

$$|\Pr[\mathbf{G}_5 \Rightarrow 1] - \Pr[\mathbf{G}_6 \Rightarrow 1]| \leq \frac{Q_{\hat{\mathsf{H}}}^2}{2^{2\lambda}}.$$

Subsequent games will internally make use of an algorithm $\hat{\mathsf{H}}^{-1}$. On input y the algorithm searches for an x such that $\hat{h}[x] = y$. If no such x is found, or if multiple x are found, then the algorithm returns $\bot$. Otherwise, it returns x. Note that in the latter case the game would abort anyways, and so we can assume that if there is a preimage of y, then this preimage is uniquely determined by y.

Game $\mathbf{G}_7$: In this game, we introduce a list $\mathsf{Pending}$ and associated algorithms $\mathsf{UpdatePending}$ and $\mathsf{AddToPending}$ to manage this list. The algorithms are presented as pseudocode in Figure 11. Intuitively, the list keeps track of honest users i and signing sessions sid for which the game can not yet extract preimages of all commitments sent in the commitment phase. More precisely, the list contains a tuple $(sid, i, \mathcal{M}_1)$ if and only if the following two conditions hold:

- The opening phase oracle $\textsc{Sig}_1(sid, i, \mathcal{M}_1)$ has been called with valid inputs, i.e., for this query the game did not output $\bot$ due to $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 0$, and at that point the following was true: For every commitment com_j in $\mathcal{M}_1$ such that $(S, j, \mathsf{com}_j) \notin \mathsf{Sim}$, we have $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) \neq \bot$ and with $(S', k, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}) := \hat{\mathsf{H}}^{-1}(\mathsf{com}_j)$ we have $S' = S$ and $k = j$, where S is the signer set associated with sid.
- There is a commitment com_j in $\mathcal{M}_1$ such that $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot$.

To ensure that the list satisfies this invariant, we add a triple $(sid, i, \mathcal{M}_1)$ to $\mathsf{Pending}$ when the first condition holds. This is done by algorithm $\mathsf{AddToPending}$. Concretely, whenever $\mathcal{A}$ calls $\textsc{Sig}_1(sid, i, \mathcal{M}_1)$, the oracle returns $\bot$ in case $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 0$. If $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 1$, the game immediately calls $\mathsf{AddToPending}(sid, i, 1, \mathcal{M}_1)$, which checks the first condition of the invariant and inserts the tripe $(sid, i, 1, \mathcal{M}_1)$ into $\mathsf{Pending}$ if it holds. Then, the game continues the simulation of $\textsc{Sig}_1$ as before. Further, we invoke algorithm $\mathsf{UpdatePending}$ whenever the map $\hat{h}$ is changed, i.e., during queries to $\hat{\mathsf{H}}$, and in corruption and signing oracles (see $\mathbf{G}_5$). On every invocation, the algorithm does the following:

1. Initialize an empty list New.
2. Iterate trough all entries $(sid, i, \mathcal{M}_1)$ in $\mathsf{Pending}$, and do the following:
   (a) Check if the entry has to be removed because it is violating the invariant. That is, check if for all j in the signer set S associated with session sid, we have $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) \neq \bot$, where $\mathcal{M}_1 = (\mathsf{com}_j)_{j \in S}$. If this is not the case, skip this entry and keep it in $\mathsf{Pending}$.
   (b) We know that for all indices $j \in S$, the value $(S'_j, k_j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) = \hat{\mathsf{H}}^{-1}(\mathsf{com}_j)$ exists. Further, it must hold that $S'_j = S$ and $k_j = j$, as otherwise this entry would not have been added to $\mathsf{Pending}$ in the first place. Remove the entry from $\mathsf{Pending}$, and determine the combined nonces and secondary public key

$$R^{(1)} = \sum_{j \in S} R_j^{(1)}, \quad R^{(2)} = \sum_{j \in S} R_j^{(2)}, \quad \mathsf{pk}^{(2)} = \sum_{j \in S} \ell_{j,S} \mathsf{pk}_j^{(2)}.$$

   (c) Let m be the message associated with the session sid.

(d) If $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m}) \notin \mathsf{New}$ but $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \neq \perp$, abort the execution of the entire game (see bad event Defined below).

(e) Otherwise, sample $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \xleftarrow{\$} \mathcal{S}$ and insert the tuple $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})$ into New.

To summarize, this algorithm removes all entries violating the invariant from the list Pending. For each such entry that is removed, the algorithm computes the combined nonces $R^{(1)}, R^{(2)}$ and secondary public key $\mathsf{pk}^{(2)}$. Roughly, it aborts the execution, if random oracle $\bar{\mathsf{H}}$ for these inputs is already defined. List New ensures that the abort is not triggered if the algorithm itself programmed $\bar{h}$ in a previous iteration within the same invocation. In addition to algorithm UpdatePending, we introduce the following events, on which the game aborts its execution:

- Event BadQuery: This event occurs, if for a random oracle query to $\hat{\mathsf{H}}$ for which the hash value is not yet defined and freshly sampled as $\mathsf{com} \xleftarrow{\$} \{0,1\}^{2\lambda}$, there is an entry $(sid, i, \mathcal{M}_1)$ in Pending such that com is in $\mathcal{M}_1$.
- Event Defined: This event occurs, if the execution is aborted during algorithm UpdatePending.

For shorthand notation, we set $\mathsf{Bad} := \mathsf{BadQuery} \vee \mathsf{Defined}$. The probability of BadQuery can be bounded as follows: Fix a random oracle query to $\hat{\mathsf{H}}$ for which the hash value is not yet defined. Fix an entry $(sid, i, \mathcal{M}_1)$. Note that over the entire game, there are at most Q_S of these entries. Further, fix an index $j \in [t+1]$. The probability that com collides with the jth entry of $\mathcal{M}_1$ is clearly at most $1/2^{2\lambda}$. With a union bound over all triples of queries, entries, and indices, we get that the probability of BadQuery is at most $Q_{\hat{\mathsf{H}}} Q_S (t+1)/2^{2\lambda}$. Next, we bound the probability of Defined assuming BadQuery does not occur. Under this assumption, one can easily observe that when an entry is removed from list Pending and $R^{(1)} = \sum_{j \in S} R_j^{(1)}$ is the combined first nonce, then there is an $j^* \in S$ such that the game sampled $R_{j^*}^{(1)}$ just before invoking algorithm UpdatePending. Precisely, it must have set $R_{j^*}^{(1)} := \mathsf{T}(g, r)$ for some random $r \xleftarrow{\$} \mathcal{D}$. By regularity of TLF, this means $R_{j^*}^{(1)}$ is uniform over $\mathcal{R}$, and this means that the combined first nonce $R^{(1)}$ is also uniform. Thus for any fixed entry of in Pending, the probability that $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}]$ is already defined when the entry is removed, is at most $Q_{\hat{\mathsf{H}}}/|\mathcal{R}|$. With a union bound over all entries we can now bound the probability of Defined by $Q_{\hat{\mathsf{H}}} Q_S/|\mathcal{R}|$. In combination, we get

$$\Pr[\mathsf{Bad}] \leq \Pr[\mathsf{BadQuery}] + \Pr[\mathsf{Defined} \mid \neg \mathsf{BadQuery}] \leq \frac{Q_{\hat{\mathsf{H}}} Q_S (t+1)}{2^{2\lambda}} + \frac{Q_{\hat{\mathsf{H}}} Q_S}{|\mathcal{R}|}.$$

and thus

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

Game $\mathbf{G}_8$: In this game, we change algorithm UpdatePending. Specifically, we change what we insert into list New. Recall from the previous game that when we removed an entry $(sid, i, \mathcal{M}_1)$ from Pending, we aborted the game if $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m}) \notin \mathsf{New}$ but $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \neq \perp$. Otherwise, we inserted tuples $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})$. Now, we instead abort if $(S, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m}) \notin \mathsf{New}$ but $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \neq \perp$, and otherwise insert $(S, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})$, where S is the signer set associated with session *sid*. One can see that the two games can only differ if for two entries $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}_1')$ that are removed from Pending in the same invocation of UpdatePending, the signer sets S and S' differ but the respective tuples $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})$ and $(R'^{(1)}, R'^{(2)}, \mathsf{pk}'^{(2)}, \mathsf{m}')$ are the same and $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \neq \perp$. In this case, game $\mathbf{G}_8$ would abort, but game $\mathbf{G}_7$ would not. We argue that this can not happen: Assume that two entries $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}_1')$ with associated signer sets S and S' are removed from Pending. Then, we know that algorithm UpdatePending has been invoked because the game programmed $\hat{h}$ at some point, say $\hat{h}[S_*, j_*, R_*^{(1)}, R_*^{(2)}, \mathsf{pk}_*^{(2)}] := \mathsf{com}_*$, such that com_* is in both $\mathcal{M}_1$ and $\mathcal{M}_1'$. Thus, the algorithm only removes the entry $(sid, i, \mathcal{M}_1)$ from the list if the first component of $\hat{\mathsf{H}}^{-1}(\mathsf{com}_*)$ is S, i.e., if $S_* = S$. Similarly, it only removes the entry $(sid', i', \mathcal{M}_1')$ if the first the first component of $\hat{\mathsf{H}}^{-1}(\mathsf{com}_*)$ is S', i.e., if $S_* = S'$. Thus, it only removes both if $S = S_* = S'$. With that, we have

$$\Pr[\mathbf{G}_7 \Rightarrow 1] = \Pr[\mathbf{G}_8 \Rightarrow 1].$$

Game $\mathbf{G}_9$: We introduce two more algorithms, presented as pseudocode in Figure 11. Intuitively, these allow us to group tuples of the form $(sid, i, \mathcal{M}_1)$ that have been inserted into list $\mathsf{Pending}$ into equivalence classes. To be clear, the relation is defined on all triples in $\mathsf{Pending}$ and on all triples that already have been removed from $\mathsf{Pending}$, but not on any other entries. The intuition, roughly, is that such triples lead to the same combined nonces if and only if they are in the same equivalence class. The effect of this is will be that we know the challenge just from the tuple $(sid, i, \mathcal{M}_1)$. We now turn to the details. We introduce an algorithm $\mathsf{Equivalent}$ that takes as input two triples $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}'_1)$ and decides whether they are equivalent as follows:

1. Let S, S' and m, m' be the signer sets and messages associated with sessions sid and sid', respectively. If $S \neq S'$ or $\mathsf{m} \neq \mathsf{m}'$, the triples are not equivalent.

2. Thus, assume $S = S'$ and write $\mathcal{M}_1 = (\mathsf{com}_j)_{j \in S}$ and $\mathcal{M}'_1 = (\mathsf{com}'_j)_{j \in S}$. Let $F \subseteq S$ (resp. $F' \subseteq S'$) be the set of indices $j \in S$ (resp $j \in S'$) such that $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot$ (resp. $\hat{\mathsf{H}}^{-1}(\mathsf{com}'_j) = \bot$). If $(\mathsf{com}_j)_{j \in F} \neq (\mathsf{com}'_j)_{j \in F'}$, then the triples are not equivalent.

3. Define $\bar{F} := S \setminus F$ and $\bar{F}' := S \setminus F'$. For each $j \in \bar{F}$, we know that the value $(\tilde{S}_j, k_j, R_j'^{(1)}, R_j'^{(2)}, \mathsf{pk}_j'^{(2)}) = \hat{\mathsf{H}}^{-1}(\mathsf{com}'_j)$ exists. Similarly, for each $j \in \bar{F}'$, we know that the value $(\tilde{S}'_j, k'_j, R_j'^{(1)}, R_j'^{(2)}, \mathsf{pk}_j'^{(2)}) = \hat{\mathsf{H}}^{-1}(\mathsf{com}'_j)$ exists. With these, we can define partially combined nonces and secondary keys

$$\begin{aligned} &\bar{R}^{(1)} := \textstyle\sum_{j \in \bar{F}} R_j^{(1)}, \quad &&\bar{R}^{(2)} := \textstyle\sum_{j \in \bar{F}} R_j^{(2)} \quad &&\bar{\mathsf{pk}}^{(2)} := \textstyle\sum_{j \in \bar{F}} \ell_{j,S} \mathsf{pk}_j^{(2)} \\ &\bar{R}'^{(1)} := \textstyle\sum_{j \in \bar{F}'} R_j'^{(1)}, \quad &&\bar{R}'^{(2)} := \textstyle\sum_{j \in \bar{F}'} R_j'^{(2)} \quad &&\bar{\mathsf{pk}}'^{(2)} := \textstyle\sum_{j \in \bar{F}'} \ell_{j,S} \mathsf{pk}_j'^{(2)}. \end{aligned}$$

   The triples are not equivalent, if $(\bar{R}^{(1)}, \bar{R}^{(2)}, \bar{\mathsf{pk}}^{(2)}) \neq (\bar{R}'^{(1)}, \bar{R}'^{(2)}, \bar{\mathsf{pk}}'^{(2)})$. Otherwise, they are equivalent.

In summary, two triples are equivalent if their signer sets, messages, partially combined nonces and secondary public keys, and remaining commitments match. It is clear that at any fixed point in time during the experiment, this is indeed an equivalence relation. In the following two claims, we argue that this relation is preserved over time. For that, we first make some preliminary observations, using notation as in the definition of equivalence above:

1. The equivalence relation can potentially only change when oracle $\hat{\mathsf{H}}$ is updated during queries to $\textsc{Sig}_1$ (i.e., the opening phase) or during corruption queries, which may make the sets F and F' change. This is because triples are only inserted into $\mathsf{Pending}$ if the only commitments without preimages are simulated, and the preimages of these are only set in such calls (see $\mathbf{G}_7$).

2. The sets F and F' can only get smaller over time, as we assume that no collisions occur.

3. When the oracle is programmed during such calls, say by setting $\hat{h}[S_*, j_*, R_*^{(1)}, R_*^{(2)}, \mathsf{pk}_*^{(2)}] := \mathsf{com}_*$, then it must hold that $(S_*, j_*, \mathsf{com}_*) \in \mathsf{Sim}$. In particular, if in this case some j is removed from F (or F') because com_j (or com'_j) now has a preimage, then it must hold that $\mathsf{com}_* = \mathsf{com}_j$ and $j_* = j$. This is because otherwise, if $j \neq j_*$, then we would have $(\tilde{S}, j, \mathsf{com}_*) \in \mathsf{Sim}$ for some $\tilde{S}$ (because the entry was added to $\mathsf{Pending}$) and $(S_*, j_*, \mathsf{com}_*) \in \mathsf{Sim}$, and such a collision was ruled out in $\mathbf{G}_5$.

4. Again, assume that the oracle is programmed during such calls by setting $\hat{h}[S_*, j_*, R_*^{(1)}, R_*^{(2)}, \mathsf{pk}_*^{(2)}]$ $:= \mathsf{com}_*$. Now, assume that both F and F' change. Then, we know (because of the previous observation), that the same $j = j_*$ is removed from both F and F', and $\mathsf{com}_j = \mathsf{com}_* = \mathsf{com}'_j$ is removed from both $(\mathsf{com}_j)_{j \in F}$ and $(\mathsf{com}'_j)_{j \in F'}$. Thus, these lists are the same before the update if and only if they are the same after the update.

5. In the setting of the previous observation, denote the point in time before the update as t_0, and the point in time after the update as t_1. Further, denote the associated partially combined nonces and secondary public keys at time t_b for $b \in \{0, 1\}$ by

$$\bar{R}_b^{(1)},\ \bar{R}_b^{(2)},\ \bar{\mathsf{pk}}_b^{(2)},\ \text{and}\ \bar{R}_b'^{(1)},\ \bar{R}_b'^{(2)},\ \bar{\mathsf{pk}}_b'^{(2)}.$$

Now, we observe that

$$\bar{R}_1^{(1)} = \bar{R}_0^{(1)} + R_*^{(1)}, \quad \bar{R}_1^{(2)} = \bar{R}_0^{(2)} + R_*^{(2)}, \quad \bar{\mathsf{pk}}_1^{(2)} = \bar{\mathsf{pk}}_0^{(2)} + \ell_{j_*,S_*}\mathsf{pk}_*^{(2)}.$$

The same holds for $\bar{R}_b^{\prime(1)}$, $\bar{R}_b^{\prime(2)}$, and $\bar{\mathsf{pk}}_b^{\prime(2)}$. Therefore, we see that

$$\begin{aligned} & (\bar{R}_0^{(1)}, \bar{R}_0^{(2)}, \bar{\mathsf{pk}}_0^{(2)}) = (\bar{R}_0^{\prime(1)}, \bar{R}_0^{\prime(2)}, \bar{\mathsf{pk}}_0^{\prime(2)}) \\ \text{if and only if } & (\bar{R}_1^{(1)}, \bar{R}_1^{(2)}, \bar{\mathsf{pk}}_1^{(2)}) = (\bar{R}_1^{\prime(1)}, \bar{R}_1^{\prime(2)}, \bar{\mathsf{pk}}_1^{\prime(2)}). \end{aligned}$$

Now, we show that the equivalence relation does not change over time, using our notation from above and the observations we made.
Equivalence Claim 1. If two triples $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}'_1)$ are equivalent at some point in time, then they stay equivalent for the rest of the game.
Proof of Equivalence Claim 1. Both signer set and message do not change over time. For the other components that determine whether the triples are equivalent, we consider two cases: Either, on an update of $\hat{\mathsf{H}}$, both do not change. In this case the triples trivially stay equivalent. In the other case, both of them change, as the lists $(\mathsf{com}_j)_{j\in F}$ and $(\mathsf{com}'_j)_{j\in F'}$ are the same before the update. Now, it easily follows from our last observation above that the triples stay equivalent.
Equivalence Claim 2. If two triples $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}'_1)$ are not equivalent at some point in time, then the probability that they become equivalent later is negligible. Concretely, if $\mathsf{Converge}$ is the event that any two non-equivalent triples become equivalent at some point in time, then

$$\Pr[\mathsf{Converge}] \leq \frac{Q_S^2(Q_S + t)}{|\mathcal{R}|}.$$

Proof of Equivalence Claim 2. Clearly, if $\mathsf{m} \neq \mathsf{m}'$ or $S \neq S'$, then the triples will stay non-equivalent. Now, consider an update of $\hat{\mathsf{H}}$ that is caused by a query to $\textsc{Sig}_1$ or the corruption oracle and will potentially change the equivalence relation. We consider two cases: In the first case, the lists $(\mathsf{com}_j)_{j\in F}$ and $(\mathsf{com}'_j)_{j\in F'}$ are the same before the update. In this case, they either do not change, in which case the triples trivially stay non-equivalent, or they both change, in which case it follows from our last observation above that they stay non-equivalent. In the second case, the lists $(\mathsf{com}_j)_{j\in F}$ and $(\mathsf{com}'_j)_{j\in F'}$ are different before the update. If they stay different after the update, the triples stay non-equivalent. If they become the same after the update, this means that an entry was removed from only one of them, say $j = j_*$ from F and thus $\mathsf{com}_j = \mathsf{com}_*$ from $(\mathsf{com}_j)_{j\in F}$. For this case, use notation $\bar{R}_b^{(1)}$ and $\bar{R}_b^{\prime(1)}$ as in the last last observation above and notice that $\bar{R}_1^{\prime(1)} = \bar{R}_0^{\prime(1)}$ because $(\mathsf{com}'_j)_{j\in F'}$ is not changed during the update. On the other hand, $(\mathsf{com}_j)_{j\in F}$ is changed by the update and we have $\bar{R}_1^{(1)} = \bar{R}_0^{(1)} + R_*^{(1)}$. Thus, if the triples become equivalent, we must have

$$\bar{R}_0^{\prime(1)} = \bar{R}_1^{\prime(1)} = \bar{R}_1^{(1)} = \bar{R}_0^{(1)} + R_*^{(1)}.$$

Notice that $R_*^{(1)}$ is sampled in the signing or corruption oracle by sampling some $r_* \xleftarrow{\$} \mathcal{D}$ and setting $R_*^{(1)} = \mathsf{T}(g, r_*)$. Thus, $R_*^{(1)}$ is uniformly distributed over $\mathcal{R}$ by the regularity of TLF and independent of $\bar{R}_0^{\prime(1)}$ and $\bar{R}_0^{(1)}$, which means that this equation holds with probability at most $1/|\mathcal{R}|$. Taking a union bound over all pairs of triples and all queries to the signing oracle and the corruption oracle, the claim follows.

With our equivalence relation at hand, we introduce an algorithm $\mathsf{GetChallenge}$ that behaves as a random oracle on equivalence classes. That is, it assigns each class a random challenge $c \xleftarrow{\$} \mathcal{S}$ in a lazy manner. More precisely, it gets as input a triple $(sid, i, \mathcal{M}_1)$ and checks if a triple in the same equivalence class[5] is already assigned a challenge c. This is done using algorithm $\mathsf{Equivalent}$. If so, it returns this challenge c. If not, it assigns a random challenge $c \xleftarrow{\$} \mathcal{S}$ to the triple $(sid, i, \mathcal{M}_1)$.

These two new algorithms are used in the following way: Recall that in previous games, algorithm $\mathsf{UpdatePending}$ would program $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \xleftarrow{\$} \mathcal{S}$ whenever an entry $(sid, i, \mathcal{M}_1)$ is removed

[5]It is essential for this algorithm that we have shown that equivalence classes are preserved over time. Otherwise, the behavior of this algorithm would be ambiguous.

from $\mathsf{Pending}$ and no abort occurs, where $\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}$ are the corresponding secondary public keys, combined nonces, and messages. Now, instead of sampling $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}]$ at random, the algorithm sets $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] := \mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$. We need to argue that this way of programming the random oracle does not change the view of the adversary. Concretely, all we need to argue is that two different inputs $x \neq x'$ to random oracle $\bar{\mathsf{H}}$ get independently sampled outputs. Clearly, it is sufficient to consider inputs of the form

$$x = (\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}), \quad x' = (\mathsf{pk}, \mathsf{pk}^{'(2)}, R^{'(1)}, R^{'(2)}, \mathsf{m}'),$$

which both are covered by the newly introduced programming in algorithm $\mathsf{UpdatePending}$. Let $(sid, i, \mathcal{M}_1)$ be the entry removed from $\mathsf{Pending}$ associated with x and $(sid', i', \mathcal{M}_1')$ be the entry removed from $\mathsf{Pending}$ associated with x'. Consider the point in time where the second entry, say $(sid', i', \mathcal{M}_1')$ has been removed. One can see that the outputs $\bar{\mathsf{H}}(x)$ and $\bar{\mathsf{H}}(x')$ are independent, unless at this point in time $(sid, i, \mathcal{M}_1)$ and $(sid', i', \mathcal{M}_1')$ were equivalent. However, by definition of equivalence (algorithm $\mathsf{Equivalent}$), them being equivalent would mean that $\mathsf{m} = \mathsf{m}'$ and $(\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}) = (\mathsf{pk}^{'(2)}, R^{'(1)}, R^{'(2)})$, as the sets F and F' are both empty because both entries have been removed from $\mathsf{Pending}$. Thus, we would have $x = x'$. This shows that the distribution of random oracle outputs does not change, and so we have

$$\Pr[\mathbf{G}_8 \Rightarrow 1] = \Pr[\mathbf{G}_9 \Rightarrow 1].$$

Game $\mathbf{G}_{10}$: In this game, we change the signing oracle and corruption oracle. Roughly, we use an honest-verifier zero-knowledge-style simulation to simulate signing without secret keys. Intuitively, we can do that, because now we know the challenge already in the opening phase before fixing nonces. More precisely, recall that until now, signers in the opening phase, i.e., on a query $\textsc{Sig}_1(sid, i, \mathcal{M}_1)$, sampled a random $r_i \xleftarrow{\$} \mathcal{D}$ and set $R_i^{(1)} := \mathsf{T}(g, r_i)$ and $R_i^{(2)} := \mathsf{T}(h, r_i)$. Later, in the response phase, the signer sent $s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$ where $c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$ and $\mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}$ are the combined secondary public key and nonces. Additionally, when the signer is corrupted, it has to send r_i as part of its state. We change this as follows: In the opening phase, consider two cases: First, if $(sid, i, \mathcal{M}_1)$ has not been added to the list $\mathsf{Pending}$, then the signer sets $c := 0$. Observe that in this case, we can assume that the signer never reaches the response phase for this session due to our changes in $\mathbf{G}_6$ and $\mathbf{G}_7$. Otherwise, it sets $\tilde{c} := \mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$. In both cases, the signer samples $s_i \xleftarrow{\$} \mathcal{D}$ and sets $R_i^{(1)} := \mathsf{T}(g, s_i) - \tilde{c} \cdot \ell_{i,S} \cdot \mathsf{pk}_i$ and $R_i^{(2)} := \mathsf{T}(h, s_i) - \tilde{c} \cdot \ell_{i,S} \cdot \mathsf{pk}_i^{(2)}$. Later, when the signer has to output something in the response phase, it outputs the s_i that it sampled in the opening phase. Further, when the signer is corrupted after the opening phase, it sets $r_i := s_i - \tilde{c} \cdot \ell_{i,S} \cdot \mathsf{sk}_i$. To argue indistinguishability, we need to show that $\tilde{c}$ and $c = \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$ are the same. This is established as follows:

1. When the signer is queried in the response phase and does not return $\perp$, we know that the entry $(sid, i, \mathcal{M}_1)$ has been removed from $\mathsf{Pending}$.
2. When it was removed from the list, the combined nonce and secondary public key that have been computed are exactly $R^{(1)}, R^{(2)}$, and $\mathsf{pk}^{(2)}$.
3. Therefore, in the invocation of $\mathsf{UpdatePending}$ in which the entry was removed from the list, one of two events happened:
   (a) Either $\bar{h}$ has been programmed as $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] := \mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$;
   (b) Or, $\bar{h}$ has been programmed as $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] := \mathsf{GetChallenge}(sid', i', \mathcal{M}_1')$ for some triple $(sid', i', \mathcal{M}_1')$ with the same associated signer set S (see $\mathbf{G}_8$) and message m. In this case, we know that $(sid', i', \mathcal{M}_1')$ is equivalent to $(sid, i, \mathcal{M}_1)$ and therefore $\mathsf{GetChallenge}(sid', i', \mathcal{M}_1')$ returned the same as what the query $\mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$ would have returned at that point.
4. Thus, we only need to argue that the output of $\mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$ did not change over time. This follows from our claims about the stability of equivalence classes over time, assuming event $\mathsf{Converge}$ does not occur.

We get

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

<u>**Game $\mathbf{G}_{11}$:**</u> We change the game by no longer assuming 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 game $\mathbf{G}_{11}$ outputs 1. Before turning to that, we emphasize the main property we have established via our changes: We do not longer need secret key shares sk_i to simulate the signer oracle. We only need them on corruption queries. Now, we bound the probability that game $\mathbf{G}_{11}$ outputs 1 by considering two events, depending on the final forgery (m^*, σ^*) with $\sigma^* = (\mathsf{pk}^{*(2)}, c^*, s^*)$:

- Event $\mathsf{Orthogonal}$: This event occurs, if $\mathbf{G}_{11}$ outputs 1, and there is no $x_0 \in \mathcal{D}$ such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$, where $h^* = \mathsf{H}(\mathsf{m}^*)$.
- Event $\mathsf{Parallel}$: This event occurs, if $\mathbf{G}_{11}$ outputs 1, and there is a $x_0 \in \mathcal{D}$ such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$, where $h^* = \mathsf{H}(\mathsf{m}^*)$.

Clearly, we have

$$\Pr[\mathbf{G}_{11} \Rightarrow 1] \leq \Pr[\mathsf{Orthogonal}] + \Pr[\mathsf{Parallel}].$$

We bound the probability of these events separately. For event $\mathsf{Orthogonal}$, we make use of Lemma 1. Concretely, we sketch a reduction that runs in the game specified in Lemma 1, such that the event bounded in Lemma 1 occurs if event $\mathsf{Orthogonal}$ occurs and some guesses of the reduction were correct. Namely, the reduction gets as input parameters par' for TLF. It samples $i^* \xleftarrow{\$} [Q_{\bar{\mathsf{H}}}]$ and simulates $\mathbf{G}_{11}$ for $\mathcal{A}$ except for the i^*th random oracle query to $\bar{\mathsf{H}}$. Let that query be $\bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$. The reduction aborts if the hash value is already defined. Otherwise, the reduction outputs its state, g, $h := \mathsf{H}(\mathsf{m}), X_1 := \mathsf{pk}, X_2 := \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}$ to the game from Lemma 1, receiving a challenge $c \in \mathcal{S}$ in return. It programs $\bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}) := c$ and continues the simulation. Later, when $\mathcal{A}$ outputs its forgery, the reduction aborts if the query defining c^* was not the i^*th query to $\bar{\mathsf{H}}$. Otherwise, it outputs s^* to the game from Lemma 1. Note that one of the random oracle queries to $\bar{\mathsf{H}}$ (possibly the one made by the game during verification) has to be the one defining c^*. Especially, the random oracle is not reprogrammed at that position because $\mathcal{A}$ is not allowed to make a signing query for m^*. Also, it is clear that if $\mathsf{Orthogonal}$ occurs and the guess was correct, then the event in Lemma 1 occurs. As the view of $\mathcal{A}$ is independent of i^* until it terminates, we have

$$\Pr[\mathsf{Orthogonal}] \leq \frac{Q_{\bar{\mathsf{H}}}}{|\mathcal{S}|}.$$

Next, we bound the probability of event $\mathsf{Parallel}$. For that, we describe a reduction $\mathcal{B}$ against the t-algebraic translation resistance of TLF:

1. $\mathcal{B}$ gets as input parameters par' for TLF, tags g, h, and images $X_0, \ldots, X_t \in \mathcal{R}$.
2. $\mathcal{B}$ simulates game $\mathbf{G}_{11}$ for $\mathcal{A}$, with the following changes
   - $\mathcal{B}$ sets up the key in a different way: Namely, it sets $\mathsf{pk} := \mathsf{pk}_0 := X_0$, and it sets $\mathsf{pk}_i := X_i$ for each $i \in [t]$. Further, let $S_0 := \{0\} \cup [t]$. $\mathcal{B}$ sets $\mathsf{pk}_i := \sum_{j \in S_0} \ell_{j,S_0}(i)\mathsf{pk}_j$ for each $i \in [n] \setminus S_0$. This means that $\mathcal{B}$ is not aware of the associated secret keys sk_i. Note that the public keys are distributed exactly as in $\mathbf{G}_{11}$.
   - $\mathcal{B}$ provides all random oracles as in $\mathbf{G}_{11}$, except for random oracle H. Namely, for this oracle, instead of sampling $h[\mathsf{m}]$ at random when $b[\mathsf{m}] = 1$, $\mathcal{B}$ samples $(h', \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', h)$ and sets $h[\mathsf{m}] := h'$ and $tr[\mathsf{m}] := \mathsf{td}$.
   - $\mathcal{B}$ provides the signing oracles as in $\mathbf{G}_{11}$. Observe that $\mathcal{B}$ does not need any secret keys for that.
   - Whenever $\mathcal{A}$ queries the corruption oracle to corrupt user i, $\mathcal{B}$ queries $x_i := \textsc{Inv}(\ell_{0,S_0}(i), \ldots, \ell_{t,S_0}(i))$. Then, it sets $\mathsf{sk}_i := x_i$ and continues the simulation of the corruption as in $\mathbf{G}_{11}$. It is clear that sk_i is correctly distributed. Further, $\mathcal{B}$ queries its oracle exactly t times because $\mathcal{A}$ corrupts exactly t parties (see $\mathbf{G}_0$).

3. Finally, when $\mathcal{A}$ outputs its forgery (m^*, σ^*) with $\sigma^* = (\mathsf{pk}^{*(2)}, c^*, s^*)$ and $\mathbf{G}_{11}$ outputs 1 (which can be efficiently checked by $\mathcal{B}$), the reduction $\mathcal{B}$ retrieves $\mathsf{td}^* := tr'[\mathsf{m}^*]$. This entry exists because of the change in $\mathbf{G}_1$. Then, it computes $X_0' := \mathsf{InvTranslate}(\mathsf{td}^*, \mathsf{pk}^{*(2)})$ for $h^* := \mathsf{H}(\mathsf{m}^*)$.
4. To recall, $\mathsf{Corrupted} \subseteq [n]$ is the set of parties $i \in [n]$ such that $\mathcal{A}$ corrupted party i. By our assumption from $\mathbf{G}_0$, we know that $|\mathsf{Corrupted}| = t$. The reduction sets $S^* := \mathsf{Corrupted} \cup \{0\}$, which has size $t{+}1$. Further, it sets $X_i' := \mathsf{T}(h, x_i)$ for each $i \in \mathsf{Corrupted}$ and $X_i' := \sum_{j \in S^*} \ell_{j,S^*}(i) X_j'$ for each $i \in [n] \setminus \mathsf{Corrupted}$. It outputs $X_0', \ldots, X_t'$ to the algebraic translation resistance game.

Clearly, the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$. Further the only difference between the view of $\mathcal{A}$ in game $\mathbf{G}_{11}$ and in the simulation provided by $\mathcal{B}$ is the distribution of random oracle H. It follows from the ε_{t}-translatability of TLF and a union bound over all queries that this changes the probability of event $\mathsf{Parallel}$ by at most $Q_{\mathsf{H}}\varepsilon_{\mathsf{t}}$. Now, it remains to argue that if event $\mathsf{Parallel}$ occurs, then $\mathcal{B}$ breaks algebraic translation resistance. So, assume that $\mathsf{Parallel}$ occurs. We have to argue that for each $i \in \{0\} \cup [t]$, there is a z_i such that $\mathsf{T}(g, z_i) = X_i$ and $\mathsf{T}(h, z_i) = X_i'$. First, for all $i \in \mathsf{Corrupted}$ this holds by construction. For $i = 0$, we know (because $\mathsf{Parallel}$) that there is an x_0 such that $\mathsf{T}(g, x_0) = \mathsf{pk}$ and $\mathsf{T}(h^*, x_0) = \mathsf{pk}^{*(2)}$ for $h^* = \mathsf{H}(\mathsf{m}^*)$. Now, we know that $X_0' = \mathsf{InvTranslate}(\mathsf{td}^*, \mathsf{pk}^{*(2)}) = \mathsf{T}(h, x_0)$, by translation completeness. Thus, it remains to show that the property holds for all $i \in [t]$ with $i \notin \mathsf{Corrupted}$. For these, we have

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

Further, we have

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

by construction. With this, we showed that

$$\Pr\left[\mathsf{Parallel}\right] \leq Q_{\mathsf{H}}\varepsilon_{\mathsf{t}} + \mathsf{Adv}^{t\text{-}\mathsf{A}\text{-}\mathsf{TRAN}\text{-}\mathsf{RES}}_{\mathcal{B},\mathsf{TLF}}(\lambda).$$

This finishes the proof. □

4 Instantiations

In this section, we instantiate our threshold signature scheme by providing concrete tagged linear function families. Our first instantiation is based on a one-more variant of the CDH assumption, and our second instantiation is based on DDH. The advantage of the latter instantiation is avoiding an interactive assumption, while the former is slightly more efficient.

4.1 Instantiation from (Algebraic) One-More CDH

We can instantiate the tagged linear function family by mapping a tag $h \in \mathbb{G}$ and a domain element $x \in \mathbb{Z}_p$ to $h^x \in \mathbb{G}$. Regularity and translatability are easy to show, and algebraic translation resistance follows from an algebraic one-more variant of CDH. We define it next.

Definition 7 (AOMCDH Assumption)**.** Let GGen be a group generation algorithm. That is, on input 1^λ, $\mathsf{GGen}(1^\lambda)$ outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p and a generator g if $\mathbb{G}$, and a description of the group operation. Consider the game **AOMCDH** in Figure 4. 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$^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
01 $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$
02 $h \xleftarrow{\$} \mathbb{G}, \quad x_0, \ldots, x_t \xleftarrow{\$} \mathbb{Z}_p$
03 for $i \in \{0\} \cup [t]$ : $X_i := g^{x_i}$
04 $(X_i')_{i=0}^t \leftarrow \mathcal{A}^{\textsc{Inv}}(\mathbb{G}, g, p, h, (X_i)_{i=0}^t)$
05 if $\forall i \in \{0\} \cup [t]$ $X_i' = h^{x_i}$ : return 1
06 return 0

Oracle $\textsc{Inv}(\alpha_0, \ldots, \alpha_t)$
07 if $q \geq t$ : return $\perp$
08 $q := q + 1$
09 $x := \sum_{i=0}^t \alpha_i x_i$
10 return $x$
```

Figure 4: The game **AOMCDH** from the definition of the AOMCDH assumption for an adversary $\mathcal{A}$.

Note that our one-more variant of CDH is different from the variant introduced in [BFP21] in the sense that the adversary has an algebraic DLOG oracle instead of a CDH oracle. To get some confidence in our AOMCDH assumption, we sketch that it is implied by the well-known algebraic one-more DLOG (AOMDL) assumption [NRS21] in the algebraic group model (AGM) [FKL18]. Note that Bauer et al. [BFP21] gave a bound for one-more DLOG in the generic group model (GGM) [Sho97]. As an immediate consequence, the advantage of any adversary against AOMCDH is bounded by the same, namely, by $\Theta\left((t^2/(p-t^2)) + (1/p)\right)$.

Lemma 2 (Informal)**.** *The* AOMCDH *assumption is implied by the algebraic one-more* DLOG *assumption in the algebraic group model.*

Sketch. We only sketch the proof. First, we recall the AOMCDH game for an algebraic adversary $\mathcal{A}$:

1. The game generates group parameters $(\mathbb{G}, g, p)$ and samples $h \xleftarrow{\$} \mathbb{G}$ and $\mathbf{x} = (x_i)_{i=0}^t \xleftarrow{\$} \mathbb{Z}_p^{t+1}$. We let $\gamma \in \mathbb{Z}_p$ be such that $h = g^\gamma$.
2. The game defines $X_i := g^{x_i}$ for all i and runs $\mathcal{A}$ on input $\mathbb{G}, g, p, h, (X_i)_{i=0}^t$ with t-time access to an oracle $\textsc{Inv}$ which on input $\alpha_0, \ldots, \alpha_t$ outputs $\sum_{i=0}^t \alpha_i x_i$.
3. When $\mathcal{A}$ terminates, it outputs $t+1$ group elements $(X_i')_{i=0}^t$ and wins the game if $X_i' = h^{x_i}$ for all i. As $\mathcal{A}$ is algebraic, we can assume that it also outputs elements $\mathbf{a} = (a_i)_i \in \mathbb{Z}_p^{t+1}$, $\mathbf{b} = (b_i)_i \in \mathbb{Z}_p^{t+1}$, and $\mathbf{C} = (C_{i,j})_{i,j} \in \mathbb{Z}_p^{(t+1)\times(t+1)}$ such that for all i we have
$$X_i' = g^{a_i} \cdot h^{b_i} \cdot \prod_{j=0}^t X_j^{C_{i,j}}.$$
Reading these equations in the exponent of g and assuming that $\mathcal{A}$ wins, this means that
$$\gamma \mathbf{x} = \mathbf{a} + \gamma \mathbf{b} + \mathbf{C}\mathbf{x}.$$

We want to argue that $\mathcal{A}$ can not win the game. For that, we will distinguish two cases and bound the probability of these using the AOMDL and the plain DLOG assumption, respectively.
First Case: $\mathbf{b} = \mathbf{x}$. In this case, it is clear that a reduction can solve AOMDL. Before we sketch the reduction, we informally recall the AOMDL game, as present in recent works, e.g., [NRS21]. The game is exactly[6] as the AOMCDH game, with two modifications: (1) there is no element h, and (2) the winning condition asks the adversary to outputs the x_i instead of group elements X_i'. Now, our reduction is as follows. It gets as input $(\mathbb{G}, g, p)$ and $(X_i)_{i=0}^t$, it samples $h \xleftarrow{\$} \mathbb{G}$, and runs $\mathcal{A}$ on input $\mathbb{G}, g, p, h, (X_i)_{i=0}^t$. To answer $\mathcal{A}'s$ queries to $\textsc{Inv}$, it simply forwards them to its own oracle provided by the AOMDL game. When $\mathcal{A}$ outputs $(X_i')_{i=0}^t$ and $\mathbf{a}, \mathbf{b}, \mathbf{C}$, the reduction simply outputs $\mathbf{b}$ as a solution to the AOMDL game.
Second Case: $\mathbf{b} \neq \mathbf{x}$. In this case, we know that there is a position $i^* \in \{0\} \cup \{t\}$ such that $x_{i^*} \neq b_{i^*}$. Rearranging the i^*th equation we get
$$\gamma = \frac{a_{i^*} + \sum_{j=0}^t c_{i^*,j} x_j}{x_{i^*} - b_{i^*}},$$

[6]In the game as defined in [NRS21], the adversary gets its challenges X_i via an oracle, in which case our reduction also works.

where we used that $x_{i^*} \neq b_{i^*}$. With this, we can construct a reduction solving the plain DLOG problem: it gets as input $(\mathbb{G}, g, p)$ and $h = g^\gamma$, samples the x_i and simulates the AOMCDH game for $\mathcal{A}$, simulating $\textsc{Inv}$ honestly using the x_i. When $\mathcal{A}$ outputs $(X_i')_{i=0}^t$ and $\mathbf{a}, \mathbf{b}, \mathbf{C}$, the reduction outputs γ computed as above. $\square$

Next, we present the tagged linear function family $\mathsf{TLF}_{\mathsf{AOMCDH}} = (\mathsf{Gen}_{\mathsf{AOMCDH}}, \mathsf{T}_{\mathsf{AOMCDH}})$ based on the AOMCDH assumption. 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:

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

Clearly, $\mathcal{D}$ and $\mathcal{R}$ are vector spaces over the field $\mathcal{S}$. Given a tag[7] $u \in \mathbb{G}$ and an input $x \in \mathbb{Z}_p$, the tagged linear function $\mathsf{T}_{\mathsf{AOMCDH}}$ is defined as follows

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

Clearly, $\mathsf{T}_{\mathsf{AOMCDH}}$ can be computed efficiently and is a homomorphism. It remains to argue regularity, translatability and algebraic translation resistance.

Lemma 3. *The tagged linear function family* $\mathsf{TLF}_{\mathsf{AOMCDH}}$ *is* ε_{r}*-regular, where* $\varepsilon_{\mathsf{r}} \leq 1/p$.

Proof. We define the set Reg from the regularity definition to be the set of group parameters and tags $u \in \mathbb{G}$ such that u is a generator of $\mathbb{G}$. Clearly, for parameters $\mathsf{par} \leftarrow \mathsf{Gen}_{\mathsf{AOMCDH}}(1^\lambda)$ and a random tag $u \xleftarrow{\$} \mathbb{G}$, the probability that $(\mathsf{par}, u) \notin \mathsf{Reg}$, i.e., that u is not a generator, is $1/p$. Further, it is clear that if u is a generator, then $\mathsf{T}_{\mathsf{AOMCDH}}(u, \cdot)$ is a bijection from $\mathbb{Z}_p$ to $\mathbb{G}$, and therefore images of uniformly random elements are uniformly random. $\square$

Lemma 4. *The tagged linear function family* $\mathsf{TLF}_{\mathsf{AOMCDH}}$ *is* ε_{t}*-translatable, where* $\varepsilon_{\mathsf{t}} \leq 2/p$.

Proof. Algorithm Shift takes as input parameters par and a tag $u \in \mathbb{G}$. It samples $r \xleftarrow{\$} \mathbb{Z}_p^*$ and outputs a tag $h := u^r$ and a trapdoor $\mathsf{td} := r$. Algorithm $\mathsf{Translate}$ takes as input the trapdoor $\mathsf{td} = r$ and an element $X \in \mathbb{G}$. It outputs X^r. Algorithm $\mathsf{InvTranslate}$ get the same input and outputs $X^{1/r}$. With that, it is clear that the distributions $\mathcal{X}_0$ and $\mathcal{X}_1$ from the definition of translatability are the same conditioning on u and h being generators. Thus, the statistical distance between $\mathcal{X}_0$ and $\mathcal{X}_1$ is at most $2/p$. Further, we have

$$\mathsf{Translate}(\mathsf{td}, \mathsf{T}_{\mathsf{AOMCDH}}(u, x)) = (u^x)^r = (u^r)^x = \mathsf{T}_{\mathsf{AOMCDH}}(h, x)$$

for $(h, \mathsf{td} = r) \in \mathsf{Shift}(\mathsf{par}, u)$. The inverse direction follows similarly. $\square$

Lemma 5. *Let $t \in \mathbb{N}$ be a number polynomial in λ. If the* 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*

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

Proof. The proof is trivial, as the game for the AOMCDH assumption is almost exactly the game for t-algebraic translation resistance of $\mathsf{TLF}_{\mathsf{AOMCDH}}$. Note that the only difference is that in the t-algebraic translation resistance game for $\mathsf{TLF}_{\mathsf{AOMCDH}}$, the inputs have the form $X_i = u^{x_i}$ for a random $u \xleftarrow{\$} \mathbb{G}$, whereas in the AOMCDH game, the inputs have the form g^{x_i}, where g is the fixed generator of $\mathbb{G}$. We can easily build a reduction that resolves this discrepancy: Namely, the reduction gets as input the values g^{x_i} from the AOMCDH assumption. It samples $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and sets $u := g^\alpha$ and defines the inputs X_i as $X_i = (g^{x_i})^\alpha = u^{x_i}$. Oracle queries to $\textsc{Inv}$ and the final output of $\mathcal{A}$ are simply forwarded by the reduction. Clearly, the reduction perfectly simulates the t-algebraic translation resistance game. Also, assuming $\alpha \neq 0$ and that $\mathcal{A}$ breaks t-algebraic translation resistance, we see that the reduction breaks AOMCDH. $\square$

[7]In this section, we do not use the symbol g for arbitrary tags, as it is reserved for the fixed group generator.

4.2 Instantiation from DDH

Here, we present our construction $\mathsf{TLF}_{\mathsf{DDH}} = (\mathsf{Gen}_{\mathsf{DDH}}, \mathsf{T}_{\mathsf{DDH}})$ of a tagged linear function family based on the DDH assumption. We first recall the DDH assumption.

Definition 8 (DDH Assumption)**.** Let GGen be a group generation algorithm. That is, on input 1^λ, $\mathsf{GGen}(1^\lambda)$ outputs the description of a prime order group $\mathbb{G}$. The description contains the prime order p and a generator g if $\mathbb{G}$, 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:

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

From now on, let GGen be an algorithm that takes as input 1^λ and outputs the description of a group $\mathbb{G}$ of prime order p, along with some generator $g \in \mathbb{G}$. Algorithm $\mathsf{Gen}_{\mathsf{DDH}}$ simply runs GGen and outputs the description of $\mathbb{G}$, p, and g as parameters par. We make use of the implicit notation for group elements from [EHK+13]. That is, we write $[\mathbf{A}] \in \mathbb{G}^{r\times l}$ for the matrix of group elements with exponents given by the matrix $\mathbf{A} \in \mathbb{Z}_p^{r\times l}$. Precisely, if $\mathbf{A} = (A_{i,j})_{i\in[r],j\in[l]}$, then $[\mathbf{A}] := (g^{A_{i,j}})_{i\in[r],j\in[l]}$. With this notation, observe that one can efficiently compute $[\mathbf{AB}]$ for any matrices $\mathbf{A} \in \mathbb{Z}_p^{r\times l}$, $\mathbf{B} \in \mathbb{Z}_p^{l\times s}$ with matching dimensions from either $[\mathbf{A}]$ and $\mathbf{B}$ or from $\mathbf{A}$ and $[\mathbf{B}]$. For our tagged linear function family, we define the following sets of scalars, tags, and the domain and range, respectively:

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

Clearly, $\mathcal{D}$ and $\mathcal{R}$ are vector spaces over $\mathcal{S}$. For a tag $[\mathbf{G}] \in \mathbb{G}^{2\times 2}$ and an input $\mathbf{x} \in \mathbb{Z}_p^2$, the tagged linear function $\mathsf{T}_{\mathsf{DDH}}$ is defined as

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

We emphasize that the tag $[\mathbf{G}]$ is given in the group, and the domain element $\mathbf{x}$ is given over the field. It is clear that $\mathsf{T}_{\mathsf{DDH}}$ can be computed efficiently and that it is a homomorphism. What remains is to show regularity, translatability and algebraic translation resistance.

Lemma 6. *The tagged linear function family* $\mathsf{TLF}_{\mathsf{DDH}}$ *is* ε_{r}*-regular, where* $\varepsilon_{\mathsf{r}} \leq (p+1)/p^2$*.*

Proof. We define the set Reg from the regularity definition as the set of group parameters and matrices $[\mathbf{T}] \in \mathbb{G}^{2\times 2}$ such that $\mathbf{T} \in \mathbb{Z}_p^{2\times 2}$ is invertible. Then, the probability that a random tag is not in the set is at most $1/p + 1/p^2 = (p+1)/p^2$. This is because for a uniform 2×2 matrix over $\mathbb{Z}_p$, the probability of not being invertible can easily be upper bounded by $1/p^2$, accounting for the chance that the first row is zero, plus $1/p$, accounting for the chance that the second row is a multiple of the first row. Given that the tag is invertible, the distribution of the image of a uniform inputs $\mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^2$ is clearly uniform. □

Lemma 7. *The tagged linear function family* $\mathsf{TLF}_{\mathsf{DDH}}$ *is* ε_{t}*-translatable, where* $\varepsilon_{\mathsf{t}} \leq (3+3p)/p^2$*.*

Proof. To prove the claim, we first have to describe a PPT algorithm Shift and a deterministic polynomial time algorithm $\mathsf{Translate}$. Algorithm Shift takes as input parameters par specifying $\mathbb{G}, p, g$ and a tag $[\mathbf{G}] \in \mathbb{G}^{2\times 2}$. With this input, it is defined as follows:

1. Let $\mathsf{GL}_2(\mathbb{Z}_p)$ be the set of invertible 2×2 matrices over $\mathbb{Z}_p$. Sample a matrix $\mathbf{R} \xleftarrow{\$} \mathsf{GL}_2(\mathbb{Z}_p)$ uniformly at random from that set.
2. Return the tag $[\mathbf{H}] := [\mathbf{RG}] \in \mathbb{G}^{2\times 2}$ and the trapdoor $\mathsf{td} := \mathbf{R}$.

Algorithm $\mathsf{Translate}$ gets as input the trapdoor $\mathsf{td} = \mathbf{R}$ and an element $[\mathbf{y}] \in \mathbb{G}^2$ in the range. It simply outputs $[\mathbf{Ry}] \in \mathbb{G}^2$. Similarly, algorithm $\mathsf{InvTranslate}$ gets as input the trapdoor $\mathsf{td} = \mathbf{R}$ and an element $[\mathbf{y}] \in \mathbb{G}^2$ in the range and outputs $[\mathbf{R}^{-1}\mathbf{y}] \in \mathbb{G}^2$. With this, translation completeness follow easily. For example, we have

$$\begin{aligned}\mathsf{Translate}(\mathbf{R}, \mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{x})) &= \mathsf{Translate}(\mathbf{R}, [\mathbf{Gx}]) \\ &= [\mathbf{RGx}] = [\mathbf{Hx}] = \mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{x}).\end{aligned}$$

It remains to show the well distributed tags property. For that, it is sufficient to bound the statistical distance between

$$\mathcal{T}_0 := \left\{(\mathsf{par}, \mathbf{G}, \mathbf{H}) \;\middle|\; \mathsf{par} \leftarrow \mathsf{GGen}(1^\lambda),\; \mathbf{G}, \mathbf{H} \overset{\$}{\leftarrow} \mathbb{Z}_p^{2\times 2}\right\}$$

and

$$\mathcal{T}_1 := \left\{(\mathsf{par}, \mathbf{G}, \mathbf{H}) \;\middle|\; \mathsf{par} \leftarrow \mathsf{GGen}(1^\lambda),\; \mathbf{G} \overset{\$}{\leftarrow} \mathbb{Z}_p^{2\times 2},\; \mathbf{R} \overset{\$}{\leftarrow} \mathsf{GL}_2(\mathbb{Z}_p),\; \mathbf{H} := \mathbf{R}\mathbf{G}\right\}.$$

For $b \in \{0,1\}$, let $\mathcal{T}_b^*$ denote the distribution $\mathcal{T}_b$ except that all matrices ($\mathbf{G}, \mathbf{H}$ in $\mathcal{T}_0$ and $\mathbf{G}, \mathbf{R}$ in $\mathcal{T}_1$) are sampled uniformly at random from the set $\mathsf{GL}_2(\mathbb{Z}_p)$. We make three observations, finishing the proof:

1. The statistical distance between $\mathcal{T}_0$ and $\mathcal{T}_0^*$ is at most $2\left(1/p + 1/p^2\right)$, because the distributions only differ if at least one of the two random matrices $\mathbf{G}, \mathbf{H}$ is not invertible.

2. The statistical distance between $\mathcal{T}_1$ and $\mathcal{T}_1^*$ is at most $1/p + 1/p^2$, because the distributions only differ if $\mathbf{G}$ is not invertible.

3. The distributions $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$ are the same. This is because the set of invertible 2×2 matrices over $\mathbb{Z}_p$ forms a group with respect to multiplication.

□

Lemma 8. *Let $t \in \mathbb{N}$ be a number polynomial in λ. If 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*

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

Proof. Assume that there is an efficient algorithm $\mathcal{A}$ breaking algebraic translation resistance. That is, $\mathcal{A}$ gets as input random tags $[\mathbf{G}], [\mathbf{H}] \overset{\$}{\leftarrow} \mathbb{G}^{2\times 2}$ and images $[\mathbf{y}_i] := \mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{x}_i)$ with $\mathbf{x}_i \overset{\$}{\leftarrow} \mathbb{Z}_p^2$ for $i \in \{0, \dots, t\}$. It gets access to the oracle $\textsc{Inv}$ and finally outputs $[\mathbf{y}_i'] \in \mathbb{G}^2$ for all $i \in \{0, \dots, t\}$. We assume without loss of generality, that $\mathcal{A}$ makes exactly t queries to $\textsc{Inv}$. It wins the game if for all $i \in \{0, \dots, t\}$ there is a $\mathbf{z}_i \in \mathbb{Z}_p^2$ such that $\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{z}_i) = [\mathbf{y}_i]$ and $\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{z}_i) = [\mathbf{y}_i']$. The intuition for our proof is that the function $\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \cdot)$ for a random tag $[\mathbf{G}]$ is bijective, so $\mathbf{z}_i = \mathbf{x}_i$ and the adversary will output $\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{x}_i)$. At the same time, based on DDH, the function is indistinguishable from being a compressing function. With an argument similar to what is done in [TZ23], we can then show that $\mathbf{z}_i \neq \mathbf{x}_i$, a contradiction. We now make this intuition formal by providing a sequence of games.

<u>**Game $\mathbf{G}_0$:**</u> Game $\mathbf{G}_0$ is the algebraic translation resistance game for $\mathsf{TLF}_{\mathsf{DDH}}$ as explained above. By definition, we have

$$\mathsf{Adv}^{\mathsf{A\text{-}TRAN\text{-}RES}}_{\mathcal{A},\mathsf{TLF}_{\mathsf{DDH}}}(\lambda) = \Pr\left[\mathbf{G}_0 \Rightarrow 1\right].$$

<u>**Game $\mathbf{G}_1$:**</u> This game is exactly as $\mathbf{G}_0$, but we let the game output 0 if the function defined by the tag $[\mathbf{H}] \in \mathbb{G}^{2\times 2}$ is not bijective. Otherwise, the game behaves as the previous game. More precisely, the game no longer samples $[\mathbf{H}] \overset{\$}{\leftarrow} \mathbb{G}^{2\times 2}$ but instead samples $\mathbf{H} \overset{\$}{\leftarrow} \mathbb{Z}_p^{2\times 2}$ and checks if $\mathbf{H}$ is invertible. If it is not, the game outputs 0. Otherwise, the game continues as in $\mathbf{G}_0$ using tag $[\mathbf{H}]$. Clearly, the distribution of $[\mathbf{H}]$ did not change. The probability of the matrix not being invertible can easily be upper bounded by $1/p^2$, accounting for the chance that the first row is zero, plus $1/p$, accounting for the chance that the second row is a multiple of the first row. We get

$$\left|\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_1 \Rightarrow 1\right]\right| \leq \frac{1}{p^2} + \frac{1}{p}.$$

<u>**Game $\mathbf{G}_2$:**</u> In this game, we change the winning condition. Recall that until now, the game outputs 1 if for all $i \in \{0, \dots, t\}$ there is a $\mathbf{z}_i \in \mathbb{Z}_p^2$ such that $\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{z}_i) = [\mathbf{y}_i]$ and $\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{z}_i) = [\mathbf{y}_i']$. Now, we replace this condition as follows: For each $i \in \{0, \dots, t\}$, the game first computes

$$[\mathbf{w}_i] := [\mathbf{H}^{-1}\mathbf{y}_i'] \in \mathbb{G}^2.$$

Observe that $\mathbf{H}^{-1}$ exists and the game holds it due to the previous change, and thus the game can efficiently compute $[\mathbf{w}_i]$ over the group. Further, observe that $\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{w}_i) = [\mathbf{y}_i']$. Given these $[\mathbf{w}_i]$, the game then accepts if and only if for all $i \in \{0, \dots, t\}$, we have $[\mathbf{w}_i] = [\mathbf{x}_i]$. In other words, the game checks that $\mathbf{w}_i = \mathbf{x}_i$, which it can do efficiently in the exponent. We argue that, except with negligible probability, the winning condition is equivalent to the winning condition in the previous game. Namely, except with probability $1/p^2 + 1/p$, the function $\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \cdot)$ is a bijection. This can be seen with arguments similar to the previous game. Assuming that it is a bijection, we now argue that the two winning conditions are equivalent:

- If the new winning condition in $\mathbf{G}_2$ holds, we can set $\mathbf{z}_i = \mathbf{w}_i = \mathbf{x}_i$ for all $i \in \{0, \dots, t\}$, which shows that the old winning condition in $\mathbf{G}_1$ holds.
- If the old winning condition in $\mathbf{G}_1$ holds, then we know that for all $i \in \{0, \dots, t\}$, we have
$$\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{z}_i) = [\mathbf{y}'_i] = \mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \mathbf{w}_i).$$
As $\mathsf{T}_{\mathsf{DDH}}([\mathbf{H}], \cdot)$ is a bijection, this means that $\mathbf{z}_i = \mathbf{w}_i$. Thus, we have
$$\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{w}_i) = \mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{z}_i) = [\mathbf{y}_i] = \mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{x}_i),$$
where the last equality follows from the old winning condition. As we assume that $\mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \cdot)$ is a bijection, this implies $\mathbf{w}_i = \mathbf{x}_i$, showing that the new winning condition holds.

With that, we obtain
$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \le \frac{1}{p^2} + \frac{1}{p}.$$

Game $\mathbf{G}_3$: This game is exactly as $\mathbf{G}_2$, but we change the way the tag $\mathbf{G} \in \mathbb{G}^{2\times 2}$ is generated. Intuitively, while $\mathbf{G}$ induced a bijection (except with negligible probability) before, we now make sure that it induces a compressing function. Namely, instead of sampling $[\mathbf{G}]$ uniformly at random, we generate it as
$$[\mathbf{G}] := \begin{pmatrix} g^{\beta} & h \\ g^{\alpha\beta} & h^{\alpha} \end{pmatrix} \text{ for } h \overset{\$}{\leftarrow} \mathbb{G},\ \alpha, \beta \overset{\$}{\leftarrow} \mathbb{Z}_p.$$

We can bound the distinguishing advantage between games $\mathbf{G}_2$ and $\mathbf{G}_3$ using a straight-forward reduction $\mathcal{B}$ against the DDH assumption.

1. The reduction $\mathcal{B}$ gets as input parameters defining the group $\mathbb{G}$ with a generator g, a random element $h \in \mathbb{G}$ and group elements $u, v \in \mathbb{G}$, which are either both random or of the form $u = g^{\alpha}, v = h^{\alpha}$.
2. The reduction $\mathcal{B}$ sets up the tag $[\mathbf{H}]$ and the vectors $\mathbf{x}_i$ as in $\mathbf{G}_2$. Then, it samples $\beta \overset{\$}{\leftarrow} \mathbb{Z}_p$ and defines
$$[\mathbf{G}] := \begin{pmatrix} g^{\beta} & h \\ u^{\beta} & v \end{pmatrix}.$$
With this, it computes $[\mathbf{y}_i] := \mathsf{T}_{\mathsf{DDH}}([\mathbf{G}], \mathbf{x}_i)$ as in $\mathbf{G}_2$ and runs $\mathcal{A}$, simulating the oracle $\textsc{Inv}$ for $\mathcal{A}$ as in $\mathbf{G}_2$.
3. When $\mathcal{A}$ terminates and outputs $[\mathbf{y}'_i] \in \mathbb{G}^2$ for all $i \in \{0, \dots, t\}$, the reduction checks the winning condition as in $\mathbf{G}_2$. Recall that this can be done efficiently, due to the change in $\mathbf{G}_2$. It outputs whatever $\mathbf{G}_2$ would output.

Clearly, the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$. Further, if $\mathcal{B}$'s input is random, $\mathcal{B}$ perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$ unless $\beta = 0$, which happens with probability $1/p$. On the other hand, if the input is of the form $u = g^{\alpha}, v = h^{\alpha}$, then $\mathcal{B}$ perfectly simulates $\mathbf{G}_3$ for $\mathcal{A}$. We get
$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \le \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathsf{GGen}}(\lambda) + \frac{1}{p}.$$

What remains is to show that the probability that $\mathbf{G}_3$ outputs 1 is negligible. The intuition is that now $\mathbf{G}$ has low rank, and $\mathcal{A}$ does not obtain enough information about $\mathbf{x}_0, \dots, \mathbf{x}_t$. To turn this intuition into a formal argument, we first introduce more notation.

Scheme	Public Key	Communication	Signature
Frost	33	98	64
TZ	33	130	97
Sparkle	33	97	64
Twinkle (AOMCDH)	33	163	97
Twinkle (DDH)	66	294	162

Table 2: Concrete efficiency of threshold signature schemes in the discrete logarithm setting without pairings. We compare the size of public keys, the communication complexity per signer, and the signature size. Sizes are given in bytes.

Notation. We set $\mathbf{X} \in \mathbb{Z}_p^{2\times(t+1)}$ to be the matrix with columns $\mathbf{x}_i$, $\mathbf{Y} := \mathbf{G}\mathbf{X}$ to be the matrix with columns $\mathbf{y}_i$, and $\mathbf{Y}' \in \mathbb{Z}_p^{2\times(t+1)}$ be the matrix with columns $\mathbf{y}'_i$. The winning condition introduced in $\mathbf{G}_2$ can now be written as $\mathbf{H}^{-1}\mathbf{Y}' = \mathbf{X}$.

Claim. We claim that for all fixed parameters par, each fixed $\mathbf{X}$, and for fixed random coins ρ for $\mathcal{A}$, there is a matrix $\mathbf{K} \in \mathbb{Z}_p^{2\times(t+1)} \setminus \{\mathbf{0}\}$ such that the view of $\mathcal{A}$ in $\mathbf{G}_3$ is independent of $\vartheta \in \mathbb{Z}_p$ if we replace $\mathbf{X}$ with $\mathbf{X} + \vartheta\mathbf{K}$. Assuming this claim holds, it is clear that

$$\Pr[\mathbf{G}_3 \Rightarrow 1] \leq \frac{1}{p}.$$

Proof of Claim. It remains to show the claim. Fix parameters par, a matrix $\mathbf{X}$, and random coins ρ. Observe that with that, all t queries of $\mathcal{A}$ to $\textsc{Inv}$ are defined. Let the jth query to this oracle be $\textsc{Inv}(\alpha_0^{(j)}, \ldots, \alpha_t^{(j)})$. The queries define a matrix $\mathbf{A} \in \mathbb{Z}_p^{(t+1)\times t}$ where the jth column of $\mathbf{A}$ is $(\alpha_0^{(j)}, \ldots, \alpha_t^{(j)})$. With this in mind, it is clear that the view of $\mathcal{A}$ in game $\mathbf{G}_3$ is given by

$$\mathbf{H},\ \mathbf{G},\ \mathbf{G}\mathbf{X},\ \mathbf{X}\mathbf{A}.$$

Therefore, it is sufficient to show that there is a matrix $\mathbf{K} \in \mathbb{Z}_p^{2\times(t+1)} \setminus \{\mathbf{0}\}$ such that

$$\mathbf{G}\mathbf{K} = \mathbf{0} \text{ and } \mathbf{K}\mathbf{A} = \mathbf{0}.$$

By the way we sample $\mathbf{G}$ in game $\mathbf{G}_3$, there exists a vector $\mathbf{g}_\perp \in \mathbb{Z}_p^2 \setminus \{\mathbf{0}\}$ such that $\mathbf{G}\mathbf{g}_\perp = \mathbf{0}$. Also, there is a vector $\mathbf{a}_\perp \in \mathbb{Z}_p^{t+1} \setminus \{\mathbf{0}\}$ such that $\mathbf{a}_\perp^t \mathbf{A} = \mathbf{0}$, which follows from the dimensions of $\mathbf{A}$. Now, we can set

$$\mathbf{K} := \mathbf{g}_\perp \mathbf{a}_\perp^t \in \mathbb{Z}_p^{2\times(t+1)} \setminus \{\mathbf{0}\}.$$

□

5 Concrete Parameters and Efficiency

In the final section of this paper, we briefly discuss the concrete efficiency of our threshold signature schemes. We compare our schemes to Frost [KG20, BTZ22, BCK+22], TZ [TZ23], and Sparkle [CKM23a]. For our comparison, we assume that all constructions are instantiated with the secp256k1 curve and SHA-256 as a hash function. We calculate key sizes, communication complexity per signer, and signature sizes. Also, for all schemes, we assume challenges c_i are sampled uniformly from $\mathbb{Z}_p$, i.e., have size 256 bit, whereas some implementations may use challenges of size 128 bit. Our results are summarized in Table 2 and the Python script used to compute these numbers is given in Supplementary Material B. We see that the sizes of our constructions are practical, but slightly less efficient compared to previous schemes. In terms of computation, consider a run of the signing protocol in which K signers participate. Here, a signer in both of our schemes has to evaluate $K + 2$ hashes, whereas it would have to evaluate $K + 1$ hashes in Sparkle, which is a minimal difference as K grows. A signer in Sparkle would have to compute 2 group operations (counting multi-scalar multiplications as one operation), whereas a signer in our AOMCDH-based (resp. DDH-based) scheme would have to do 6 (resp. 12) such operations. For verification, the number of operations compared to Sparkle increases by a factor of 2 for the hashes, and a factor of 2 (resp. 4) for the multi-scalar multiplications for our AOMCDH-based (resp. DDH-based) scheme.

To sum up, our schemes are slightly less efficient than previous schemes, but they are still in a highly practical regime. Given the strong properties that our schemes achieve from conservative assumptions without the algebraic group model, it is natural to pay such a small price in terms of efficiency.

Acknowledgments. CISPA authors are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585, and by the European Union, ERC-2023-STG, Project ID: 101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Tessaro and Zhu are supported in part by NSF grants CNS- 2026774, CNS-2154174, a JP Morgan Faculty Award, a CISCO Faculty Award, and a gift from Microsoft.

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, Heidelberg. (Cited on page 7.)

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Heidelberg, May / June 2006. (Cited on page 7.)

[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, Heidelberg, April 2012. (Cited on page 5, 10.)

[AHS20] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. A survey of ECDSA threshold signing. Cryptology ePrint Archive, Report 2020/1390, 2020. `https://eprint.iacr.org/2020/1390`. (Cited on page 7.)

[ASY22] Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl, July 2022. (Cited on page 7.)

[BCK$^+$22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Heidelberg, August 2022. (Cited on page 4, 9, 27.)

[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, Heidelberg, December 2021. (Cited on page 7.)

[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, Heidelberg, December 2018. (Cited on page 7.)

[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, Heidelberg, December 2021. (Cited on page 22.)

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018. (Cited on page 7.)

[BHK$^+$23] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. Sprint: High-throughput robust distributed schnorr signatures. Cryptology ePrint Archive, Paper 2023/427, 2023. `https://eprint.iacr.org/2023/427`. (Cited on page 7.)

[BKP13] Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 218–236. Springer, Heidelberg, June 2013. (Cited on page 7.)

[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 7.)

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001. (Cited on page 7.)

[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 6, 7.)

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003. (Cited on page 7.)

[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: 2023-09-12. (Cited on page 3.)

[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, Heidelberg, August 2022. (Cited on page 7.)

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. `https://eprint.iacr.org/2022/833`. (Cited on page 4, 9, 27.)

[CAHL$^+$22] Rutchathon Chairattana-Apirom, Lucjan Hanzlik, Julian Loss, Anna Lysyanskaya, and Benedikt Wagner. PI-cut-choo and friends: Compact blind signatures via parallel instance cut-and-choose and more. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 3–31. Springer, Heidelberg, August 2022. (Cited on page 4, 9.)

[CCL$^+$20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Heidelberg, May 2020. (Cited on page 7.)

[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 7.)

[CGJ$^+$99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Heidelberg, August 1999. (Cited on page 7.)

[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, Heidelberg, August 2023. (Cited on page 4, 9.)

[Che05] Benoît Chevallier-Mames. An efficient CDH-based signature scheme with a tight security reduction. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 511–526. Springer, Heidelberg, August 2005. (Cited on page 4, 5.)

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

[CKM23a] 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, Heidelberg, August 2023. (Cited on page 3, 4, 5, 9, 27.)

[CKM$^+$23b] 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, Heidelberg, August 2023. (Cited on page 3.)

[CKP$^+$23] Elizabeth Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. In *Asiacrypt 2023*. Springer-Verlag, 2023. (Cited on page 7.)

[DCX$^+$23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bunz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. Cryptology ePrint Archive, Paper 2023/598, 2023. https://eprint.iacr.org/2023/598. (Cited on page 7.)

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994. (Cited on page 7.)

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

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

[DJN$^+$20] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 382–400. Springer, Heidelberg, September 2020. (Cited on page 7.)

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019. (Cited on page 7.)

[DOK$^+$20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 654–673. Springer, Heidelberg, September 2020. (Cited on page 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, Heidelberg, May 2021. (Cited on page 7.)

[DYX$^+$22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE Computer Society Press, May 2022. (Cited on page 7.)

[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, Heidelberg, August 2013. (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, Heidelberg, August 2018. (Cited on page 3, 22.)

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, page 320. ACM, June / July 1998. (Cited on page 7.)

[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 7.)

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018. (Cited on page 7.)

[GG20] Rosario Gennaro and Steven Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. `https://eprint.iacr.org/2020/540`. (Cited on page 7.)

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 156–174. Springer, Heidelberg, June 2016. (Cited on page 7.)

[GHKR08] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold RSA for dynamic and ad-hoc groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 88–107. Springer, Heidelberg, April 2008. (Cited on page 7.)

[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, 7.)

[GJKW07] Eu-Jin Goh, Stanislaw Jarecki, Jonathan Katz, and Nan Wang. Efficient signature schemes with tight reductions to the Diffie-Hellman problems. *Journal of Cryptology*, 20(4):493–514, October 2007. (Cited on page 4, 5.)

[GKS23] Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice signatures from threshold homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1318, 2023. `https://eprint.iacr.org/2023/1318`. (Cited on page 7.)

[GKSŚ20] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020. `https://eprint.iacr.org/2020/498`. (Cited on page 7.)

[GS23] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2023/1175, 2023. `https://eprint.iacr.org/2023/1175`. (Cited on page 7.)

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375. Springer, Heidelberg, May 2019. (Cited on page 4, 9.)

[IN83] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983. (Cited on page 7.)

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

[KG20] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020. (Cited on page 4, 27.)

[KGS23] Chelsea Komlo, Ian Goldberg, and Douglas Stebila. A formal treatment of distributed key generation, and new constructions. Cryptology ePrint Archive, Report 2023/292, 2023. https://eprint.iacr.org/2023/292. (Cited on page 7.)

[KLP17] Eike Kiltz, Julian Loss, and Jiaxin Pan. Tightly-secure signatures from five-move identification protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 68–94. Springer, Heidelberg, December 2017. (Cited on page 4, 5, 10.)

[KLR21] Jonathan Katz, Julian Loss, and Michael Rosenberg. Boosting the security of blind signature schemes. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 468–492. Springer, Heidelberg, December 2021. (Cited on page 4, 9.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Heidelberg, August 2016. (Cited on page 4, 5.)

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1751–1767. ACM Press, November 2020. (Cited on page 7.)

[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, 10.)

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374. (Cited on page 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 7.)

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018. (Cited on page 3, 7.)

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Heidelberg, December 2001. (Cited on page 7.)

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptogr.*, 87(9):2139–2164, 2019. (Cited on page 7.)

[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, Heidelberg. (Cited on page 7, 22.)

[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 7.)

[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, Heidelberg, April 1991. (Cited on page 3.)

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. (Cited on page 7.)

[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, Heidelberg, April 2023. (Cited on page 4, 5, 7, 9, 10.)

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 89–104. Springer, Heidelberg, August 1998. (Cited on page 7.)

[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, 7.)

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

[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, Heidelberg, May 1997. (Cited on page 22.)

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000. (Cited on page 7.)

[Sho23] Victor Shoup. The many faces of schnorr. Cryptology ePrint Archive, Paper 2023/1019, 2023. `https://eprint.iacr.org/2023/1019`. (Cited on page 7.)

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001. (Cited on page 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, Heidelberg, 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, Heidelberg, April 2023. (Cited on page 3, 4, 6, 7, 9, 25, 27.)

A Pseudocode

```
**Alg** $\mathsf{Setup}(1^\lambda)$
$\mathsf{par}' \leftarrow \mathsf{TLF.Gen}(1^\lambda), \quad g \xleftarrow{\$} \mathcal{T}$
**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** $\mathsf{Sig}_0(S, i, \mathsf{sk}_i, \mathsf{m})$
$h := \mathsf{H}(\mathsf{m})$
$r_i \xleftarrow{\$} \mathcal{D}$
$R_i^{(1)} := \mathsf{T}(g, r_i), \; R_i^{(2)} := \mathsf{T}(h, r_i)$
$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$
$\mathsf{com}_i := \hat{\mathsf{H}}(S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$
$\mathsf{pm}_1 := \mathsf{com}_i$
$St_1 := (i, S, \mathsf{sk}_i, h, \mathsf{m}, r_i)$
**return** $(\mathsf{pm}_1, St_1)$

**Alg** $\mathsf{Sig}_1(St_1, \mathcal{M}_1)$
**parse** $(i, S, \mathsf{sk}_i, h, \mathsf{m}, r_i) := St_1$
$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$
$R_i^{(1)} := \mathsf{T}(g, r_i), \; R_i^{(2)} := \mathsf{T}(h, r_i)$
$\mathsf{pm}_2 := (R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$
**return** $(\mathsf{pm}_2, St_2 := (\mathcal{M}_1, St_1))$

**Alg** $\mathsf{Sig}_2(St_2, \mathcal{M}_2)$
**parse** $(\mathcal{M}_1, (i, S, \mathsf{sk}_i, h, \mathsf{m}, r_i)) := St_2$
**parse** $(\mathsf{com}_j)_{j \in S} := \mathcal{M}_1$
**parse** $((R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}))_{j \in S} := \mathcal{M}_2$
**for** $j \in S$ :
    **if** $\hat{\mathsf{H}}(S, j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) \neq \mathsf{com}_j$ :
        **return** $\perp$
$R^{(1)} := \sum_{j \in S} R_j^{(1)}$
$R^{(2)} := \sum_{j \in S} R_j^{(2)}$
$\mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \mathsf{pk}_j^{(2)}$
$c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$
**return** $\mathsf{pm}_3 := s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$

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

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

Figure 5: The (t, n)-threshold signature scheme $\mathsf{Twinkle}[\mathsf{TLF}] = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ for a tagged linear function family TLF.

```
Game $\mathbf{G}_0$-$\mathbf{G}_{11}$
Pending := $\emptyset$                                                        // $\mathbf{G}_7$-$\mathbf{G}_{11}$
Reps := $\emptyset$                                                           // $\mathbf{G}_9$-$\mathbf{G}_{11}$
$\mathsf{par}' \leftarrow \mathsf{TLF.Gen}(1^\lambda)$,  $g \xleftarrow{\$} \mathcal{T}$,  $\mathsf{par} := (\mathsf{par}', g)$
if $(\mathsf{par}', g) \notin \mathsf{Reg}$ : abort                               // $\mathbf{G}_4$-$\mathbf{G}_{10}$
$a_0, \ldots, a_t \xleftarrow{\$} \mathcal{D}$
for $i \in [n]$ : $\mathsf{sk}_i := \sum_{j=0}^{t} a_j i^j$
for $i \in [n]$ : $\mathsf{pk}_i := \mathsf{T}(g, \mathsf{sk}_i)$
$\mathsf{pk} := \mathsf{pk}_0 := \mathsf{T}(g, a_0)$
$\textsc{Sig} := (\textsc{Next}, \textsc{Sig}_0, \textsc{Sig}_1, \textsc{Sig}_2)$
$(\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\textsc{Sig},\textsc{Corr},\mathsf{H},\hat{\mathsf{H}},\bar{\mathsf{H}}}(\mathsf{par}, \mathsf{pk})$
if $b[\mathsf{m}^*] = 0$ : return $0$                                               // $\mathbf{G}_1$-$\mathbf{G}_{11}$
if $\exists x \neq x'$ s.t. $\hat{h}[x] = \hat{h}[x'] \neq \bot$ : return $0$             // $\mathbf{G}_6$-$\mathbf{G}_{11}$
if $\mathsf{m}^* \in \mathsf{Queried}$ : return $0$
return $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}^*, \sigma^*)$

Oracle $\textsc{Corr}(i)$
if $|\mathsf{Corrupted}| \geq t$ : return $\bot$
$\mathsf{Corrupted} := \mathsf{Corrupted} \cup \{i\}$
for $sid \in \mathsf{Sessions}$ s.t. $\mathsf{round}[sid, i] = 1$ :                     // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    parse $(i, S, h, \mathsf{m}, \mathsf{com}_i) := \mathsf{state}[sid, i]$                  // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    $\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m}], \mathsf{pk}_i)$                      // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    $r_i \xleftarrow{\$} \mathcal{D}$, $R_i^{(1)} := \mathsf{T}(g, r_i)$, $R_i^{(2)} := \mathsf{T}(h, r_i)$          // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    if $\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] \neq \bot$ : abort              // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    $\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] := \mathsf{com}_i$                   // $\mathbf{G}_5$-$\mathbf{G}_{11}$
    $\mathsf{state}[sid, i] := (i, S, h, \mathsf{m}, r_i)$                              // $\mathbf{G}_5$-$\mathbf{G}_{11}$
for $sid \in \mathsf{Sessions}$ s.t. $\mathsf{round}[sid, i] \geq 2$ :                    // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    parse $(\mathcal{M}_1, (i, S, h, \mathsf{m}, s_i, c)) := \mathsf{state}[sid, i]$           // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $r_i := s_i - c \cdot \ell_{i,S} \cdot \mathsf{sk}_i$                                // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathsf{state}[sid, i] := (\mathcal{M}_1, (i, S, h, \mathsf{m}, r_i))$                    // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
$\mathsf{UpdatePending}()$                                                  // $\mathbf{G}_7$-$\mathbf{G}_{11}$
return $(\mathsf{sk}_i, \mathsf{state}[\cdot, i])$
```

Figure 6: Games $\mathbf{G}_0$-$\mathbf{G}_{11}$ in the proof of Theorem 1. Lines with highlighted are only executed in the respective games. Signing oracles are given in Figures 7 to 9. Random oracles are given in Figure 10. Algorithm UpdatePending is given in Figure 11. Oracle $\textsc{Next}$ is as in Figure 2.

```
Oracle SIG_0$(sid, i)$
if $\mathsf{Allowed}(sid, i, 0, \perp) = 0$ :
    return $\perp$
$h := \mathsf{H}(\mathsf{m}),\quad S := \mathsf{signers}[sid]$
if $b[\mathsf{m}] = 1$ : abort                                                    // $\mathbf{G}_1$-$\mathbf{G}_{11}$
$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$                              // $\mathbf{G}_0$-$\mathbf{G}_2$
$\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m}], \mathsf{pk}_i)$         // $\mathbf{G}_3$-$\mathbf{G}_{11}$
$r_i \xleftarrow{\$} \mathcal{D}$                                                  // $\mathbf{G}_0$-$\mathbf{G}_4$
$R_i^{(1)} := \mathsf{T}(g, r_i),\ R_i^{(2)} := \mathsf{T}(h, r_i)$                // $\mathbf{G}_0$-$\mathbf{G}_4$
$\mathsf{com}_i := \hat{\mathsf{H}}(S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$   // $\mathbf{G}_0$-$\mathbf{G}_4$
$\mathsf{state}[sid, i] := (i, S, h, \mathsf{m}, r_i)$                             // $\mathbf{G}_0$-$\mathbf{G}_4$
$\mathsf{com}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$                                // $\mathbf{G}_5$-$\mathbf{G}_{11}$
if $\exists (S', i') \neq (S, i)$ s.t. $(S', i', \mathsf{com}_i) \in \mathsf{Sim}$ : abort   // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\mathsf{Sim} := \mathsf{Sim} \cup \{(S, i, \mathsf{com}_i)\}$                     // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\mathsf{state}[sid, i] := (i, S, h, \mathsf{m}, \mathsf{com}_i)$                  // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\mathsf{round}[sid, i] := 1$
return $\mathsf{pm}_1[sid, i] := \mathsf{com}_i$
```

Figure 7: Signing Oracle $\textsc{Sig}_0$ in the proof of Theorem 1. Lines with highlighted are only executed in the respective games. Algorithm Allowed is as in Figure 2.

```
Oracle SIG_1$(sid, i, \mathcal{M}_1)$
if $\mathsf{Allowed}(sid, i, 1, \mathcal{M}_1) = 0$ :
    return $\perp$
$\mathsf{m} := \mathsf{message}[sid],\ S := \mathsf{signers}[sid],\ H := S \setminus \mathsf{Corrupted}$
$\mathsf{added} := \mathsf{AddToPending}(sid, i, \mathcal{M}_1)$                  // $\mathbf{G}_7$-$\mathbf{G}_{11}$
parse $(i, S, h, \mathsf{m}, r_i) := \mathsf{state}[sid, i]$                        // $\mathbf{G}_0$-$\mathbf{G}_4$
parse $(i, S, h, \mathsf{m}, \mathsf{com}_i) := \mathsf{state}[sid, i]$             // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\mathsf{pk}_i^{(2)} := \mathsf{T}(h, \mathsf{sk}_i)$                               // $\mathbf{G}_0$-$\mathbf{G}_2$
$\mathsf{pk}_i^{(2)} := \mathsf{Translate}(tr[\mathsf{m}], \mathsf{pk}_i)$          // $\mathbf{G}_3$-$\mathbf{G}_{11}$
$r_i \xleftarrow{\$} \mathcal{D}$                                                   // $\mathbf{G}_5$-$\mathbf{G}_9$
$R_i^{(1)} := \mathsf{T}(g, r_i),\ R_i^{(2)} := \mathsf{T}(h, r_i)$                 // $\mathbf{G}_0$-$\mathbf{G}_9$
if $\mathsf{added} = 1$ : $c := \mathsf{GetChallenge}(sid, i, \mathcal{M}_1)$      // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
if $\mathsf{added} = 0$ : $c := 0$                                                  // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
$s_i \xleftarrow{\$} \mathcal{D}$                                                   // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
$R_i^{(1)} := \mathsf{T}(g, s_i) - c \cdot \ell_{i,S} \cdot \mathsf{pk}_i,\ R_i^{(2)} := \mathsf{T}(h, s_i) - c \cdot \ell_{i,S} \cdot \mathsf{pk}_i^{(2)}$   // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
if $\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] \neq \perp$ : abort    // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\hat{h}[S, i, R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}] := \mathsf{com}_i$        // $\mathbf{G}_5$-$\mathbf{G}_{11}$
$\mathsf{pm}_2[sid, i] := (R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)})$
$\mathsf{state}[sid, i] := (\mathcal{M}_1, (i, S, h, \mathsf{m}, r_i))$             // $\mathbf{G}_0$-$\mathbf{G}_9$
$\mathsf{state}[sid, i] := (\mathcal{M}_1, (i, S, h, \mathsf{m}, s_i, c))$          // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
$\mathsf{round}[sid, i] := 2$
$\mathsf{UpdatePending}()$                                                          // $\mathbf{G}_7$-$\mathbf{G}_{11}$
return $\mathsf{pm}_2[sid, i]$
```

Figure 8: Signing Oracle $\textsc{Sig}_1$ in the proof of Theorem 1. Lines with highlighted are only executed in the respective games. Algorithm Allowed is as in Figure 2.

```
Oracle $\textsc{Sig}_2(sid, i, \mathcal{M}_2)$
if $\mathsf{Allowed}(sid, i, 2, \mathcal{M}_2) = 0$ : return $\perp$
parse $(\mathcal{M}_1, (i, S, h, \mathsf{m}, r_i)) := \mathsf{state}[sid, i]$        // G_0-G_9
parse $(\mathcal{M}_1, (i, S, h, \mathsf{m}, s_i, c)) := \mathsf{state}[sid, i]$        // G_10-G_11
parse $(\mathsf{com}_i)_{i \in S} := \mathcal{M}_1$
parse $((R_i^{(1)}, R_i^{(2)}, \mathsf{pk}_i^{(2)}))_{i \in S} := \mathcal{M}_2$
for $j \in S$ : if $\hat{\mathsf{H}}(S, j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) \neq \mathsf{com}_j$ : return $\perp$
$R^{(1)} := \sum_{j \in S} R_j^{(1)}, \quad R^{(2)} := \sum_{j \in S} R_j^{(2)}$
$\mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \mathsf{pk}_j^{(2)}$
$c := \bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$
$\mathsf{round}[sid, i] := 3$
$s_i := c \cdot \ell_{i,S} \cdot \mathsf{sk}_i + r_i$        // G_0-G_9
return $\mathsf{pm}_3 := s_i$
```

Figure 9: Signing Oracle $\textsc{Sig}_2$ in the proof of Theorem 1. Lines with highlighted are only executed in the respective games. Algorithm Allowed is as in Figure 2.

```
Oracle $\mathsf{H}(\mathsf{m})$
01 if $h[\mathsf{m}] = \perp$ :
02     $h[\mathsf{m}] \xleftarrow{\$} \mathcal{T}$
03     $b[\mathsf{m}] \leftarrow \mathcal{B}_\gamma$        // G_1-G_11
04     if $b[\mathsf{m}] = 0$ :        // G_2-G_11
05         $(h, \mathsf{td}) \leftarrow \mathsf{Shift}(\mathsf{par}', g)$        // G_2-G_11
06         $h[\mathsf{m}] := h, \ tr[\mathsf{m}] := \mathsf{td}$        // G_2-G_11
07 return $h[\mathsf{m}]$

Oracle $\hat{\mathsf{H}}(S, j, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)})$
08 if $\hat{h}[S, j, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}] = \perp$ :
09     $\mathsf{com} \xleftarrow{\$} \{0,1\}^{2\lambda}$
10     $\hat{h}[S, j, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}] := \mathsf{com}$
11     if $\exists (sid, i, \mathcal{M}_1) \in \mathsf{Pending}$
                 s.t. $\mathsf{com} \in \mathcal{M}_1$ :        // G_7-G_11
12             abort        // G_7-G_11
13 $\mathsf{UpdatePending}()$        // G_7-G_11
14 return $\hat{h}[S, j, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}]$

Oracle $\bar{\mathsf{H}}(\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m})$
15 if $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] = \perp$ :
16     $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \xleftarrow{\$} \mathcal{S}$
17 return $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}]$

Alg $\hat{\mathsf{H}}^{-1}(y)$
18 $P := \{x \in \{0,1\}^* \mid \hat{h}[x] = y\}$
19 if $|P| \neq 1$ : return $\perp$
20 parse $\{x\} = P$
21 return $x$
```

Figure 10: Random oracles $\mathsf{H}, \hat{\mathsf{H}}, , \bar{\mathsf{H}}$ and algorithm $\hat{\mathsf{H}}^{-1}$ in the proof of Theorem 1. Lines with highlighted are only executed in the respective games. Here, $\mathcal{B}_\gamma$ denotes a Bernoulli distribution with parameter $\gamma = 1/(Q_S + 1)$.

```
Alg AddToPending($sid, i, \mathcal{M}_1$)
$S :=$ signers$[sid]$,   NotSim $:= \{j \in S \mid (S, j, \mathsf{com}_j) \notin \mathsf{Sim}\}$
parse $(\mathsf{com}_j)_{j \in S} := \mathcal{M}_1$
if $\exists j \in$ NotSim s.t. $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot$ : return 0
for $j \in$ NotSim :  $(S'_j, k_j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) := \hat{\mathsf{H}}^{-1}(\mathsf{com}_j)$
if $\exists j \in$ NotSim s.t. $S'_j \neq S \vee k_j \neq j$ :  return 0
Pending := Pending $\cup \{(sid, i, \mathcal{M}_1)\}$
return 1

Alg UpdatePending()
New $:= \emptyset$
for $(sid, i, \mathcal{M}_1) \in$ Pending :
    $S :=$ signers$[sid]$,  m := message$[sid]$
    parse $(\mathsf{com}_j)_{j \in S} := \mathcal{M}_1$
    if $\forall j \in S$ :  $\hat{\mathsf{H}}^{-1}(\mathsf{com}_j) \neq \bot$ :
        for $j \in S$ :  $(S'_j, k_j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) := \hat{\mathsf{H}}^{-1}(\mathsf{com}_j)$
        if $\exists j \in S$ s.t. $S'_j \neq S \vee k_j \neq j$ :  continue
        Pending := Pending $\setminus \{(sid, i, \mathcal{M}_1)\}$
        $R^{(1)} := \sum_{j \in S} R_j^{(1)}$,   $R^{(2)} := \sum_{j \in S} R_j^{(2)}$,   $\mathsf{pk}^{(2)} := \sum_{j \in S} \ell_{j,S} \mathsf{pk}_j^{(2)}$
        if $(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m}) \notin$ New :                         // $\mathbf{G}_7$
        if $(S, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m}) \notin$ New :                   // $\mathbf{G}_8$-$\mathbf{G}_{11}$
            if $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \neq \bot$ :  abort
        $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] \xleftarrow{\$} \mathcal{S}$
        $\bar{h}[\mathsf{pk}, \mathsf{pk}^{(2)}, R^{(1)}, R^{(2)}, \mathsf{m}] :=$ GetChallenge$(sid, i, \mathcal{M}_1)$   // $\mathbf{G}_9$-$\mathbf{G}_{11}$
        New := New $\cup \{(R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})\}$                 // $\mathbf{G}_7$
        New := New $\cup \{(S, R^{(1)}, R^{(2)}, \mathsf{pk}^{(2)}, \mathsf{m})\}$           // $\mathbf{G}_8$-$\mathbf{G}_{11}$

Alg Equivalent $((sid, i, \mathcal{M}_1), (sid', i', \mathcal{M}'_1))$
m := message$[sid]$,   m$'$ := message$[sid']$
$S :=$ signers$[sid]$,   $S' :=$ signers$[sid']$
if $S \neq S' \vee \mathsf{m} \neq \mathsf{m}'$ :  return 0
parse $(\mathsf{com}_j)_{j \in S} := \mathcal{M}_1$,   $(\mathsf{com}'_j)_{j \in S} := \mathcal{M}'_1$
$F := \{j \in S \mid \hat{\mathsf{H}}^{-1}(\mathsf{com}_j) = \bot\}$,  $\bar{F} := S \setminus F$
$F' := \{j \in S \mid \hat{\mathsf{H}}^{-1}(\mathsf{com}'_j) = \bot\}$,  $\bar{F}' := S \setminus F'$
for $j \in \bar{F}$ :   $(\tilde{S}_j, k_j, R_j^{(1)}, R_j^{(2)}, \mathsf{pk}_j^{(2)}) := \hat{\mathsf{H}}^{-1}(\mathsf{com}_j)$
for $j \in \bar{F}'$ :   $(\tilde{S}'_j, k_j, {R'_j}^{(1)}, {R'_j}^{(2)}, {\mathsf{pk}'_j}^{(2)}) := \hat{\mathsf{H}}^{-1}(\mathsf{com}'_j)$
$\bar{R}^{(1)} := \sum_{j \in \bar{F}} R_j^{(1)}$,    $\bar{R}^{(2)} := \sum_{j \in \bar{F}} R_j^{(2)}$,    $\bar{\mathsf{pk}}^{(2)} := \sum_{j \in \bar{F}} \ell_{j,S} \mathsf{pk}_j^{(2)}$
$\bar{R}'^{(1)} := \sum_{j \in \bar{F}'} {R_j}'^{(1)}$,    $\bar{R}'^{(2)} := \sum_{j \in \bar{F}'} {R_j}'^{(2)}$,    $\bar{\mathsf{pk}}'^{(2)} := \sum_{j \in \bar{F}'} \ell_{j,S} {\mathsf{pk}_j}'^{(2)}$
if $(\bar{R}^{(1)}, \bar{R}^{(2)}, \bar{\mathsf{pk}}^{(2)}) \neq (\bar{R}'^{(1)}, \bar{R}'^{(2)}, \bar{\mathsf{pk}}'^{(2)})$ :  return 0
if $(\mathsf{com}_j)_{j \in F} \neq (\mathsf{com}'_j)_{j \in F'}$ :  return 0
return 1

Alg GetChallenge($sid, i, \mathcal{M}_1$)
for rep $\in$ Reps :
    if Equivalent $((sid, i, \mathcal{M}_1), \mathsf{rep}) = 1$ :  return C[rep]
Reps := Reps $\cup \{(sid, i, \mathcal{M}_1)\}$,   C$[(sid, i, \mathcal{M}_1)] \xleftarrow{\$} \mathcal{S}$
return C$[(sid, i, \mathcal{M}_1)]$
```

Figure 11: Algorithms AddToPending, , UpdatePending managing list Pending, and algorithms Equivalent, GetChallenge to implement a random oracle on equivalence classes. The algorithms are used in the proof of Theorem 1. Lines with highlighted comments are only executed in the respective games.

B Script for Parameter Computation

Listing 1: Python Script to compute concrete efficiency of threshold signature schemes.

```python
#!/usr/bin/env python


################PURPOSE#OF#THIS#SCRIPT##################
# For each scheme, we compute sizes of public keys,    #
# signatures, and communication complexity per signer. #
# we assume that secp256k1 is used                     #
########################################################

import math
from tabulate import tabulate

#sizes of group elements and exponents
#we assume secp256k1; all sizes in bytes
sizec = 32 # size of a challenge
sizege = 33
sizefe = 32
sizehash = 32

sizeschnorrsig = sizec+sizefe

frost = {
        "name": "Frost",
        "pk": sizege,
        "comm": 2*sizege + sizefe,
        "sig": sizeschnorrsig,
}

tz = {
    "name": "TZ",
        "pk": sizege,
        "comm": 2*sizege + 2*sizefe,
        "sig": sizege + 2*sizefe,
}

sparkle = {
    "name": "Sparkle",
        "pk": sizege,
        "comm": sizehash + sizege + sizefe,
        "sig": sizeschnorrsig,
}

sizerange = 2*sizege
sizedomain = 2*sizefe
twinkle = {
    "name": "Twinkle-DDH",
        "pk": sizerange,
        "comm": sizehash + 3*sizerange + sizedomain,
        "sig": sizerange + sizedomain + sizec,
}

sizerange = sizege
sizedomain = sizefe
twinkleOneMore = {
    "name": "Twinkle-OMCDH",
        "pk": sizerange,
        "comm": sizehash + 3*sizerange + sizedomain,
        "sig": sizerange + sizedomain + sizec,
}

schemes = [frost,tz,sparkle,twinkle,twinkleOneMore]

#####################Main Part########################

data = [["Scheme", "PK", "Communication (per Signer)", "Signature Size"]]

for s in schemes:
        data.append([s["name"], s["pk"], s["comm"], s["sig"]])

print(tabulate(data,headers='firstrow',tablefmt='fancy_grid'))
```