Adaptively Secure Threshold Blind BLS Signatures and Threshold Oblivious PRF

Stanislaw Jarecki and Phillip Nazarian

University of California Irvine, USA, [sjarecki, pnazaria]@uci.edu

Abstract. We show the first threshold blind signature scheme and threshold Oblivious PRF (OPRF) scheme which remain secure in the presence of an *adaptive adversary*, who can adaptively decide which parties to corrupt throughout the lifetime of the scheme. Moreover, our adaptively secure schemes preserve the minimal round complexity and add only a small computational overhead over prior solutions that offered security only for a much less realistic *static adversary*, who must choose the subset of corrupted parties before initializing the protocol.
Our threshold blind signature scheme computes standard BLS signatures while our threshold OPRF computes the 2HashDH OPRF [58], and we prove adaptive security of both schemes in the Algebraic Group Model (AGM). Our adaptively secure threshold schemes are as practical as the underlying standard (i.e. single-server) BLS blind signature [14] and 2HashDH OPRF, and they can be used to add cryptographic fault-tolerance and decentralize trust in any system that relies on blind signatures, like anonymous credentials and e-cash, or on OPRF, like the *OPAQUE* password authentication and the *Privacy Pass* anonymous authentication scheme, among many others.

1 Introduction

Threshold and Blind Signatures. In threshold signatures, introduced by Desmedt and Frankel [41, 42], n parties can compute signatures but corruption of any t parties leaks no information on the signature key. Threshold signatures enhance security and robustness of signatures, and have been a topic of a long line of research, with the recent interest driven by blockchain and other decentralized systems, motivating efficiency and security improvements, e.g. [50, 72, 73, 49, 21, 24, 69, 36, 4, 12, 34, 30], and a NIST call for standardization [16].

Blind signatures, introduced by Chaum [28], enable a server who holds the key, to issue a signature on a message held by a user, in a way that keeps the message hidden to the signer, and the user obtains only a signature on the single message which it specified in the protocol. Blind signatures are a powerful tool that enables many privacy-protecting applications, including ecash [28, 29, 76], anonymous credentials [18, 17], electronic voting [52], and blockchain transactions [57]. They are undergoing standardization efforts [40, 25, 6] and are used in products by Apple [7] and Google [51]. In all these applications the party who holds the signature key is the root of trust, and one

can make these schemes fault-tolerant and reduce the risk of key compromise, by secret-sharing this key and replacing the blind signature scheme with a *threshold blind signature scheme* (tBSig) [84, 70, 35].

BLS signatures [15], based on a bilinear map group, have the simplest (and round-minimal) threshold signature protocol and blind signature protocol, both proposed by Boldyreva[1] [14]. Vo et al. [84] observed that these two protocols can be naturally composed into a *threshold blind BLS* scheme, which is still round-minimal and essentially as efficient as the BLS signature itself. There are only three further works on provably secure threshold blind signatures. First, Kuchta and Manulis [70] construct threshold blind signatures secure under the Computational Diffie-Hellman (CDH) assumption, but it is much more computationally intensive than threshold blind BLS. Second, Doerner et al. [43] provide a threshold blind scheme that generates BBS+ signatures and is secure assuming the security of oblivious transfer. Third, Crites et al. [35] show a construction that is secure under the Discrete Logarithm (DL) assumption in the Algebraic Group Model (AGM). Only the first of these is round-optimal, and all three are shown secure only for a *static adversary*, who chooses the subset of corrupted parties before the protocol starts to operate.[2]

Threshold Oblivious PRF. An oblivious pseudorandom function (OPRF) is a protocol by which a user and server can jointly evaluate a PRF. Similar to a blind signature scheme, the user does not learn the PRF key and the server does not learn the PRF input. Applications include private set intersection [56, 62, 63, 68], password-protected secret sharing [11, 58, 59, 37], private information retrieval [45], password-authenticated key exchange (PAKE) [61], threshold PAKE [58, 53], and key management for outsourced file systems [60].

Among many OPRF realizations, see the survey on OPRF in [23], a popular low-cost approach is called "(Two-)Hashed Diffie-Hellman" (2HashDH) OPRF [58]. The 2HashDH OPRF of [58] is almost identical to Boldyreva's blind BLS, except that there is no pairing that allows for public verification of function output. Statically secure threshold OPRF (tOPRF) protocols, which compute 2HashDH OPRF in a threshold setting [59, 53], are likewise very similar to the (statically secure) threshold blind BLS [84].

Adaptive security. In the context of threshold cryptosystems an adaptive adversary model considers attackers who can corrupt up to a prescribed threshold of protocol participants, but can decide when and whom to corrupt at any time, possibly on the basis of protocol executions observed so far. By

[1] There are many other blind signature schemes, e.g. [77, 80, 2, 82, 35, 27, 65], but these are more complex and are not round-optimal. There are also alternative schemes using bilinear maps that are proven secure under weaker assumptions, e.g. [26, 55], but these schemes are significantly more computationally intensive than the blind BLS of Boldyreva and do not produce standard BLS signatures.

[2] Some other threshold blind signature schemes have been proposed under informal security models and proofs, e.g. [66, 78]. There have also been recent works targeting closely related notions such as threshold anonymous credentials, e.g. [79]. None of these works consider an adaptive adversary.

contrast, the static adversary model, where the attacker must decide which subset of participants to corrupt before the protocol starts, is an artificial restriction, without much practical justification beyond the fact that it makes security proofs easier. Indeed, constructing threshold cryptosystems, e.g. threshold signatures, with provable security against adaptive adversary has been a long-standing challenge, e.g. [22, 44, 74, 3, 5, 71], which has recently seen a renewed interest spiked by emerging practical applications of threshold cryptosystems, e.g. [8, 32, 38, 48, 9, 39]. We note that adaptively secure threshold protocols usually tend to be either more expensive or rely on stronger assumptions than schemes known to be statically secure. Our results follow this pattern, as we explain further below.

The applications of threshold *blind* signatures or threshold Oblivious PRF, would benefit from adaptive security because static security is an artificial constraint. If a protocol achieves adaptive security with only very mild impact on efficiency (as is the case for the protocols we show), such protocol will be preferable in practice because stronger confidence in application security will outweigh a mild efficiency impact. To pick just one example from the applications of blind signatures and threshold OPRF listed above, an adaptive tOPRF would imply adaptively secure proactive threshold PAKE, applying a black-box tOPRF-to-tPAKE compiler of [53].

We stress that until our work, there was *no threshold blind signature or threshold OPRF solutions which were shown adaptively secure*. Moreover, with regards specifically to BLS signatures, even for threshold *non-blind* BLS, adaptive security was not known until very recently. First, Bacho and Loss [8] proved adaptive security for Boldyreva's threshold BLS scheme, relying on the AGM and the One-More Discrete Log (OMDL) assumption. Even more recently, Das and Ren [39] showed a modified threshold BLS scheme, with a very mild impact on efficiency, which is adaptively secure under the Decisional Diffie-Hellman (DDH) and (co-)Computational Diffie-Hellman (CDH) assumptions, but not the AGM.

Our contributions. We provide the first tBSig and tOPRF schemes with a proof of adaptive security. Our proofs rely on the AGM and the OMDL, DDH, and Strong Discrete Log (SDL) assumptions in the Random Oracle Model (ROM). These dependencies are strong but justified in comparison to previous work. The One-More Diffie-Hellman (OMDH) assumption is necessary for standard, i.e. *single-server*, blind BLS [14] and 2HashDH OPRF [58], and OMDH implies SDL. The AGM and OMDL have been used by proofs of adaptively secure *non-blind* threshold signatures, including threshold BLS [8] and threshold Schnorr [34], and the only proof of *statically secure* tOPRF [53] uses DDH and OMDH asumptions[3], and all these works assume ROM. We note that our adaptively-secure tOPRF and BLS-based tBSig schemes impose only a small computational overhead over resp. the statically-secure tOPRF of [53] and statically-secure threshold blind BLS [14, 84].

[3] A secure tOPRF based on just OMDH was claimed by [59], but [53] showed an error in the proof of [59], and used the DDH assumption to fix it.

We note that we define adaptive tOPRF using game-based definitions comparable to the game-based definitions of threshold blind signatures [70, 33]. However, in Appendix D we show a simple extension of our game-based tOPRF which realizes an adaptively secure *Universally Composable (UC) tOPRF* formalized in [53], and computes the same PRF as the 2HashDH OPRF of [58]. Since our proofs are in the AGM, we show the above in the UC framework adapted to AGM by Abdalla et al. [1]. Specifically, using the terminology of [1], we show that our protocol $\mathbb{G}_1$-AGM *emulates* the UC tOPRF functionality of [53], where $\mathbb{G}_1$ is a prime-order group. In particular, we show a simulator which is $\mathbb{G}_1$-*algebraic*, i.e. it uses only algebraic operations on the same group, and the UC tOPRF functionality is algebraic as well, because it is independent of a group which the implementation uses.

From a technical standpoint, we build on a novel blinding technique introduced in two recent works, the adaptive threshold (but not blind) BLS signature of [39] and the (static) threshold OPRF of [53]. The technique adds a blinding factor to threshold exponentiation, by masking it with an exponentiation using an independent base, sampled via an RO hash, and exponents which form a zero-sharing (which causes the blinding factor to vanish after interpolation). The two works used this technique for very different purposes: [39] used it to prove adaptive security for *non-blind* threshold BLS, while [53] used it to eliminate attacks on *blind* threshold exponentiation/BLS, present already in the static setting. We combine and expand on both usages of these blinding factors (and we use two such factors instead of one) to realize adaptive threshold blind BLS and oblivious PRF, and our techniques (see below) may have further applications to threshold cryptosystems.

Lastly, throughout the paper our main focus is on tOPRF, including in our security notions, protocol presentation, and our security proofs, but throughout we mark as shadowed text the changes required to shift from tOPRF to tBSig. Indeed, one can see e.g. in our main protocol, shown in Figure 3, that these difference are small. Intuitively, tBSig differs from tOPRF only in the presence of the bilinear map which can be used for verification of the tOPRF output, which in the tBSig context is interpreted as a signature. The presence of this bilinear map doesn't have a major effect on most of our arguments, and we explain that this is so in more details in a subsection at the end of Section 4.

1.1 Technical Overview

In this section, we provide a high-level explanation of the main ideas in our work. As a jumping-off point, we begin with a simplified overview of the adaptively secure threshold *non-blind* BLS scheme of Das and Ren [39], and we overview the security challenges encountered by its threshold *blind* counterpart. Overcoming these challenges requires several novel techniques, which we also outline here.

Das-Ren argument for adaptive threshold *non-blind* BLS. Recall that the BLS signature is $y = H_0(x)^k$ where $\mathsf{pk} = g^k$ is the public key, k is the secret key, and H_0 is a hash onto group $\langle g' \rangle$ s.t. there is a bilinear map on $\langle g \rangle \times \langle g' \rangle$.

In Das and Ren's threshold BLS servers hold a Shamir secret-sharing $\{k_i\}$ of k and a sharing $\{z_i\}$ of $z = 0$. To sign message x, server i computes partial signature $q_i := H_0(x)^{k_i} \cdot H_1(x)^{z_i}$, where H_1 is another hash onto group $\langle g' \rangle$. Using interpolation in the exponent the user can aggregate q_i's to recover

$$y := \prod_{i \in \mathbf{S}} (q_i)^{\lambda_i} = H_0(x)^{\sum_{i \in \mathbf{S}} \lambda_i k_i} \cdot H_1(x)^{\sum_{i \in \mathbf{S}} \lambda_i z_i} = H_0(x)^k \cdot H_1(x)^0 = H_0(x)^k$$

for a large enough subset $\mathbf{S}$ and Lagrange interpolation coefficients $\{\lambda_i\}_{i \in \mathbf{S}}$. Note that the "blinding factors" $H_1(x)^{z_i}$ in the partial signatures disappear from the final signature, but their presence is integral to the proof of security.

In proving adaptive security, the reduction must be able to, upon corruption of any server, produce secret keys (k_i, z_i) that explain that server's previous partial signatures. This thwarts a simple Computational Diffie-Hellman (CDH) reduction that sets $\mathsf{pk} := g^\alpha$ where g^α is one of the two CDH challenge group elements, and uses trapdoors embedded in H_0 responses to compute partial signatures in spite of not knowing α, and therefore not knowing all k_i. Das and Ren's idea is that the reduction generates all k_i's honestly, but generates z_i's as a sharing of random z rather than $z = 0$, and embeds the CDH challenge in H_1 responses. Specifically, the reduction sets $\mathsf{pk} := g^k \cdot g^\alpha = g^{k+\alpha}$, and for all x queried to H_0 except one of them chosen at random, it sets $H_1(x)$ and $H_0(x)$ s.t. $H_1(x)^z = H_0(x)^\alpha$, and this way partial signatures $q_i = H_0(x)^{k_i} \cdot H_1(x)^{z_i}$ interpolate to $H_0(x)^k \cdot H_1(x)^z = H_0(x)^{k+\alpha}$. This correlated programming of H_0, H_1 is possible using only g^α and z (and if the reduction trapdoors H_0 outputs as $H_0(x) := g^{\tau_x}$ for random τ_x's), and it is undetectable under the DDH assumption. If the adversary corrupts only the allowed threshold of servers then the fact that $z \neq 0$ is hidden, and all appears as normal if the key was $k' = k + \alpha$. The other group element of the CDH challenge is programmed to $H_0(x^*) := g^\beta$, where x^* is randomly chosen from the queries to H_0, and if the adversary outputs forgery $y^* = H_0(x^*)^{k+\alpha}$ for the guessed argument x^*, the reduction can extract $g^{\alpha \cdot \beta}$ and thus win CDH.

Moving to threshold *blind* BLS. Using the "blind exponentiation" technique due to Chaum [28], either BLS signature or BLS threshold signature scheme can be made blind [14, 84]: The user computes $p := H_0(x)^r$ for randomly chosen r and sends it to the servers. The servers use p in place of $H_0(x)$ in their partial signatures, and the user exponentiates the aggregated result to $1/r$ to arrive at the final signature. To make Das and Ren's threshold BLS scheme blind, the servers can use p instead of x in their blinding factors, i.e. set $q_i := p^{k_i} \cdot H_1(p)^{z_i}$.

At first glance, it might seem that Das and Ren's proof techniques are still immediately applicable to this *blind* threshold BLS scheme, but this is not the case. In the newly blinded scheme an adversarial user can query the servers using arbitrary p values, and this introduces major complications in the proof of security. There are four types of adversarial query behaviors that our proof must handle. Below we summarize these patterns, along with sketching the proof techniques we use to show that the scheme is secure in spite of them.

#1: Blinded Queries. Suppose the reduction programs $H_0(x) := g^{\tau_x}$ for a random trapdoor τ_x. An honest user will then pick random r and send $p := H_0(x)^r = g^{\tau_x \cdot r}$ to the servers. If our reduction were to follow the above sketch, it would have to program $H_1(p) := p^{\alpha/z} = (g^\alpha)^{(\tau_x \cdot r)/z}$ given the CDH challenge g^α, but this seems hard because p is a random group element, and it is not clear how the reduction could link it to exponents τ_x and r.

We resort to the Algebraic Group Model (AGM) to solve this: If the adversary must accompany $p = H_0(x)^r$ with its algebraic representation, i.e. base $H_0(x)$ and exponent r, then the reduction learns x, r and can set $H_1(p) := (g^\alpha)^{(\tau_x \cdot r)/z}$.

#2: Re-Randomized Queries. Suppose the adversary is an honest user who re-uses some input x, i.e. it sets $p_1 := H_0(x)^{r_1}$ and $p_2 := H_0(x)^{r_2}$ for random r_1, r_2. If $H_1(p_1) = (p_1)^{\alpha/z}$ and $H_1(p_2) = (p_2)^{\alpha/z}$ as above, the adversary can detect that these responses are not random because $H_1(p_1)^{1/r_1} = H_1(p_2)^{1/r_2}$.

We solve this problem using a second zero-sharing $\{\hat{z}_i\}$ and making the partial signatures include two blinding factors, i.e. $q_i := p^{k_i} \cdot H_1(p)^{z_i} \cdot H_2(p)^{\hat{z}_i}$. In the reduction shares z_i and $\hat{z}_i$ interpolate to random z and $\hat{z}$, and the reduction programs H_1, H_2 in a correlated way that achieves the same result as before, setting $H_1(p) := p^{\delta_p \cdot \alpha/z}$ and $H_2(p) := p^{(1-\delta_p) \cdot \alpha/\hat{z}}$, where δ_p is randomly chosen for each p. This way q_i's still interpolate to $p^{k+\alpha}$, and under the DDH assumption this correlated programming remains undetectable to the adversary.

#3: Mixed Queries. Honest users use only a single $H_0(x)$ base to form p. However, an adversary can instead combine many H_0 outputs to form $p := H_0(x_1)^{r_1} \cdot ... \cdot H_0(x_L)^{r_L}$. This strategy is not necessarily self-defeating: If the adversary forms L queries $p_1, ..., p_L$, each p_i using a different randomizer vector $r_{i,1}, ..., r_{i,L}$, and sends each such p_i to $t+1$ servers and interpolates these answers into p_i^k, then it can extract $H_0(x_1)^k$, ..., $H_0(x_L)^k$ as long as these randomizer vectors are linearly independent. This is not a wasteful strategy because it extracts signatures on L values using $L \cdot (t+1)$ queries. Das and Ren's CDH reduction relies upon guessing a single forgery target x^* and embedding the CDH challenge (rather than a trapdoor) into $H_0(x^*)$, but in our case, the above adversary can make $H_0(x^*)$ a component in every query p, and if it contains a challenge, and not a trapdoor, the reduction wouldn't know how to respond (in particular, it could not set $H_1(p)$ and $H_2(p)$ as above).

We circumvent this problem by replacing the above CDH reduction with two reductions, to resp. One-More Discrete Logarithm (OMDL) and Discrete Logarithm (DL). In the first reduction, to OMDL, we embed the t OMDL challenges into the blinding factor shares $z_i, \hat{z}_i$, while the k_i shares are picked honestly. The reduction only knows g^{z_i} and $g^{\hat{z}_i}$, but this is sufficient to compute partial signatures due to the embedding of trapdoors in H_1 and H_2 responses. (This OMDL reduction programs H_1 and H_2 outputs differently than described above, because at this point in the argument the main key is no longer "rigged" as in the Das-Ren reduction described above, and pk is simply set as g^k for k known to the reduction.) Upon server corruption, the DL oracle is used to find the correct $z_i, \hat{z}_i$. We use this OMDL reduction to argue that the adversary must form its final signatures via correct interpolation of partial

signatures on the same query p; otherwise, the reduction learns a system of equations that is sufficient to solve for some z_i or $\hat{z}_i$. Consequently, any p that is not queried to at least $t+1$ servers (minus the number of eventually corrupted servers) is useless to the adversary. The above step conceptually simplifies the multiple servers setting into a single signing entity. Moreover, the same argument ensures that the adversary's final signatures are accompanied by algebraic representations which involve $t+1$ partial signatures for any meaningfully used server query p.

In the second step, the reduction solves DL if the adversary computes the signatures on L arguments but makes less than $L \cdot (t+1)$ partial signature queries (it is $t+1$ minus the number of eventually corrupted parties, but we omit that for simplicity). This reduction proceeds similarly as the reduction described in part #2 above, but the only challenge is g^α: The reduction picks all $k_i, z_i, \hat{z}_i$ shares and correlates H_1, H_2 responses as the reduction in part #2. However, unlike in Das and Ren's proof, this reduction trapdoors all H_0 outputs (in particular there is no x^* for which $H_0(x^*)$ is programmed differently), and we show that if the adversary's algebraic representations explain L correct signatures using less than L (times $t+1$) different p queries, then these representations provide a system of equations by which the reduction can solve for α.

#4: Recursive Queries. Lastly, the adversary might perform recursive queries, i.e. the adversary can use a server's response as part of the input to a new server query. Similarly, the adversary can use H_1 and H_2 outputs to form server queries (and/or new H_1 and H_2 inputs). Recursive and mixed querying behaviors might even occur in tandem, creating bizarre queries like $p = H_0(x_1) \cdot H_1(H_0(x_1) \cdot H_1(H_0(x_2)))$. Unlike the other three querying behaviors, we have no concrete example of an adversary that would act in these ways and succeed; nonetheless, our reductions must be able to handle this behavior.

To solve this problem, we upgrade the aforementioned DL reduction into a q-Strong DL (q-SDL) reduction. In the q-SDL problem, the reduction is given not just g^α, but also $g^{\alpha^2}, g^{\alpha^3}, \ldots, g^{\alpha^q}$. These higher powers can be used by the reduction to compute proper H_1 and H_2 responses to recursive random oracle queries. The q-SDL reduction knows all secret keys $\{k_i, z_i, \hat{z}_i\}$, so there is no further problem handling recursive queries to the servers.

2 Preliminaries

2.1 Group Setting and Hardness Assumptions

We recall the setting of a bilinear pairing group needed for BLS signatures, and the assumptions used in our security proofs.

Definition 1 (Bilinear Pairing). *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of prime order m with generators g_1, g_2, g_T, respectively. A* bilinear pairing *is an efficiently computable function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the property that $e((g_1)^a, (g_2)^b) = (g_T)^{ab}$ for all $a, b \in \mathbb{Z}_m$.*

In a type 1 (or "symmetric") pairing setup, there are efficiently computable homomorphisms from $\mathbb{G}_1$ to $\mathbb{G}_2$ and from $\mathbb{G}_2$ to $\mathbb{G}_1$ (it may be that $\mathbb{G}_1 = \mathbb{G}_2$). In a type 2 pairing setup, there is an efficiently computable homomorphism only from $\mathbb{G}_2$ to $\mathbb{G}_1$. In a type 3 pairing setup, there is no efficiently computable homomorphism in either direction.

Definition 2 (Discrete Logarithm Problem). *Let $\mathbb{G}$ be a cyclic group of prime order m with generator g. The* Discrete Logarithm problem (DL) *on $\mathbb{G}$ is, given (g, g^ξ) where $\xi \leftarrow_\$ \mathbb{Z}_m$, to find ξ. We say that DL is (ϵ, T)-hard in $\mathbb{G}$ if all algorithms running in time T have no better than ϵ probability of solving the DL problem.*

Definition 3 (N-Relational Discrete Logarithm Problem). *Let $\mathbb{G}$ be a cyclic group of prime order m with generator g. The* N-Relational Discrete Logarithm problem (N-RDL) *on $\mathbb{G}$ is, given $(g^{\xi_1}, \ldots, g^{\xi_N})$ where $\xi_1, \ldots, \xi_N \leftarrow_\$ \mathbb{Z}_m$, to find $c_1, \ldots, c_N \in \mathbb{Z}_m$ such that $\prod_{i\in[N]}(g^{\xi_i})^{c_i} = 1$ and $\exists_{i\in[N]}\ c_i \neq 0$. We say that N-RDL is (ϵ, T)-hard in $\mathbb{G}$ if all algorithms running in time T have no better than ϵ probability of solving the N-RDL problem.*

Lemma 1. *If DL is (ϵ, T)-hard in $\mathbb{G}$, then N-RDL is $(\frac{m}{m-1} \cdot \epsilon, \mathcal{O}(T - N \log m))$-hard in $\mathbb{G}$.*

The equivalence between the DL and N-RDL problems is well-known [67]. We include a brief proof of Lemma 1 in Appendix A. For brevity, we will omit the factor $\frac{m}{m-1} \approx 1$ throughout the rest of this work.

Definition 4 (q-Strong Discrete Logarithm Problem). *Let $\mathbb{G}$ be a cyclic group of prime order m with generator g. The* q-Strong Discrete Logarithm problem (q-SDL) *on $\mathbb{G}$ is, given $(g, g^\xi, g^{\xi^2}, \ldots, g^{\xi^q})$ where $\xi \leftarrow_\$ \mathbb{Z}_m$, to find ξ. We say that q-SDL is (ϵ, T)-hard in $\mathbb{G}$ if all algorithms running in time T have no better than ϵ probability of solving the q-SDL problem.*

DL is a special case of q-SDL where $q = 1$.

Proving security of our pairing-based signature scheme will require a different version of q-SDL adapted to that setting.

Definition 5 (q-Strong co-Discrete Logarithm Problem). *Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic groups of prime order m with generators g_1, g_2, respectively. The* q-Strong co-Discrete Logarithm problem (q-co-SDL) *on $(\mathbb{G}_1, \mathbb{G}_2)$ is, given $(g_1, (g_1)^\xi, (g_1)^{\xi^2}, \ldots, (g_1)^{\xi^q}, g_2, (g_2)^\xi)$ where $\xi \leftarrow_\$ \mathbb{Z}_m$, to find ξ. We say that q-co-SDL is (ϵ, T)-hard in $(\mathbb{G}_1, \mathbb{G}_2)$ if all algorithms running in time T have no better than ϵ probability of solving the q-co-SDL problem.*

We note that our variant of the q-co-SDL assumption is weaker than the one that has appeared in previous pairing-based works, e.g. [54, 46, 83, 31, 86], as our version provides only $(g_2)^\xi$ and not all q higher powers $(g_2)^{\xi^2}, \ldots, (g_2)^{\xi^q}$.

Definition 6 (t-One-More Discrete Logarithm Problem). *Let $\mathbb{G}$ be a cyclic group of prime order m with generator g. The* t-One-More Discrete Logarithm problem (t-OMDL) *on $\mathbb{G}$ is, given $(g, g^{\xi_1}, \ldots, g^{\xi_t})$ where $\xi_1, \ldots, \xi_t \leftarrow_\$ \mathbb{Z}_m$ and given access to discrete logarithm oracle $\mathsf{DL}_g(\cdot)$, to find all $\xi_1, \ldots, \xi_t$ while making at most $t-1$ calls to $\mathsf{DL}_g(\cdot)$. We say that t-OMDL is (ϵ, T)-hard in $\mathbb{G}$ if all algorithms running in time T have no better than ϵ probability of solving the t-OMDL problem.*

DL is a special case of t-OMDL where $t = 1$.

Definition 7 (Decisional Diffie-Hellman Problem). *Let $\mathbb{G}$ be a cyclic group of prime order m with generator g. The* Decisional Diffie-Hellman problem (DDH) *on $\mathbb{G}$ is to distinguish Diffie-Hellman tuples from random tuples. In particular, the DDH advantage of an algorithm $\mathcal{A}$ is defined as:*

$$\left| \Pr_{a,b \leftarrow_\$ \mathbb{Z}_m} [\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr_{a,b,c \leftarrow_\$ \mathbb{Z}_m} [\mathcal{A}(g, g^a, g^b, g^c) = 1] \right|$$

We say that DDH is (ϵ, T)-hard in $\mathbb{G}$ if all algorithms running in time T have no better than ϵ DDH advantage.

2.2 Threshold Oblivious Pseudorandom Function

Threshold OPRF was defined in the UC setting in [59, 53] but here we provide simpler game-based security definitions for tOPRF. We do so for three reasons: First, to break down the monolithic UC definition to several easier to understand properties, e.g. to see which assumption is used to achieve which property. Second, to relate the tOPRF properties to the corresponding properties of threshold Blind Signatures in Section 2.3. Third, in Appendix D we show that a scheme that satisfies the game-based tOPRF properties can be generically modified, by hashing the tOPRF input and output via a (Random Oracle) hash, to realize the UC tOPRF functionality of [53].

Throughout this work we refer to the two types of parties in our schemes as *users* and *servers*. Our definitions are simplified for the case of non-interactive, i.e. single-round, protocols, but they can be easily extended to the general case where UEval and SEval consist of multiple sequential rounds.

Definition 8 (tOPRF). *A* threshold oblivious pseudorandom function *with domain $\mathcal{X}$ and codomain $\mathcal{Y}$ is a tuple of PPT algorithms* $\mathsf{tOPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{UEval}, \mathsf{SEval}, \mathsf{UAgg}, \mathsf{Offline})$ *with the following properties:*

1. $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$. *Given an input of security parameter length, the setup algorithm selects public parameters* pp *for the protocol. The output of* Setup *is an implicit input to all other algorithms.*
2. $\mathsf{KeyGen}(n, t) \to (\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{aux})$. *Given the number of servers n and threshold t, the key generation algorithm produces a secret key sk_i for each server and auxiliary public data* aux.

3. $\mathsf{UEval}(x, \mathsf{aux}) \to (\mathsf{st}^{\mathsf{U}}, \mathsf{pm}^{\mathsf{U}})$. *Given function input* $x \in \mathcal{X}$ *and public data* aux*, the user evaluation algorithm produces a message* pm^{U} *sent to all servers, and an internal state* st^{U} *to be used later when aggregating the server responses.*
4. $\mathsf{SEval}(i, \mathsf{sk}_i, \mathsf{ctx}, \mathsf{pm}^{\mathsf{U}}) \to \mathsf{pm}^i$. *Given server* i*'s secret key* sk_i*, server context data* ctx*, and user message* pm^{U}*, the server evaluation algorithm produces a response message* pm^i*.*
5. $\mathsf{UAgg}(\mathsf{st}^{\mathsf{U}}, \mathbf{S}, \{\mathsf{pm}^i\}_{i \in \mathbf{S}}) \to y/\perp$. *Given a state from* UEval *and server responses* $\{\mathsf{pm}^i\}_{i \in \mathbf{S}}$ *for a set of* $t+1$ *servers* $\mathbf{S}$*, the user aggregation algorithm produces either a function output* $y \in \mathcal{Y}$ *or a failure symbol* $\perp$*.*
6. $\mathsf{Offline}(\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{aux}, x) \to y$. *Given an output of* KeyGen *and input* $x \in \mathcal{X}$*, the offline evaluation algorithm produces a function output* $y \in \mathcal{Y}$*.*

tOPRF *must further satisfy* Deterministic Correctness*,* Pseudorandomness*,* Blindness*, and* One-More Unpredictability*, as defined below.*

Correctness. Deterministic Correctness ensures that a properly performed function evaluation on any given input always produces the same output, i.e. that the protocol computes a function defined by KeyGen outputs.

Definition 9 (Deterministic Correctness). *Consider the following experiment* $(*)$*, which takes as input security parameter* κ*, integers* n *and* t *s.t.* $0 \leq t < n$*,* $x \in \mathcal{X}$*, set* $\mathbf{S} \subseteq [n]$ *s.t.* $|\mathbf{S}| = t+1$*, and* $\mathsf{ctx} \in \{0,1\}^*$*:*

$$
\begin{aligned}
\mathsf{pp} &\leftarrow \mathsf{tOPRF.Setup}(1^\kappa) \\
(\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{aux}) &\leftarrow \mathsf{tOPRF.KeyGen}(n, t) \\
(\mathsf{st}^{\mathsf{U}}, \mathsf{pm}^{\mathsf{U}}) &\leftarrow \mathsf{tOPRF.UEval}(x, \mathsf{aux}) \\
\forall_{i \in \mathbf{S}} \qquad \mathsf{pm}^i &\leftarrow \mathsf{tOPRF.SEval}(i, \mathsf{sk}_i, \mathsf{ctx}, \mathsf{pm}^{\mathsf{U}}) \\
y &\leftarrow \mathsf{tOPRF.UAgg}(\mathsf{st}^{\mathsf{U}}, \mathbf{S}, \{\mathsf{pm}^i\}_{i \in \mathbf{S}})
\end{aligned}
$$

Threshold oblivious pseudorandom function tOPRF *is* deterministically correct *iff algorithm* $\mathsf{tOPRF.Offline}$ *is deterministic and, on every possible input, experiment* $(*)$ *always produces* $(\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{aux}, y)$ *such that* $y = \mathsf{tOPRF.Offline}(\{\mathsf{sk}_i\}_{i \in [n]}, \mathsf{aux}, x)$*.*

Robustness. We define a robustness property, which ensures that dishonest servers cannot trick a user into performing an incorrect evaluation. In works on OPRFs, e.g. [45, 58], this property is referred to as *verifiability*, but we choose the term *robustness* to avoid confusion with the verifiability property of a signature scheme (which is different).

Definition 10 (Robustness). *The robustness game* $\mathbf{Game}^{\mathsf{robust}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ *is defined as:*

$$
\begin{aligned}
\mathsf{pp} &\leftarrow \mathsf{tOPRF.Setup}(1^\kappa) \\
(n, t, \mathsf{st}^{\mathcal{A}}) &\leftarrow \mathcal{A}(1^\kappa) \\
(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}) &\leftarrow \mathsf{tOPRF.KeyGen}(n, t) \\
(x, \mathsf{st}^{\mathcal{A}}) &\leftarrow \mathcal{A}(\mathsf{st}^{\mathcal{A}}, \{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}) \\
(\mathsf{st}^{\mathsf{U}}, \mathsf{pm}^{\mathsf{U}}) &\leftarrow \mathsf{tOPRF.UEval}(x, \mathsf{aux}) \\
(\mathbf{S}, \{\mathsf{pm}^i\}_{i\in\mathbf{S}}) &\leftarrow \mathcal{A}(\mathsf{st}^{\mathcal{A}}, \mathsf{pm}^{\mathsf{U}}) \\
y &\leftarrow \mathsf{tOPRF.UAgg}(\mathsf{st}^{\mathsf{U}}, \mathbf{S}, \{\mathsf{pm}^i\}_{i\in\mathbf{S}})
\end{aligned}
$$

The advantage of adversary $\mathcal{A}$ in the robustness game is defined as:

$$\mathbf{Adv}^{\mathsf{robust}}_{\mathcal{A},\mathsf{tOPRF}} = \Pr[y \neq \perp \wedge y \neq \mathsf{tOPRF.Offline}(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}, x)]$$

where $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}, x, y) \leftarrow \mathbf{Game}^{\mathsf{robust}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$.

Threshold oblivious pseudorandom function tOPRF *is* robust *iff, for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{robust}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ is negligible.*

Pseudorandomness. We define tOPRF pseudorandomness using the standard definition for PRF Pseudorandomness applied to the function computed in the honest protocol execution.

Definition 11 (Pseudorandomness). *The advantage of adversary $\mathcal{A}$ against protocol* tOPRF *in the pseudorandomness game is defined as:*

$$\mathbf{Adv}^{\mathsf{pseudorandom}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = |\Pr[\mathcal{A}^{\mathsf{tOPRF.Offline}(\{\mathsf{sk}_i\}_{i\in[n]},\mathsf{aux},\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\kappa) = 1]|$$

where in the first probability $\mathsf{pp} \leftarrow \mathsf{tOPRF.Setup}(1^\kappa)$ *and* $(\{sk_i\}_{i\in[n]}, \mathsf{aux}) \leftarrow \mathsf{tOPRF.KeyGen}(n, t)$ *and in the second probability f is a uniformly chosen function $\mathcal{X} \to \mathcal{Y}$. Before beginning, $\mathcal{A}$ outputs integers n and t s.t. $0 \leq t < n$.*

Threshold oblivious pseudorandom function tOPRF *is* pseudorandom *iff, for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{pseudorandom}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ is negligible.*

Blindness and Obliviousness. We next define the two closely related properties of Blindness and Obliviouness. The Blindness property ensures that servers do not learn the function inputs that users are evaluating. Even if function inputs and outputs are later revealed, the servers still cannot link those evaluations to the particular sessions that produced them. This property is required to hold even in the case that all servers collude. This definition has appeared before in [70, 33].

Definition 12 (Blindness). *The advantage of adversary $\mathcal{A}$ against protocol* tOPRF *in the blindness game $\mathbf{Game}^{\mathsf{blind}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ (see Figure 1), is defined as:*

$$\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = |\Pr[\mathbf{Game}^{\mathsf{blind}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = 1] - 1/2|$$

Threshold oblivious pseudorandom function tOPRF *is* blind *iff, for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ is negligible.*

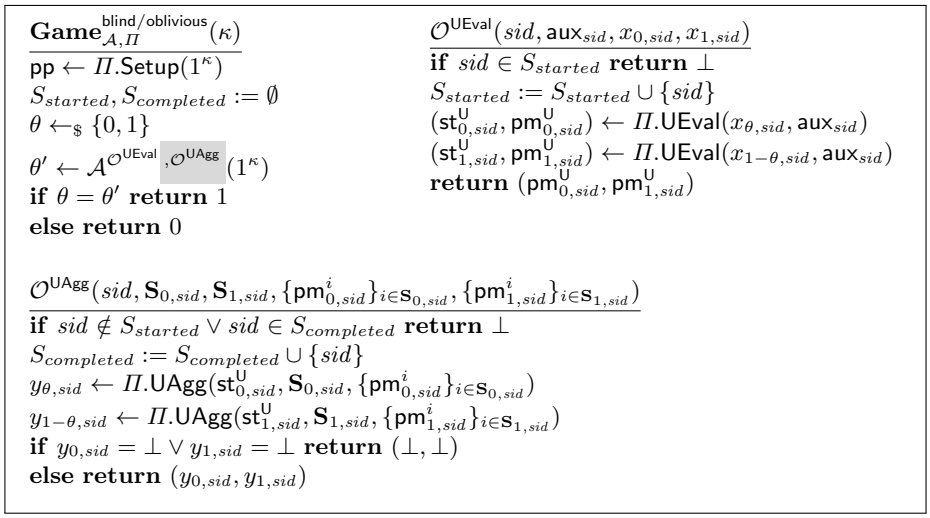

Fig. 1. The blindness game $\mathbf{Game}^{\mathsf{blind}}_{\mathcal{A},\Pi}(\kappa)$ (include shadowed text) and the obliviousness game $\mathbf{Game}^{\mathsf{oblivious}}_{\mathcal{A},\Pi}(\kappa)$ (omit shadowed text) for a threshold oblivious pseudorandom function or threshold blind signature scheme.

The Obliviousness property, an extension of the corresponding property of single-server OPRFs, is a weaker variant of blindness, where $\mathcal{A}$ has no access to $\mathcal{O}^{\mathsf{UAgg}}$. In other words, the adversary does not get to see the user's eventual function outputs. Consequentially, a scheme which is oblivious but not blind could allow the adversary to rig some evaluation sessions to produce incorrect outputs: Since those outputs are not revealed to the adversary, this rigging doesn't help the adversary discern which inputs correspond to which sessions.

Definition 13 (Obliviousness). *The advantage of adversary $\mathcal{A}$ against protocol* tOPRF *in the obliviousness game* $\mathbf{Game}^{\mathsf{oblivious}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ *(see Figure 1), is defined as:*

$$\mathbf{Adv}^{\mathsf{oblivious}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = |\Pr[\mathbf{Game}^{\mathsf{oblivious}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = 1] - 1/2|$$

Threshold oblivious pseudorandom function tOPRF *is* oblivious *iff, for all PPT adversaries $\mathcal{A}$,* $\mathbf{Adv}^{\mathsf{oblivious}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ *is negligible.*

Blindness implies Robustness. Blindness is achievable only if the user holds the public data aux. Otherwise the user could not detect the session-rigging attack described above. This provides a hint that blindness is closely related to the user's ability to verify the correctness of the messages $\{\mathsf{pm}^i\}_{i\in\mathbf{S}}$ received from the servers, i.e. that blindness implies robustness. To see this, suppose there existed an adversary $\mathcal{A}$ that broke robustness. An adversary in the blindness game could run $\mathcal{A}$ as a subroutine and use the resulting $(\mathsf{aux}, x, \mathbf{S}, \{\mathsf{pm}_i\}_{i\in\mathbf{S}})$ as input 0 to $\mathcal{O}^{\mathsf{UEval}}$ and $\mathcal{O}^{\mathsf{UAgg}}$. For input 1, the adversary could use the same x, but coupled with honest SEval responses. With non-negligible probability, the UEval and UAgg executions faced with input 0 will produce an incorrect function

output. The executions faced with input 1 will (by determinstic correctness) always produce a correct function output. The adversary, who sees these outputs, clearly has non-negligible advantage in discerning which one corresponds to the session that was rigged.

Although verifiability, i.e. robustness, is a valued property for OPRFs, many applications require users to evaluate OPRFs without having reliable access to trusted public data aux. We explain in Appendix B how our tOPRF scheme gracefully degrades from blind to merely oblivous when users run without input aux (in which case robustness is lost as well).

One-More Unpredictability. Finally we define One-More Unpredictability, which ensures that users cannot evaluate the function on more inputs than allowed by the servers. Specifying precisely how many evaluations should be allowed after a given series of server actions is not trivial.

In the realm of non-blind threshold signatures, Bellare et al. [13] recently introduced a hierarchy of unforgeability definitions and observed that many past works only target the weakest possible notion, labeled *UF-0*. Under UF-0, an adversary may be able to compute a signature on a message after the participation of only a single honest server. Except in the case that t servers are corrupted, the security guarantee of UF-0 is obviously much weaker than one would expect from a threshold cryptosystem. Their next definition, *UF-1*, codifies the more intuitive notion that $t + 1$ servers must sign a message in order to create a signature for it. The hierarchy continues to *UF-2*, which specifies that $t + 1$ servers must sign a message and must see the same "leader request", which may contain some extra data in addition to the message itself.

Since we must consider blind/oblivious evaluation, the task of defining *One-More* Unpredictability (or Unforgeability) is even more nuanced. All previous works on threshold blind signatures either (i) only considered the case where t servers are corrupted [84, 70] or (ii) used a very loose game-based definition under which an adversary may be able to compute signatures with only a single server's participation [33]. These works target a security notion roughly analagous to UF-0. Previous works on threshold oblivious PRFs [59, 53] have used simulation-based security definitions that are stronger, requiring that each function evaluation "uses up" a set of $t + 1$ server participation "tickets". Though not directly comparable, such a notion may be considered on par with UF-1. We define One-More Unpredictability in a way that is stronger still and roughly analagous to UF-2. In particular, we require $t + 1$ servers to participate in evaluations on the same input $(\mathsf{ctx}, \mathsf{pm}^{\mathsf{U}})$. This ctx-binding property was proposed for tOPRFs in [53] and is useful to applications that wish to enforce server agreement on arbitrary side data, e.g. a timestamp or user ID.

Definition 14 (One-More Unpredictability). *The advantage of adversary $\mathcal{A}$ against protocol* tOPRF *in the one-more unpredictability game* $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ *(as defined in Figure 2), is defined as:*

$$\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = \Pr[\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa) = 1]$$

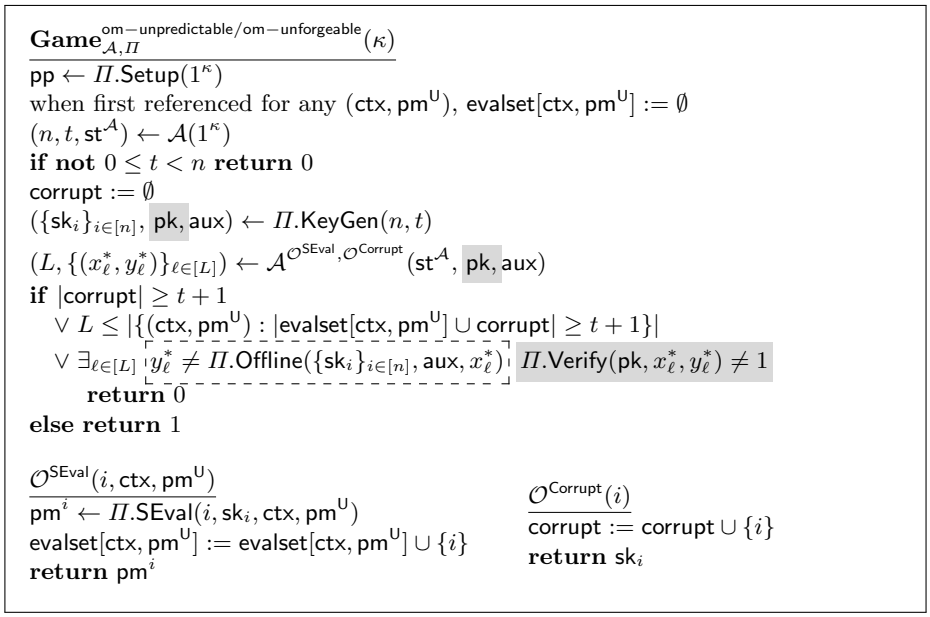

Fig. 2. The one-more unpredictability game $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ for a threshold oblivious pseudorandom function (dashed text) and the one-more unforgeability game $\mathbf{Game}^{\mathsf{om-unforgeable}}_{\mathcal{A},\mathsf{tBSig}}(\kappa)$ for a threshold blind signature scheme (shadowed text).

Threshold oblivious pseudorandom function tOPRF *is* one-more unpredictable *iff, for all PPT adversaries* $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{tOPRF}}(\kappa)$ *is negligible.*

2.3 Threshold Blind Signature

In this section we define Threshold Blind Signature (tBSig), a functionality very similar to tOPRF. Unlike a tOPRF, a tBSig need not be deterministic nor pseudorandom, but it must be publicly verifiable.

Definition 15 (tBSig). *A* threshold blind signature scheme *with message space* $\mathcal{X}$ *and signature space* $\mathcal{Y}$ *is a tuple of PPT algorithms* $\mathsf{tBSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{UEval}, \mathsf{SEval}, \mathsf{UAgg}, \mathsf{Verify})$ *with the following properties:*

1. $\mathsf{Setup}(1^\kappa) \to \mathsf{pp}$.
2. $\mathsf{KeyGen}(n, t) \to (\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pk}, \mathsf{aux})$. *The key generation algorithm outputs a public verification key* pk.
3. $\mathsf{UEval}(x, \mathsf{pk}, \mathsf{aux}) \to (\mathsf{st}^\mathsf{U}, \mathsf{pm}^\mathsf{U})$.
4. $\mathsf{SEval}(i, \mathsf{sk}_i, \mathsf{ctx}, \mathsf{pm}^\mathsf{U}) \to \mathsf{pm}^i$.
5. $\mathsf{UAgg}(\mathsf{st}^\mathsf{U}, \mathbf{S}, \{\mathsf{pm}^i\}_{i\in\mathbf{S}}) \to y/\perp$.
6. $\mathsf{Verify}(\mathsf{pk}, x, y) \to 0/1$. *Given a public key* pk*, a message* $x \in \mathcal{X}$*, and a purported signature* $y \in \mathcal{Y}$*, the verification algorithm confirms whether the signature is valid.*

A tBSig is further required to satisfy Verifiable Correctness*,* Blindness*, and* One-More Unforgeability*, as defined below.*

Verifiable Correctness ensures that a properly performed signing process on any given message really does produce a valid signature.

Definition 16 (Verifiable Correctness). *Consider the following experiment* $(*)$*, which takes as input security parameter* κ*, integers* n *and* t *s.t.* $0 \leq t < n$*,* $x \in \mathcal{X}$*, set* $\mathbf{S} \subseteq [n]$ *s.t.* $|\mathbf{S}| = t + 1$*, and* $\mathsf{ctx} \in \{0,1\}^*$*:*

$$\begin{aligned} \mathsf{pp} &\leftarrow \mathsf{tBSig.Setup}(1^\kappa) \\ (\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pk}, \mathsf{aux}) &\leftarrow \mathsf{tBSig.KeyGen}(n, t) \\ (\mathsf{st}^{\mathsf{U}}, \mathsf{pm}^{\mathsf{U}}) &\leftarrow \mathsf{tBSig.UEval}(x, \mathsf{aux}) \\ \forall_{i\in\mathbf{S}} \quad \mathsf{pm}^i &\leftarrow \mathsf{tBSig.SEval}(i, \mathsf{sk}_i, \mathsf{ctx}, \mathsf{pm}^{\mathsf{U}}) \\ y &\leftarrow \mathsf{tBSig.UAgg}(\mathsf{st}^{\mathsf{U}}, \mathbf{S}, \{\mathsf{pm}^i\}_{i\in\mathbf{S}}) \end{aligned}$$

Threshold blind signature scheme tBSig *is* verifiably correct *iff, on every possible input, experiment* $(*)$ *always produces* (pk, y) *such that* $\mathsf{tBSig.Verify}(\mathsf{pk}, x, y) = 1$*.*

The Blindness property is identical to the tOPRF Blindness property.

Definition 17 (Blindness). *Threshold blind signature scheme* tBSig *is* blind *iff* $(\mathsf{tBSig.Setup}, \mathsf{tBSig.KeyGen}, \mathsf{tBSig.UEval}, \mathsf{tBSig.SEval}, \mathsf{tBSig.UAgg})$ *satisfies threshold oblivious pseudorandom function blindness (Definition 12).*

Finally we consider One-More Unforgeability, which we define almost identically to tOPRF One-More Unpredictability. We merely alter the adversary's win condition in the game to match tBSig's different notion of correctness. Note that this definition enforces "strong" unforgeability, under which an adversary must be unable to re-randomize a signature on some message into a different signature on the same message. For "weak" unforgeability, $\mathbf{Game}^{\mathsf{om-unforgeable}}$ could be amended so that the win condition requires every x_k^* to be unique.

Definition 18 (One-More Unforgeability). *The advantage of adversary* $\mathcal{A}$ *against protocol* tBSig *in the one-more unforgeability game* $\mathbf{Game}^{\mathsf{om-unforgeable}}_{\mathcal{A},\mathsf{tBSig}}(\kappa)$ *(as defined in Figure 2), is defined as:*

$$\mathbf{Adv}^{\mathsf{om-unforgeable}}_{\mathcal{A},\mathsf{tBSig}}(\kappa) = \Pr[\mathbf{Game}^{\mathsf{om-unforgeable}}_{\mathcal{A},\mathsf{tBSig}}(\kappa) = 1]$$

Threshold blind signature scheme tBSig *is* one-more unforgeable *iff, for all PPT adversaries* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathsf{om-unforgeable}}_{\mathcal{A},\mathsf{tBSig}}(\kappa)$ *is negligible.*

3 The 2B–HashTDH and 2B–tBlindBLS Protocols

Figure 3 presents our main protocols, the threshold oblivious pseudorandom function scheme 2B–HashTDH ("twice-blind threshold hashed Diffie-Hellman")

and the threshold blind signature scheme 2B–tBlindBLS ("twice-blind threshold blind BLS"). The term "twice-blind" refers to the *two* blinding factors applied to the server responses.

Note that scheme 2B–HashTDH is a threshold oblivious evaluation of function $F_k(x) = H_0(x)^k$. In Appendix D we show that a slight variant of this scheme, where the final output is re-defined as $F'_k(x) = \tilde{H}(x, F_k(x))$ for a random oracle hash $\tilde{H}$, realizes the universally composable tOPRF notion of [53].

The protocols in Figure 3 utilize two non-interactive zero knowledge proof of knowledge systems (NIZKs), denoted $\mathsf{NIZK}_{\mathsf{KeyGen}}$ and $\mathsf{NIZK}_{\mathsf{SEval}}$. Intuitively, the former ensures that public keys are well-formed, and the latter ensures that servers behave honestly. We formally define these NIZKs in Appendix B; both have standard DL-based realizations. Note that they are only relevant to the proof of blindness and play no role in one-more unpredictability/unforgeability.

Our schemes differ from the simplest tOPRF and tBSig protocols [59, 84] only in the addition of the two server response blinding factors and the NIZKs. The former come at no cost to the users and a low cost to servers: just the storage of key shares z_i and $\hat{z}_i$, computing two hash onto a group operations, and replacing an exponentiation with a multi-exponentiation on three bases. The blinding factors' importance is explained in detail in the technical overview (Section 1.1). Our NIZKs impose a more significant (though still reasonable) overhead, but in section B.1, we explain that the cost of NIZKs can be avoided or mitigated in many important cases.

4 Proof of Adaptive Security

To prove the security of our schemes, we begin by showing that 2B–HashTDH and 2B–tBlindBLS are secure tOPRFs. A simple lemma then implies that 2B–tBlindBLS is also a secure tBSig.

Correctness, Pseudorandomness, and Blindness. It is easy to see that the schemes satisfy deterministic correctness (Definition 9); the function they compute is $\mathsf{Offline}(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}, x) = H_0(x)^k$ where k is the secret key that interpolates from $\{k_i\}_{i\in[n]}$. It is also apparent that, under the DDH assumption, this function is pseudorandom (Definition 11); we omit the proof of this fact, which is a well-known and straightforward DDH reduction.

We also wish to prove that 2B–HashTDH and 2B–tBlindBLS are blind. Doing so does not require any novel techniques and is not related to the challenge of adaptive corruptions, so we defer this proof to Appendix C.

One-More Unpredictability. Here we tackle the main challenge of our work: proving the one-more unpredictability of 2B–HashTDH against an adaptive adversary (which will imply the one-more unforgeability of 2B–tBlindBLS). This proof requires very slight differences (mostly in Lemma 2) for the case of 2B–tBlindBLS, due to its different definition of pk. We mark the differences with shadowed text.

$\underline{\mathsf{Setup}(1^\kappa)}$

$\mathbb{G}_1$, $\mathbb{G}_2, \mathbb{G}_T$ are groups of prime order m with generators g_1, g_2, g_T, respectively. $H_0 : \{0,1\}^* \to \mathbb{G}_1$, $H_1 : \{0,1\}^* \to \mathbb{G}_1$, and $H_2 : \{0,1\}^* \to \mathbb{G}_1$ are hash functions. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing. Pick $g_k, g_z, g_{\hat{z}} \leftarrow_\$ \mathbb{G}_1$.

$\underline{\mathsf{KeyGen}(n,t)}$

pick $a_0, a_1, \ldots, a_t, b_1, \ldots, b_t, \hat{b}_1, \ldots, \hat{b}_t \leftarrow_\$ \mathbb{Z}_m$
define $k := a_0$, polynomials $k(i) := \sum_{\iota=0}^{t} a_\iota i^\iota$, $z(i) := \sum_{\iota=1}^{t} b_\iota i^\iota$, $\hat{z}(i) := \sum_{\iota=1}^{t} \hat{b}_\iota i^\iota$
$\forall_{i\in[n]}\ k_i := k(i), z_i := z(i), \hat{z}_i := \hat{z}(i), \mathsf{pk}_i := (g_k)^{k_i} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$
$\mathsf{pk} := (g_k)^k$ $\mathsf{pk} := (g_2)^k$, $\pi_{\mathsf{KeyGen}} := \mathsf{NIZK}_{\mathsf{KeyGen}}.\mathsf{Prove}(\mathsf{pk}, k)$
return $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pk}, \mathsf{aux}) := (\{k_i, z_i, \hat{z}_i, \mathsf{pk}_i\}_{i\in[n]}, \mathsf{pk}, (n, t, \{\mathsf{pk}_i\}_{i\in[n]}, \pi_{\mathsf{KeyGen}}))$

$\underline{\mathsf{UEval}(x, \mathsf{pk}, \mathsf{aux} = (n, t, \{\mathsf{pk}_i\}_{i\in[n]}, \pi_{\mathsf{KeyGen}}))}$

pick $r \leftarrow_\$ \mathbb{Z}_m$
$p := H_0(x)^r$
return $(\mathsf{st}^{\mathsf{U}}, \mathsf{pm}^{\mathsf{U}}) := ((\mathsf{pk}, \mathsf{aux}, r, p), p)$

$\underline{\mathsf{SEval}(i, \mathsf{sk}_i = (k_i, z_i, \hat{z}_i, \mathsf{pk}_i), \mathsf{ctx}, \mathsf{pm}^{\mathsf{U}} = p)}$

$q_i := p^{k_i} \cdot H_1(\mathsf{ctx}, p)^{z_i} \cdot H_2(\mathsf{ctx}, p)^{\hat{z}_i}$
$\pi_i := \mathsf{NIZK}_{\mathsf{SEval}}.\mathsf{Prove}((p, H_1(\mathsf{ctx}, p), H_2(\mathsf{ctx}, p), \mathsf{pk}_i, q_i), (k_i, z_i, \hat{z}_i))$
return $\mathsf{pm}^i := (q_i, \mathsf{ctx}, \pi_i)$

$\underline{\mathsf{UAgg}(\mathsf{st}^{\mathsf{U}} = (\mathsf{pk}, n, t, \{\mathsf{pk}_i\}_{i\in[n]}, \pi_{\mathsf{KeyGen}}, r, p), \mathbf{S}, \{\mathsf{pm}^i = (q_i, \mathsf{ctx}_i, \pi_i)\}_{i\in\mathbf{S}})}$

if $\exists_{i,j\in\mathbf{S}}\ \mathsf{ctx}_i \neq \mathsf{ctx}_j$ **return** $\perp$
if $\mathsf{NIZK}_{\mathsf{KeyGen}}.\mathsf{Verify}(\mathsf{pk}, \pi_{\mathsf{KeyGen}}) \neq 1$ **return** $\perp$
if $\exists_{i\in\mathbf{S}}\ \mathsf{NIZK}_{\mathsf{SEval}}.\mathsf{Verify}((p, H_1(\mathsf{ctx}_i, p), H_2(\mathsf{ctx}_i, p), \mathsf{pk}_i, q_i), \pi_i) \neq 1$ **return** $\perp$
$\forall_{i\in\mathbf{S}}$ compute Lagrange coeff. λ_i
$y := (\prod_{i\in\mathbf{S}} q_i^{\lambda_i})^{1/r}$
if $\prod_{i\in\mathbf{S}} \mathsf{pk}_i^{\lambda_i} \neq \mathsf{pk}$ $e(y, g_2) \neq e(H_0(x), \mathsf{pk})$ **return** $\perp$; **else return** y

$\underline{\mathsf{Offline}(\{\mathsf{sk}_i = (k_i, \ldots)\}_{i\in[n]}, \mathsf{aux}, x)}$

$\forall_{i\in[t+1]}$ compute Lagrange coeff. λ_i
return $H_0(x)^{\sum_{i\in[t+1]} \lambda_i k_i}$

$\underline{\mathsf{Verify}(\mathsf{pk}, x, y)}$

if $e(y, g_2) = e(H_0(x), \mathsf{pk})$ **return** 1
else return 0

Fig. 3. The threshold oblivious pseudorandom function protocol 2B–HashTDH (dashed text) and the threshold blind BLS signature protocol 2B–tBlindBLS (shadowed text).

Preliminaries. We begin by classifying all the $\mathbb{G}_1$ elements received and sent by an adversary $\mathcal{A}$ in the one-more unpredictability game $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}$. At the game's start $\mathcal{A}$ sees $g_1, g_k, g_z, g_{\hat{z}}, \mathsf{pk}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n \in \mathbb{G}_1$. Throughout the game, $\mathcal{A}$ sends queries to random oracles H_0, H_1, H_2. Define that $\mathcal{A}$ queries H_0 at most q times with inputs $x_1, x_2, \ldots, x_q \in \{0,1\}^*$ and receives responses $h_1, h_2, \ldots, h_q \in \mathbb{G}_1$. Assume without loss of generality that $\mathcal{A}$ always queries H_1 and H_2 in tandem on the same inputs. Define that $\mathcal{A}$ queries H_1 and H_2 at most $\overline{q}$ times with inputs $(\mathsf{ctx}_1, p_1), (\mathsf{ctx}_2, p_2), \ldots (\mathsf{ctx}_{\overline{q}}, p_{\overline{q}}) \in \{0,1\}^* \times \mathbb{G}_1$ and receives responses $\overline{h}_1, \overline{h}_2, \ldots, \overline{h}_{\overline{q}} \in \mathbb{G}_1$ from H_1 and $\hat{\overline{h}}_1, \hat{\overline{h}}_2, \ldots, \hat{\overline{h}}_{\overline{q}} \in \mathbb{G}_1$ from H_2. Also, $\mathcal{A}$ sends queries to SEval. Assume without loss of generality that $\mathcal{A}$ queries H_1 and H_2 on every (ctx, p) sent to SEval. Define that query $\mathsf{SEval}(i, \mathsf{ctx}_{\overline{j}}, p_{\overline{j}})$ produces response $q_{i,\overline{j}} \in \mathbb{G}_1$ (note that SEval is deterministic). Finally, $\mathcal{A}$ outputs purported function evaluations $(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_L^*, y_L^*) \in \{0,1\}^* \times \mathbb{G}_1$.

Since we assume that $\mathcal{A}$ is algebraic, every $\mathbb{G}_1$ element sent by $\mathcal{A}$ must be accompanied by an algebraic representation in terms of $\mathcal{A}$'s previously seen $\mathbb{G}_1$ elements. Label the y_ℓ^* representations (using notation $[X]^+ \equiv \{0\} \cup [X]$):

$$y_\ell^* = \prod_{j \in [q]^+} (h_j)^{r_j^\ell} \cdot \prod_{J \in [\overline{q}]^+} (\overline{h}_J)^{s_J^\ell} \cdot (\hat{\overline{h}}_J)^{u_J^\ell} \cdot \prod_{i,J \in [n] \times [\overline{q}]^+} (q_{i,J})^{v_{i,J}^\ell}$$

Group elements $g_k, g_z, g_{\hat{z}}$ are not explicit in this representation. Looking ahead, all our reductions and simulations will generate these values in the same way as H_0, H_1, H_2 responses, respectively. Therefore, $(g_k, g_z, g_{\hat{z}})$ can be understood as $(h_0, \overline{h}_0, \hat{\overline{h}}_0)$ for $p_0 := h_0$. For the same reason, $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ can be understood as $(q_{1,0}, \ldots, q_{n,0})$. Since pk can be expressed in terms of $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, assume without loss of generality that $\mathcal{A}$ does not use directly pk in any representations. Finally, observe that g_1 is never used by our protocols. We therefore omit it here under the simplifying assumption that $g_1 = g_k$.

The H_1, H_2, and SEval inputs $p_{\overline{j}}$ are also $\mathbb{G}_1$ elements sent by $\mathcal{A}$ and so must be accompanied by algebraic representations. Because these values are indexed chronologically, the representation of $p_{\overline{j}}$ cannot include any $\overline{h}_J$, $\hat{\overline{h}}_J$, or $q_{i,J}$ for $J \geq \overline{j}$. Therefore, label them as:

$$p_{\overline{j}} = \prod_{j \in [q]^+} (h_j)^{\overline{r}_j^{\overline{j}}} \cdot \prod_{J=0}^{\overline{j}-1} (\overline{h}_J)^{\overline{s}_J^{\overline{j}}} \cdot (\hat{\overline{h}}_J)^{\overline{u}_J^{\overline{j}}} \cdot \prod_{i \in [n]} \prod_{J=0}^{\overline{j}-1} (q_{i,J})^{\overline{v}_{i,J}^{\overline{j}}}$$

For all $J \in [\overline{q}]^+$, define $\mathbf{S}(J) \subseteq [n]$ to be the set of servers i for which $\exists_{\ell \in [L]}\ v_{i,J}^\ell \neq 0$ or $\exists_{\overline{j} \in [\overline{q}]}\ \overline{v}_{i,J}^{\overline{j}} \neq 0$. (Except for $J = 0$) $\mathcal{A}$ must have queried SEval with (ctx_J, p_J) for every server in $\mathbf{S}(J)$. Assume without loss of generality that $\mathbf{S}(J)$ and $\mathsf{corrupt}$ are disjoint; for $i \in \mathsf{corrupt}$, $\mathcal{A}$ knows how to represent $q_{i,J}$ in terms of other elements. Similarly, assume without loss of generality that, for all J, $|\mathbf{S}(J)| \leq t + 1 - |\mathsf{corrupt}|$; $\mathcal{A}$ can represent any $q_{i,J}$ values beyond that threshold in terms of the others using interpolation.

Per the code of SEval, we know that $q_{i,J} = (p_J)^{k_i} \cdot (\overline{h}_J)^{z_i} \cdot (\hat{\overline{h}}_J)^{\hat{z}_i}$. We can rewrite the representation of y^*_ℓ accordingly:

$$y^*_\ell = \prod_{j \in [q]^+} (h_j)^{r^\ell_j} \cdot \prod_{J \in [\overline{q}]^+} (p_J)^{\sum_{i \in \mathbf{S}(J)} v^\ell_{i,J} k_i} \cdot (\overline{h}_J)^{s^\ell_J + \sum_{i \in \mathbf{S}(J)} v^\ell_{i,J} z_i} \cdot (\hat{\overline{h}}_J)^{u^\ell_J + \sum_{i \in \mathbf{S}(J)} v^\ell_{i,J} \hat{z}_i} \tag{1}$$

We can perform the same expansion on the $p_{\overline{j}}$ representations:

$$p_{\overline{j}} = \prod_{j \in [q]^+} (h_j)^{\overline{r}^{\overline{j}}_j} \cdot \prod_{J=0}^{\overline{j}-1} (p_J)^{\sum_{i \in \mathbf{S}(J)} \overline{v}^{\overline{j}}_{i,J} k_i} \cdot (\overline{h}_J)^{\overline{s}^{\overline{j}}_J + \sum_{i \in \mathbf{S}(J)} \overline{v}^{\overline{j}}_{i,J} z_i} \cdot (\hat{\overline{h}}_J)^{\overline{u}^{\overline{j}}_J + \sum_{i \in \mathbf{S}(J)} \overline{v}^{\overline{j}}_{i,J} \hat{z}_i} \tag{2}$$

Without loss of generality, assume $\mathcal{A}$ queries H_0 on every x^*_ℓ and define $j(\ell) \in [q]$ to be the index such that $x^*_\ell = x_{j(\ell)}$.

The Wrapper Algorithm. To bound the probability that $\mathcal{A}$ wins the one-more unpredictability game, we define a wrapper algorithm $\mathcal{B}(\mathcal{A})$ that runs $\mathcal{A}$ as a subroutine, simulating its input and oracle responses (step I). When $\mathcal{A}$ finishes, $\mathcal{B}$ verifies the win conditions and aborts if they fail (step II). $\mathcal{B}$ then performs several additional steps that act as a blueprint for subsequent reductions. In step III, $\mathcal{B}$ iteratively "unrolls" any recursive parts of the algebraic representations provided by $\mathcal{A}$ for its PRF evaluation outputs. Along the way, this step checks that every SEval response used in those representations appears as part of a set of $t + 1 - |\mathsf{corrupt}|$ properly interpolated responses from different servers. If this check fails, $\mathcal{A}$ must have found an unexpected way of making some blinding factor cancel out. In this case, $\mathcal{B}$ aborts, and an OMDL reduction is implied because a non-trivial relationship can be extracted between the $\{z_i, \hat{z}_i\}$ values. If step III checks pass, then $\mathcal{B}$ in step IV checks that the PRF evaluation outputs provided by $\mathcal{A}$ are not represented in other unexpected way, i.e. that the secret key does not appear directly in the representation and that recursive query responses are not meaningfully used. If this check fails, $\mathcal{B}$ aborts, and an SDL reduction is implied because a non-trivial equation can be extracted involving the secret element α, a component of the "rigged" effective key k'. Finally, in step V, we use a matrix rank argument to show that, if all previous checks pass, then it is impossible for $\mathcal{A}$ to have produced more than the allowed number of evaluation outputs, which is a contradiction with step II. Since our reductions show that step III and IV aborts occur with negligible probability, it follows that the step II abort must occur with high probability, i.e. that $\mathcal{A}$ does not win the game.

Our SDL reduction requires careful rigging of the public keys and random oracle responses, but our OMDL reduction does not. Also, we must ultimately show that $\mathcal{A}$ cannot win the game in the "real world", i.e. without rigging. Below we first present the reduction blueprint $\mathcal{B}$ in the "rigged" form. Then, we use two hybrids $\mathcal{B}'$ and $\mathcal{B}''$ to argue that the removal of the rigging behavior is not detectable by $\mathcal{A}$, under the DDH assumption. Our SDL reduction is based on $\mathcal{B}$, and our OMDL reduction is based on $\mathcal{B}''$.

$\underline{\mathcal{B} \text{ step I: run } \mathcal{A}}$

– First generate the initial input to $\mathcal{A}$:
  - Pick "rigging values" $\alpha, z, \hat{z} \leftarrow_\$ \mathbb{Z}_m$.
  - Pick $\tau_k, \delta_z \leftarrow_\$ \mathbb{Z}_m$ and define public parameters $g_k := (g_1)^{\tau_k}$, $g_z := (g_k)^{(\delta_z)\cdot\alpha/z}$, $g_{\hat{z}} := (g_k)^{(1-\delta_z)\cdot\alpha/\hat{z}}$.
  - Receive (n, t) from $\mathcal{A}$.
  - Pick $a_0(=k), a_1, \ldots a_t, b_1, \ldots b_t, \hat{b}_1, \ldots \hat{b}_t \leftarrow_\$ \mathbb{Z}_m$.
  - Define $k(i) := \sum_{\iota=0}^{t} a_\iota i^\iota$, $z(i) := \sum_{\iota=1}^{t} b_\iota i^\iota$, $\hat{z}(i) := \sum_{\iota=1}^{t} \hat{b}_\iota i^\iota$, for all i.
  - Compute honest keys $\forall_{i\in[n]}\ k_i := k(i), z_i := z(i), \hat{z}_i := \hat{z}(i)$.
  - Define the "effective" PRF key $k' := k + \alpha$ and corresponding public key $\mathsf{pk}' := (g_k)^{k+\alpha}$ (or $\mathsf{pk}' := (g_2)^{k+\alpha}$ for 2B–tBlindBLS).
  - Compute rigged keys $\forall_{i\in[n]}\ z'_i := z_i + z, \hat{z}'_i := \hat{z}_i + \hat{z}, \mathsf{pk}'_i := (g_k)^{k_i+\alpha} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$. Set $\{z'_i\}_{i\in[n]}$ and $\{\hat{z}'_i\}_{i\in[n]}$ as sharings of $z' := z$ and $\hat{z}' := \hat{z}$.
  - Simulate proof π_{KeyGen} (see Appendix B for details).
  - Send $(\mathsf{pk}', (n, t, \{\mathsf{pk}'_i\}_{i\in[n]}, \pi_{\mathsf{KeyGen}}))$ to $\mathcal{A}$.
– Respond to SEval and Corrupt game oracle queries earnestly using the rigged keys $\{\mathsf{sk}'_i = (k_i, z'_i, \hat{z}'_i, \mathsf{pk}'_i)\}_{i\in[n]}$, except simulate the $\mathsf{NIZK}_{\mathsf{SEval}}$ proofs in SEval responses (see Appendix B for details). Maintain evalset as in $\mathbf{Game}^{\mathsf{om-unpredictable}}$, indexed by (ctx, p).
– Respond to H_0 queries using random $\mathbb{G}_1$ elements with known discrete logarithms (i.e. "trapdoors"), i.e. set $h_j = H_0(x_j) := (g_1)^{\tau_j}$ for $\tau_j \leftarrow_\$ \mathbb{Z}_m$.
– Respond to H_1 and H_2 queries in a manner that is correlated and dependent on the input element. Specifically, pick $\delta_{\overline{j}} \leftarrow_\$ \mathbb{Z}_m$ and set $\overline{h}_{\overline{j}} = H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) := (p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha/z}$ and $\hat{\overline{h}}_{\overline{j}} = H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) := (p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha/\hat{z}}$.

(One can verify at this point that honest evaluation by $\mathcal{A}$ will behave as expected, i.e. $\mathcal{A}$ will correctly compute PRF outputs for the effective key k'. We will later show by DDH reduction that $\mathcal{A}$'s view is indeed indistinguishable from the real world in spite of $\mathcal{B}$'s rigging of keys and random oracle responses.)

$\mathcal{B}$ step II: verify $\mathcal{A}$'s victory

Receive claimed function evaluations $\{(x^*_\ell, y^*_\ell)\}_{\ell\in[L]}$ from $\mathcal{A}$, and verify that (i) $|\mathsf{corrupt}| < t + 1$, (ii) $L > |\{\overline{j} : |\mathsf{evalset}[\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}] \cup \mathsf{corrupt}| \geq t + 1\}|$, and (iii) $\forall_{\ell\in[L]}\ y^*_\ell = H_0(x^*_\ell)^{k'}$. If any condition fails output (II.AdvLoses) and abort.

$\mathcal{B}$ step III: verify proper interpolation

For all $J \in [\overline{q}]^+$ such that $|\mathbf{S}(J)| = t + 1 - |\mathsf{corrupt}|$, define $\{\lambda^J_i\}$ to be the Lagrange interpolation coefficients for interpolation set $\mathbf{S}(J) \cup \mathsf{corrupt}$. Also, as a shorthand, define $\kappa_J := k' - \sum_{i\in\mathsf{corrupt}} \lambda^J_i k_i$. For all other J (i.e. those for which $|\mathbf{S}(J)| < t + 1 - |\mathsf{corrupt}|$), define $\kappa_J := 0$.

For each $\ell \in [L]$, we use the following loop to eliminate all $p_{\overline{j}}$ elements from the representation of y^*_ℓ. The loop begins with $\overline{j} := \overline{q}$ and ends with $\overline{j} = 0$:

1. At the start of each iteration, we have the following representation for y^*_ℓ, where $\{\mu^\ell_J\}_{J>\overline{j}}$ are defined by the previous iterations of the loop. Note that

in the first iteration, i.e. $\overline{j} = \overline{q}$, this representation matches equation (1).

$$
\begin{aligned}
y^*_\ell = & \prod_{j \in [q]^+} (h_j)^{r^\ell_j + \sum_{J=\overline{j}+1}^{\overline{q}} \mu^\ell_J \kappa_J \overline{r}^J_j} \\
& \cdot \prod_{J=0}^{\overline{j}} (p_J)^{\sum_{i \in \mathbf{S}(J)} (v^\ell_{i,J} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,J}) \cdot k_i} \\
& \cdot \prod_{J=0}^{\overline{j}} (\overline{h}_J)^{(s^\ell_J + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{s}^G_J) + \sum_{i \in \mathbf{S}(J)} (v^\ell_{i,J} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,J}) \cdot z'_i} \\
& \cdot \prod_{J=0}^{\overline{j}} \cdot (\hat{\overline{h}}_J)^{(u^\ell_J + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{u}^G_J) + \sum_{i \in \mathbf{S}(J)} (v^\ell_{i,J} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,J}) \cdot \hat{z}'_i}
\end{aligned} \tag{3}
$$

2. (Case 1) Set $\mu^\ell_{\overline{j}} := 0$ if the following three conditions are all satisfied:

$$(s^\ell_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{s}^G_{\overline{j}}) = 0$$

$$(u^\ell_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{u}^G_{\overline{j}}) = 0$$

$$(v^\ell_{i,\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,\overline{j}}) = 0 \quad \text{for all } i \in \mathbf{S}(\overline{j})$$

(This case happens when $\overline{h}_{\overline{j}}$, $\hat{\overline{h}}_{\overline{j}}$ and $\{q_{i,\overline{j}}\}_{i \in \mathbf{S}(\overline{j})}$ bases do not meaningfully participate in the representation of y^*_ℓ.)

3. (Case 2) Otherwise do the following steps:
   - Verify that $|\mathbf{S}(\overline{j})| = t + 1 - |\mathsf{corrupt}|$, otherwise output (III.BadInterp) and abort. *(This abort indicates that $\mathcal{A}$ did not make enough* SEval *queries involving $p_{\overline{j}}$ to perform proper interpolation.)*
   - Set $\mu^\ell_{\overline{j}}$ to a value that satisfies all the equations below:

$$(s^\ell_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{s}^G_{\overline{j}}) = \mu^\ell_{\overline{j}} \cdot \sum_{i \in \mathsf{corrupt}} \lambda^{\overline{j}}_i z'_i$$

$$(u^\ell_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{u}^G_{\overline{j}}) = \mu^\ell_{\overline{j}} \cdot \sum_{i \in \mathsf{corrupt}} \lambda^{\overline{j}}_i \hat{z}'_i$$

$$(v^\ell_{i,\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,\overline{j}}) = \mu^\ell_{\overline{j}} \cdot \lambda^{\overline{j}}_i \quad \text{for all } i \in \mathbf{S}(\overline{j})$$

   If there is no such value $\mu^\ell_{\overline{j}}$ then output (III.BadInterp) and abort. *(This abort indicates that $\mathcal{A}$, though having queried enough $q_{i,\overline{j}}$ values,*

did not perform proper interpolation with them, e.g. the blinding factors $\overline{h}_{\overline{j}}, \hat{\overline{h}}_{\overline{j}}$ do not cancel out in the algebraic expression for y^*_ℓ.)

- Now it is verified that the exponents of the $p_{\overline{j}}$, $\overline{h}_{\overline{j}}$, and $\hat{\overline{h}}_{\overline{j}}$ factors in equation (3) are just some constant multiple $\mu^\ell_{\overline{j}}$ of interpolation coefficients. Therefore, these factors simplify to:

$$(p_{\overline{j}})^{\sum_{i\in\mathbf{S}(\overline{j})}\mu^\ell_{\overline{j}}\lambda^{\overline{j}}_i k_i}\cdot(\overline{h}_{\overline{j}})^{\mu^\ell_{\overline{j}}\sum_{i\in\mathsf{corrupt}}\lambda^{\overline{j}}_i z'_i+\sum_{i\in\mathbf{S}(\overline{j})}\mu^\ell_{\overline{j}}\lambda^{\overline{j}}_i z'_i}\cdot(\hat{\overline{h}}_{\overline{j}})^{\mu^\ell_{\overline{j}}\sum_{i\in\mathsf{corrupt}}\lambda^{\overline{j}}_i \hat{z}'_i+\sum_{i\in\mathbf{S}(\overline{j})}\mu^\ell_{\overline{j}}\lambda^{\overline{j}}_i \hat{z}'_i}$$

4. In either case, the $p_{\overline{j}}$, $\overline{h}_{\overline{j}}$, and $\hat{\overline{h}}_{\overline{j}}$ factors simplify to:

$$\left((p_{\overline{j}})^{k-\sum_{i\in\mathsf{corrupt}}\lambda^{\overline{j}}_i k_i}\cdot(\overline{h}_{\overline{j}})^{z'}\cdot(\hat{\overline{h}}_{\overline{j}})^{\hat{z}'}\right)^{\mu^\ell_{\overline{j}}} \tag{4}$$

   Due to the rigged programming of H_1 and H_2 in step I, we know that $\overline{h}_{\overline{j}} = (p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha/z}$ and $\hat{\overline{h}}_{\overline{j}} = (p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha/\hat{z}}$ where $z = z'$ and $\hat{z} = \hat{z}'$. Therefore, $(\overline{h}_{\overline{j}})^{z'}\cdot(\hat{\overline{h}}_{\overline{j}})^{\hat{z}'} = (p_{\overline{j}})^{\alpha}$ and we can simplify expression (4) further:

$$\left((p_{\overline{j}})^{k-\sum_{i\in\mathsf{corrupt}}\lambda^{\overline{j}}_i k_i}\cdot(\overline{h}_{\overline{j}})^{z'}\cdot(\hat{\overline{h}}_{\overline{j}})^{\hat{z}'}\right)^{\mu^\ell_{\overline{j}}} = \left((p_{\overline{j}})^{\kappa_{\overline{j}}}\right)^{\mu^\ell_{\overline{j}}} \tag{5}$$

   Expand $p_{\overline{j}}$ using its provided representation, equation (2), and then substitute these factors back into equation (3). The result is precisely the representation of y^*_ℓ needed for the next loop iteration.
5. Decrement $\overline{j}$ and iterate.

At the end of this loop, if no abort occurs, we extract values $\{\mu^\ell_J\}_{J\in[\overline{q}]^+}$ and obtain a greatly simplified representation of y^*_ℓ, for all $\ell\in[L]$:

$$y^*_\ell = \prod_{j\in[q]^+}(h_j)^{r^\ell_j+\sum_{J\in[\overline{q}]^+}\mu^\ell_J\kappa_J\overline{r}^J_j}$$

$\mathcal{B}$ step IV: verify that $\mathcal{A}$ does not know the secret key k'

Rather than compute the μ^ℓ_J values numerically, we can represent each one as a polynomial function of α while treating α as a variable of unknown value. Such a representation can be constructed during the loop in step III. There, μ^ℓ_J is defined as the value satisfying three equations that depend upon the values $\{\kappa_G\}$ and $\{\mu^\ell_G\}$ for $G > J$. The $\{\kappa_G\}$ values are linear functions of α (in the simplest case where no servers are corrupted, all $\kappa_G = k+\alpha$). The $\{\mu^\ell_G\}$ values have their own representations as polynomial functions of α, found by previous loop iterations. Each iteration can create a polynomial of degree at most 1 greater than those already found, so these polynomials are all of degree at most $\overline{q}$.

Define coefficients $\{\beta^\ell_{J,e}\}$ found via the procedure described above:

$$\forall_{\ell,J\in[L]\times[\overline{q}]^+}\quad \mu^\ell_J = \sum_{e=0}^{\overline{q}}\beta^\ell_{J,e}\alpha^e \tag{6}$$

Next, verify that:

$$\forall_{\ell,j\in[L]\times[q]} \quad \gamma_j^\ell := \sum_{J\in[\overline{q}]^+} \beta_{J,0}^\ell \overline{r}_j^J = \begin{cases} 1 & j = j(\ell) \\ 0 & j \neq j(\ell) \end{cases} \tag{7}$$

If not, output (IV.KeyKnown) and abort. This abort indicates that $\mathcal{A}$ has found an unexpected representation for the secret key k', which is using either the scalar r_j^ℓ (implying direct knowledge of k') or the higher-degree coefficients $\{\beta_{J,e}^\ell\}_{e\geq 1}$ (implying meaningful participation of recursive queries).

Note that the $J = 0$ terms can be omitted from the summation in equation (7) because $\overline{r}_j^0 = 0$ for all $j \neq 0$.

$\mathcal{B}$ step V: expose contradiction

Express the matrix of all $\{\gamma_j^\ell\}$ values in terms of the $\{\beta_{J,0}^\ell\}$ and $\{\overline{r}_j^J\}$ values:

$$\begin{bmatrix} \gamma_1^1 & \gamma_2^1 & \cdots & \gamma_q^1 \\ \gamma_1^2 & \gamma_2^2 & & \gamma_q^2 \\ \vdots & & \ddots & \vdots \\ \gamma_1^L & \gamma_2^L & \cdots & \gamma_q^L \end{bmatrix} = \begin{bmatrix} \beta_{1,0}^1 & \beta_{2,0}^1 & \cdots & \beta_{\overline{q},0}^1 \\ \beta_{1,0}^2 & \beta_{2,0}^2 & & \beta_{\overline{q},0}^2 \\ \vdots & & \ddots & \vdots \\ \beta_{1,0}^L & \beta_{2,0}^L & \cdots & \beta_{\overline{q},0}^L \end{bmatrix} \begin{bmatrix} \overline{r}_1^1 & \overline{r}_2^1 & \cdots & \overline{r}_q^1 \\ \overline{r}_1^2 & \overline{r}_2^2 & \cdots & \overline{r}_q^2 \\ \vdots & & \ddots & \vdots \\ \overline{r}_1^{\overline{q}} & \overline{r}_2^{\overline{q}} & \cdots & \overline{r}_q^{\overline{q}} \end{bmatrix}$$
$$\Gamma = BR$$

Matrix Γ is $L\times q$. As verified in step IV, each row of Γ consists of all 0s except for a single 1; each column contains at most a single 1. Therefore, $\mathsf{rank}(\Gamma) = L$.

Matrix B is $L \times \overline{q}$, and matrix R is $\overline{q} \times q$. Define $\overline{q}^* \leq \overline{q}$ to be the number of columns in B that are not entirely 0. $\mathsf{rank}(B) \leq \overline{q}^*$, and therefore $\mathsf{rank}(BR) \leq \overline{q}^*$. Since $\Gamma = BR$, it follows that $L \leq \overline{q}^*$.

The existence of a nonzero $\beta_{J,0}^\ell$ implies that (Case 2) occured in the $\overline{j} = J$ iteration of step III. As verified there, then, $|\mathbf{S}(J)| = t + 1 - |\mathsf{corrupt}|$, which implies that $\mathcal{A}$ must have queried SEval with (ctx_J, p_J) for at least $t+1-|\mathsf{corrupt}|$ uncorrupted servers, i.e. $|\mathsf{evalset}[\mathsf{ctx}_J, p_J] \cup \mathsf{corrupt}| \geq t+1$. Therefore, as verified in step II, $L > \overline{q}^*$. This is a contradiction!

Reductions. It remains to show that the probability of (II.AdvLoses) is equal (up to a negligible difference) to the probability that $\mathcal{A}$ loses the real one-more unpredictability game and that events (III.BadInterp) and (IV.KeyKnown) both occur with negligible probability. We approach these events in the reverse of the order they appear in $\mathcal{B}$.

All our reductions have some steps in common, since they are all based on the same "reduction blueprint" $\mathcal{B}$. First, they all run $\mathcal{A}$, whose runtime we denote $T_\mathcal{A}$. Second, they all run the loop in $\mathcal{B}$ step III; since $L \leq q$, this loop is $\mathcal{O}(q\overline{q})$. The $\overline{q}$-co-SDL reduction in the proof of Lemma 2 additionally maintains polynomial representations of the μ_J^ℓ values as explained in $\mathcal{B}$ step IV; this increases the runtime of the step III loop to $\mathcal{O}(q\overline{q}^2)$. That reduction also runs a $\mathcal{O}(\overline{q}^{1.815} \log m)$ runtime polynomial root-finding algorithm as a subroutine. Finally, two of the reductions employ as a substep the reduction from DL to

N-RDL (Lemma 1), with $N \in \mathcal{O}(q + \overline{q})$. Therefore, all reductions run in time $\mathcal{O}(T_{\mathcal{A}} + q\overline{q}^2 + q \log m + \overline{q}^{1.815} \log m)$. To avoid redundancy, we omit repeated mention of runtimes from our reductions' explanations until we combine our results in Theorem 1.

Lemma 2. *If $\overline{q}$-co-SDL is $(\epsilon_{SDL}, T_{SDL})$-hard in $(\mathbb{G}_1, \mathbb{G}_2)$ and $\mathcal{A}$ makes at most q queries to H_0 and at most $\overline{q}$ queries to H_1 and H_2, then $\Pr[\mathcal{B}(\mathcal{A}) \to (\text{IV.KeyKnown})] \leq 2 \cdot \epsilon_{SDL}$.*

Proof. At the time of a (IV.KeyKnown) event, we have found some $\ell, j \in [L] \times [q]$ that violates equation (7). Define F as the event that the following equation holds:

$$r_j^\ell + \sum_{J \in [\overline{q}]^+} \mu_J^\ell \kappa_J \overline{r}_j^J = \begin{cases} k + \alpha & j = j(\ell) \\ 0 & j \neq j(\ell) \end{cases} \tag{8}$$

Intuitively, event F states that no H_0 responses other than $H_0(x_\ell^*)$ are meaningfully used in $\mathcal{A}$'s algebraic representation of $y_\ell^* = H_0(x_\ell^*)^{k+\alpha}$.

Given a $\overline{q}$-co-SDL input $\mathsf{challenge} = (g_1, (g_1)^\alpha, (g_1)^{\alpha^2}, \ldots, (g_1)^{\alpha^{\overline{q}}}, g_2, (g_2)^\alpha)$, we use $\mathcal{A}$ to solve for α. We follow one of two reduction strategies at random. Strategy #1 succeeds if event F occurs, and strategy #2 succeeds if it doesn't.

Strategy #1

Run $\mathcal{B}(\mathcal{A})$, but using the group elements of $\mathsf{challenge}$ to compute everything involving α. To confirm that this is possible, observe that α is never directly needed by $\mathcal{B}$:

- In step I, the τ_k trapdoor embedded in g_k allows us to compute the public parameters g_z and $g_{\hat{z}}$ and the public keys $\mathsf{pk}'_i = (g_k)^{k_i+\alpha} \cdot (g_z)^{z'_i} \cdot (g_{\hat{z}})^{\hat{z}'_i}$ and $\mathsf{pk}' = (g_k)^{k+\alpha}$. (For 2B–tBlindBLS, $\mathsf{pk}' = (g_2)^{k+\alpha}$ can be computed easily.)
- In step I, every $p_{\overline{j}}$ input to H_1 and H_2 is accompanied by its algebraic representation. In these representations, only bases involving α have discrete logarithms that are unknown to us. Therefore, the elements of $\mathsf{challenge}$ are sufficient to compute the correct responses. (In the extreme, $\mathcal{A}$ might recursively call the random oracles on their own outputs $\overline{q}$ times, in which case all elements of $\mathsf{challenge}$ are needed.)
- In step II, the $\tau_{j(\ell)}$ trapdoor embedded in $h_{j(\ell)}$ allows us to compute the correct function output $(h_{j(\ell)})^{k+\alpha}$ for each x_ℓ^*.
- In step III, we cannot compute the $\{\mu_{\overline{j}}^\ell\}$ values numerically. However, it is sufficient to represent them as polynomial functions of α, as explained at the start of step IV. The condition for (Case 1) can be checked in the exponent using the elements of $\mathsf{challenge}$. The check for (III.BadInterp) can be skipped, since this reduction only succeeds in the case of (IV.KeyKnown), which requires that the earlier (III.BadInterp) does not occur. Instead, we can simply assume in (Case 2) that a satisfactory $\mu_{\overline{j}}^\ell$ exists and use any one of the three equations to find its representation as a polynomial function of α, see equation (6).

After applying the definition of κ_J and the aforementioned polynomial representation of μ_J^ℓ, equation (8) is a polynomial function of α, with degree at most $\overline{q}+1$. The degree 1 coefficient of this polynomial is:

$$\left(\sum_{J\in[\overline{q}]^+} \beta_{J,0}^\ell \overline{r}_j^J\right) - \begin{cases} 1 & j = j(\ell) \\ 0 & j \neq j(\ell) \end{cases}$$

The violation of equation (7) guarantees that this coefficient is nonzero, and therefore that the polynomial as a whole is nonzero. The polynomial has at most $\overline{q}+1$ roots, one of which must be α. We can use a polynomial root-finding algorithm, and then test all possibilities until finding the correct α and thereby solving $\overline{q}$-co-SDL.

Strategy #2

There is an obvious reduction from $\overline{q}$-co-SDL to DL, and (per Lemma 1) there is a reduction from DL to RDL. Therefore, we present strategy #2 as an RDL reduction on input $((g_1)^{\chi_k}, (g_1)^{\chi_1}, \ldots, (g_1)^{\chi_q})$.

Run $\mathcal{B}(\mathcal{A})$, but use $g_k := (g_1)^{\chi_k}$ and for all $j \in [q]$ use $h_j := (g_1)^{\chi_j}$ to respond to random oracle query $H_0(x_j)$. To confirm that this is possible, observe that the $\tau_k, \{\tau_j\}_{j\in[q]}$ trapdoors are never used by $\mathcal{B}$.

Since (II.AdvLoses) did not occur, y_ℓ^* must correctly equal $(h_{j(\ell)})^{k+\alpha}$:

$$(h_{j(\ell)})^{k+\alpha} = \prod_{j\in[q]^+} (h_j)^{r_j^\ell + \sum_{J\in[\overline{q}]^+} \mu_J^\ell \kappa_J \overline{r}_j^J}$$

If F does not occur, the exponent of h_j is nonzero, and we have an RDL solution.

The probability of correctly guessing whether or not F will occur is 1/2. The asymptotically fastest algorithms for polynomial root-finding over a finite field are probabilistic and can find the roots in time $\mathcal{O}(\overline{q}^{1.815}\log m)$ with failure probability $\leq 1/2$ [64, 85]. Thus, our reduction wins $\overline{q}$-co-SDL with probability $\geq 1/2 \cdot 1/2 \cdot \Pr[\mathcal{B}(\mathcal{A}) \to (\text{IV.KeyKnown})]$. Our hardness assumption therefore implies that $\Pr[\mathcal{B}(\mathcal{A}) \to (\text{IV.KeyKnown})] \leq 4 \cdot \epsilon_{SDL}$. □

To help us proceed, we define a hybrid $\mathcal{B}'$ of the wrapper algorithm. $\mathcal{B}'$ only runs steps I through III and does *not* perform any key rigging in step I. In other words, the $\{z_i'\}$ and $\{\hat{z}_i'\}$ keys really are sharings of zero in $\mathcal{B}'$. In particular, $\mathcal{B}'$ sets all $z_i' := z_i$ and $\hat{z}_i' := \hat{z}_i$; consequently, $z' := 0$ and $\hat{z}' := 0$. Also, $k' := k$, $\mathsf{pk}' := (g_k)^k$ (for 2B–HashTDH) or $\mathsf{pk}' := (g_2)^k$ (for 2B–tBlindBLS), and all $\mathsf{pk}_i' := (g_k)^{k_i} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$.

Lemma 3. *Define event* (EarlyAbort) *to be the union of* (II.AdvLoses) *and* (III.BadInterp)*. This event is equally probable for $\mathcal{B}(\mathcal{A})$ and $\mathcal{B}'(\mathcal{A})$, i.e.* $\Pr[\mathcal{B}(\mathcal{A}) \to (\text{EarlyAbort})] = \Pr[\mathcal{B}'(\mathcal{A}) \to (\text{EarlyAbort})]$.

Proof. If $\mathcal{A}$ corrupts $t+1$ or more servers, then (EarlyAbort) occurs with probability 1 in both $\mathcal{B}$ and $\mathcal{B}'$. Therefore, it suffices to show that the view of $\mathcal{A}$ is identical in $\mathcal{B}$ and $\mathcal{B}'$ as long as $\mathcal{A}$ does not corrupt $t+1$ servers.

In both $\mathcal{B}$ and $\mathcal{B}'$, $k(i)$ is a uniformly random t-degree polynomial. In $\mathcal{B}$, α is a uniformly random field element. For any $k(i)$ and α that may be chosen in $\mathcal{B}$, an identical view can produced in $\mathcal{B}'$ if polynomial $k'(i) = k(i) + \alpha$ is chosen:

- In $\mathcal{B}$ we set $\mathsf{pk} := (g_k)^{k(0)+\alpha}$ while in $\mathcal{B}'$ we set $\mathsf{pk} := (g_k)^{k'(0)}$. (For 2B–tBlindBLS we replace g_k with g_2.) Clearly these are equal.
- In $\mathcal{B}$, $\mathsf{pk}_i := (g_k)^{k(i)+\alpha} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$. In $\mathcal{B}'$, $\mathsf{pk}_i := (g_k)^{k'(i)} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$. Clearly these are equal.
- In $\mathcal{B}$, SEval response $q_{i,\overline{j}} = (p_{\overline{j}})^{k(i)} \cdot H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{z_i+z} \cdot H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{\hat{z}_i+\hat{z}}$. In $\mathcal{B}'$, $q_{i,\overline{j}} = (p_{\overline{j}})^{k'(i)} \cdot H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{z_i} \cdot H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{\hat{z}_i}$. These are equal due to the rigged programming of H_1 and H_2:

$$\begin{aligned}
& (p_{\overline{j}})^{k(i)} \cdot H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{z_i+z} \cdot H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{\hat{z}_i+\hat{z}} \\
= \; & (p_{\overline{j}})^{k(i)} \cdot ((p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha/z})^{z_i+z} \cdot ((p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha/\hat{z}})^{\hat{z}_i+\hat{z}} \\
= \; & (p_{\overline{j}})^{k(i)} \cdot (p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha} \cdot (p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha} \cdot ((p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha/z})^{z_i} \cdot ((p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha/\hat{z}})^{\hat{z}_i} \\
= \; & (p_{\overline{j}})^{k(i)+\alpha} \cdot ((p_{\overline{j}})^{(\delta_{\overline{j}})\cdot\alpha/z})^{z_i} \cdot ((p_{\overline{j}})^{(1-\delta_{\overline{j}})\cdot\alpha/\hat{z}})^{\hat{z}_i} \\
= \; & (p_{\overline{j}})^{k(i)+\alpha} \cdot H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{z_i} \cdot H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})^{\hat{z}_i}
\end{aligned}$$

- In $\mathcal{B}$, Corrupt response $\mathsf{sk}_i = (k(i), z_i + z, \hat{z}_i + \hat{z})$. In $\mathcal{B}'$, $\mathsf{sk}_i = (k'(i), z_i, \hat{z}_i)$. Since at most t servers are corrupted, the perfect security of Shamir secret sharing ensures that these key shares are uniformly distributed. In particular, $\mathcal{A}$ does not see that z'_i and $\hat{z}'_i$ are not really sharings of zero in $\mathcal{B}$. It remains only to show that sk_i agrees with the previously revealed public key pk_i and SEval responses $\{q_{i,\overline{j}}\}_{\overline{j}\in[\overline{q}]}$. The latter is immediate, because the revealed secret keys are the same as those used in SEval. The former holds due to the rigged choice of g_z and $g_{\hat{z}}$. We omit step-by-step analysis of this fact because it is essentially the same as the analysis given for $q_{i,\overline{j}}$ above.
- The only remaining element of $\mathcal{A}$'s view is the NIZKs sent during KeyGen and SEval. In both $\mathcal{B}$ and $\mathcal{B}'$, all NIZKs are simulated, and the only inputs to the simulators are the values that are shown to be equal above. Therefore, the NIZKs are also equal across $\mathcal{B}$ and $\mathcal{B}'$.
- Also, steps II and III of $\mathcal{B}$ and $\mathcal{B}'$ only use these same values that are shown to be equal above.

□

We move now to a second hybrid $\mathcal{B}''$ that additionally does not rig the choice of g_z and $g_{\hat{z}}$ and the programming of the random oracles in step I. In other words, g_z, $g_{\hat{z}}$, and the outputs of H_1 and H_2 really are all random (and uncorrelated) $\mathbb{G}_1$ elements in $\mathcal{B}''$. In particular, $g_z := (g_1)^{\tau_z}$ and $g_{\hat{z}} := (g_1)^{\tau_{\hat{z}}}$ where $\tau_z, \tau_{\hat{z}} \leftarrow_\$ \mathbb{Z}_m$; for all $\overline{j} \in [\overline{q}]$, $H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) := (g_1)^{\overline{\tau}_{\overline{j}}}$ and $H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) := (g_1)^{\hat{\overline{\tau}}_{\overline{j}}}$ where $\overline{\tau}_{\overline{j}}, \hat{\overline{\tau}}_{\overline{j}} \leftarrow_\$ \mathbb{Z}_m$. In $\mathcal{B}''$, the view of $\mathcal{A}$ is identical to the real world one-more unpredictability game except for the use of simulated NIZKs.

Lemma 4. *If DDH is $(\epsilon_{DDH}, T_{DDH})$-hard in $\mathbb{G}_1$ then* $\Pr[\mathcal{B}'(\mathcal{A}) \to (\textsc{EarlyAbort})] \leq \Pr[\mathcal{B}''(\mathcal{A}) \to (\textsc{EarlyAbort})] + \epsilon_{DDH}$.

Proof. Given a DDH input $\mathsf{challenge} = (d_1, d'_1, d_2, d'_2)$, we can use $\mathcal{A}$ to distinguish whether or not it is a Diffie-Hellman tuple. Run $\mathcal{B}'(\mathcal{A})$, but using the group elements of $\mathsf{challenge}$ to program H_1 and H_2. In particular:

$$H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) = \overline{h}_{\overline{j}} := \left((d_1)^{\delta_{1,\overline{j}}} \cdot (d_2)^{\delta_{2,\overline{j}}} \right)^{\alpha}$$

$$H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}}) = \hat{\overline{h}}_{\overline{j}} := \left(\frac{p_{\overline{j}}}{(d'_1)^{\delta_{1,\overline{j}}} \cdot (d'_2)^{\delta_{2,\overline{j}}}} \right)^{\alpha/\hat{z}}$$

where $\delta_{1,\overline{j}}, \delta_{2,\overline{j}} \leftarrow_{\$} \mathbb{Z}_m$ for all $\overline{j} \in [\overline{q}]$. Set g_z and $g_{\hat{z}}$ in the same way for random $\delta_{1,z}, \delta_{2,z} \leftarrow_{\$} \mathbb{Z}_m$. Notice that we do not explicitly choose z. If $\mathsf{challenge}$ is a Diffie-Hellman tuple, then z is effectively the secret exponent hidden in that tuple (called b in Definition 7). We are able to generate many pairs of correlated group elements from $\mathsf{challenge}$ by using the well-known "DDH randomized self-reduction" technique [81, 75].

In the case that $\mathsf{challenge}$ is a Diffie-Hellman tuple, $(d_1)^{\delta_{1,\overline{j}}} \cdot (d_2)^{\delta_{2,\overline{j}}}$ is a uniformly random group element and $(d'_1)^{\delta_{1,\overline{j}}} \cdot (d'_2)^{\delta_{2,\overline{j}}} = ((d_1)^{\delta_{1,\overline{j}}} \cdot (d_2)^{\delta_{2,\overline{j}}})^z$. Thus, the execution is identical to $\mathcal{B}'(\mathcal{A})$.

In the case that $\mathsf{challenge}$ is a random tuple, $\overline{h}_{\overline{j}}$ and $\hat{\overline{h}}_{\overline{j}}$ are independently and uniformly random group elements. Thus, the execution is identical to $\mathcal{B}''(\mathcal{A})$. Note that step III of $\mathcal{B}$ makes use of the correlated programming of H_1 and H_2 in equation (5). However, that equation still holds in $\mathcal{B}''$ for a different reason: $z' = \hat{z}' = 0$ and $k = k'$.

If event ($\textsc{EarlyAbort}$) occurs, then our distinguisher outputs 1. Otherwise, our distinguisher outputs 0. This distinguisher's advantage is exactly equal to the difference between $\Pr[\mathcal{B}'(\mathcal{A}) \to (\textsc{EarlyAbort})]$ and $\Pr[\mathcal{B}''(\mathcal{A}) \to (\textsc{EarlyAbort})]$. Our hardness assumption therefore implies that $\Pr[\mathcal{B}'(\mathcal{A}) \to (\textsc{EarlyAbort})] \leq \Pr[\mathcal{B}''(\mathcal{A}) \to (\textsc{EarlyAbort})] + \epsilon_{DDH}$. □

We now consider the remaining abort conditions in $\mathcal{B}''$.

Lemma 5. *If t-OMDL is $(\epsilon_{OMDL}, T_{OMDL})$-hard in $\mathbb{G}_1$ and $\mathcal{A}$ makes at most q queries to H_0 and $\overline{q}$ queries to H_1 and H_2, then* $\Pr[\mathcal{B}''(\mathcal{A}) \to (\text{III.}\textsc{BadInterp})] \leq 2 \cdot \epsilon_{OMDL}$.

Proof. At the time of a (III.$\textsc{BadInterp}$) abort, we have found some $\ell, \overline{j} \in [L] \times [\overline{q}]$ such that, in the representation of y^*_ℓ, the exponent of either $\overline{h}_{\overline{j}}$ or $\hat{\overline{h}}_{\overline{j}}$ is not a properly-formed interpolation of some $t+1$ $\{z_i\}$ or $\{\hat{z}_i\}$ keys, respectively. Without loss of generality, assume that it's the $\overline{h}_{\overline{j}}$ component that triggered the abort (the $\hat{\overline{h}}_{\overline{j}}$ case is entirely symmetric). Define F as the event that the following equation holds:

$$(s^{\ell}_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^{\ell}_G \kappa_G \overline{s}^G_{\overline{j}}) + \sum_{i \in \mathbf{S}(\overline{j})} (v^{\ell}_{i,\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^{\ell}_G \kappa_G \overline{v}^G_{i,\overline{j}}) \cdot z_i = 0 \qquad (9)$$

Intuitively, event F states that no H_1 responses are meaningfully used in $\mathcal{A}$'s algebraic representation of $y^*_\ell = H_0(x^*_\ell)^{k+\alpha}$.

Given a t-OMDL input $\mathsf{challenge} = (g_1, (g_1)^{\xi_1}, \ldots, (g_1)^{\xi_t})$, we use $\mathcal{A}$ to solve for all $\{\xi_\iota\}$. We follow one of two reduction strategies at random. Strategy #1 succeeds if event F occurs, and strategy #2 succeeds if it doesn't.

Strategy #1

Run $\mathcal{B}''(\mathcal{A})$, but using the group elements of $\mathsf{challenge}$ to compute everything involving the secret keys $\{z_i\}$ and $\{\hat{z}_i\}$. In particular:

- For each $\iota \in [t]$, pick $\nu_\iota \leftarrow_\$ \mathbb{Z}_m$ and define $\hat{\xi}_\iota := \xi_\iota + \nu_\iota$. The $\{\hat{\xi}_\iota\}$ values cannot be directly computed, but $\{(g_1)^{\hat{\xi}_\iota}\}$ can.
- In step I, define polynomials $z(i) := \sum_{\iota=1}^{t} \xi_\iota i^\iota$ and $\hat{z}(i) := \sum_{\iota=1}^{t} \hat{\xi}_\iota i^\iota$. Once again, these polynomials are only computable in the exponent of g_1; this is sufficient to compute the public keys.
- In response to an SEval query to server i with input $(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})$, we can use the H_1 and H_2 trapdoors to compute the correct response in spite of not directly knowing z_i and $\hat{z}_i$: $(p_{\overline{j}})^{k_i} \cdot (\overline{h}_{\overline{j}})^{z_i} \cdot (\hat{\overline{h}}_{\overline{j}})^{\hat{z}_i} = (p_{\overline{j}})^{k_i} \cdot ((g_1)^{z_i})^{\overline{\tau}_{\overline{j}}} \cdot ((g_1)^{\hat{z}_i})^{\hat{\overline{\tau}}_{\overline{j}}}$.
- When $\mathcal{A}$ wishes to $\mathsf{Corrupt}$ server i, use a single discrete logarithm oracle call to find the correct z_i and $\hat{z}_i$ to return. In particular, $z_i = \mathsf{DL}_{g_1}((g_1)^{z_i})$ and $\hat{z}_i = z_i - \sum_{\iota=1}^{t} \nu_\iota i^\iota$.
- In step III, we treat z_i and $\hat{z}_i$ (for uncorrupted i) as unknown variables.

The first condition that could trigger (III.BadInterp) is $|\mathbf{S}(\overline{j})| < t + 1 - |\mathsf{corrupt}|$. Equation (9) has only $|\mathbf{S}(\overline{j})|$ unknowns $\{z_i\}_{i \in \mathbf{S}(\overline{j})}$. Use $\mathsf{DL}_{g_1}(\cdot)$ $|\mathbf{S}(\overline{j})|-1$ times to learn all of them but one, then use equation (9) to solve for the last. Use $\mathsf{DL}_{g_1}(\cdot)$ $t - |\mathsf{corrupt}| - |\mathbf{S}(\overline{j})|$ more times until a total of t z_i values are known. The underlying challenge solutions $\{\xi_\iota\}_{\iota \in [t]}$ can then be interpolated. Only $t-1$ $\mathsf{DL}_{g_1}(\cdot)$ oracle queries have been made.

The second condition that could trigger (III.BadInterp) is that there exists no $\mu^\ell_{\overline{j}}$ satisfying $(s^\ell_{\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{s}^G_{\overline{j}}) = \mu^\ell_{\overline{j}}(\sum_{i \in \mathsf{corrupt}} \lambda^{\overline{j}}_i z_i)$ and $(v^\ell_{i,\overline{j}} + \sum_{G=\overline{j}+1}^{\overline{q}} \mu^\ell_G \kappa_G \overline{v}^G_{i,\overline{j}}) = \mu^\ell_{\overline{j}} \lambda^{\overline{j}}_i$ for all $i \in \mathbf{S}(\overline{j})$. However, since $|\mathbf{S}(\overline{j})| = t + 1 - |\mathsf{corrupt}|$, we know that $\{z_i\}_{i \in \mathsf{corrupt} \cup \mathbf{S}(\overline{j})}$ do in fact interpolate to 0:

$$\sum_{i \in \mathsf{corrupt}} \lambda^{\overline{j}}_i z_i + \sum_{i \in \mathbf{S}(\overline{j})} \lambda^{\overline{j}}_i z_i = 0 \tag{10}$$

Equations (9) and (10) are independent linear equations with the same $t + 1 - |\mathsf{corrupt}|$ unknowns $\{z_i\}_{i \in \mathbf{S}(\overline{j})}$. Use $\mathsf{DL}_{g_1}(\cdot)$ $t - 1 - |\mathsf{corrupt}|$ times to learn all of them but two, then use equations (9) and (10) to solve for the last two. A total of $t + 1$ z_i values are now known, so the underlying challenge solutions $\{\xi_\iota\}_{\iota \in [t]}$ can be interpolated. Only $t - 1$ $\mathsf{DL}_{g_1}(\cdot)$ oracle queries have been made.

Strategy #2

There is an obvious reduction from t-OMDL to DL, and (per Lemma 1) there is a reduction from DL to RDL. Therefore, we present strategy #2 as an RDL reduction on input $((g_1)^{\chi_k}, (g_1)^{\chi_z}, (g_1)^{\chi_{\hat{z}}}, (g_1)^{\chi_1}, \ldots, (g_1)^{\chi_{q+2\overline{q}}})$.

Run $\mathcal{B}''(\mathcal{A})$, but use $(g_k, g_z, g_{\hat{z}}) := ((g_1)^{\chi_k}, (g_1)^{\chi_z}, (g_1)^{\chi_{\hat{z}}})$. For all $j \in [q]$ use $h_j := (g_1)^{\chi_j}$ to respond to random oracle query $H_0(x_j)$. For all $\overline{j} \in [\overline{q}]$ use $\overline{h}_{\overline{j}} := (g_1)^{\chi_{q+\overline{j}}}$ and $\hat{\overline{h}}_{\overline{j}} := (g_1)^{\chi_{q+\overline{q}+\overline{j}}}$ to respond to random oracle queries $H_1(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})$ and $H_2(\mathsf{ctx}_{\overline{j}}, p_{\overline{j}})$, respectively. To confirm that this is possible, observe that the $\tau_k, \tau_z, \tau_{\hat{z}}, \{\tau_j\}_{j\in[q]}, \{\overline{\tau}_{\overline{j}}, \hat{\overline{\tau}}_{\overline{j}}\}_{j\in[\overline{q}]}$ trapdoors are never used by $\mathcal{B}''$.

Since (II.AdvLoses) did not occur, y^*_ℓ must correctly equal $(h_{j(\ell)})^k$. If F does not occur, then the exponent of $\overline{h}_{\overline{j}}$ is nonzero in equation (3). This is nearly an RDL solution, but we must eliminate the $\{p_J\}_{J\in[\overline{j}]^+}$ bases. Recursively expand each one using its representation, equation (2), until only h_j, $\overline{h}_G$, and $\hat{\overline{h}}_G$ bases remain. Since p_J can only include $\overline{h}_G$ and $\hat{\overline{h}}_G$ factors for $G < J$, this expansion does not tamper with the exponent of $\overline{h}_{\overline{j}}$, which therefore remains nonzero. Thus, we have an RDL solution.

The probability of correctly guessing whether or not F will occur is $1/2$, so this procedure wins t-OMDL with probability $1/2 \cdot \Pr[\mathcal{B}''(\mathcal{A}) \to (\text{III.BadInterp})]$. Our hardness assumption therefore implies that $\Pr[\mathcal{B}''(\mathcal{A}) \to (\text{III.BadInterp})] \leq 2 \cdot \epsilon_{OMDL}$. □

Finally we are ready to combine our previous results and conclude the proof.

Theorem 1. *If $\overline{q}$-co-SDL is $(\epsilon_{SDL}, T_{SDL})$-hard in $(\mathbb{G}_1, \mathbb{G}_2)$, DDH is $(\epsilon_{DDH}, T_{DDH})$-hard in $\mathbb{G}_1$, t-OMDL is $(\epsilon_{OMDL}, T_{OMDL})$-hard in $\mathbb{G}_1$, $\mathcal{A}$ runs in time $\mathcal{O}(\min\{T_{OMDL}, T_{DDH}, T_{SDL}\} - q\overline{q}^2 - q\log m - \overline{q}^{1.815}\log m)$, $\mathcal{A}$ is algebraic with respect to $\mathbb{G}_1$, simulation of $\mathsf{NIZK}_{\mathsf{KeyGen}}$ and $\mathsf{NIZK}_{\mathsf{SEval}}$ incurs security loss ϵ_{NIZK}, $\mathcal{A}$ makes at most q queries to H_0 and $\overline{q}$ queries to H_1 and H_2, and H_0, H_1, and H_2 are modeled as random oracles, then $\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}} \leq 2 \cdot \epsilon_{OMDL} + \epsilon_{DDH} + 4 \cdot \epsilon_{SDL} + \epsilon_{NIZK}$. Thus, $\mathsf{2B\text{–}HashTDH}$ is one-more unpredictable (Definition 14).*

Proof. Recall that the execution of $\mathcal{B}(\mathcal{A})$ ends with a contradiction, and it therefore aborts before that point with probability 1.

$$
\begin{aligned}
1 &= \Pr[\mathcal{B}(\mathcal{A}) \to (\text{EarlyAbort})] + \Pr[\mathcal{B}(\mathcal{A}) \to (\text{IV.KeyKnown})] \\
&\leq \Pr[\mathcal{B}(\mathcal{A}) \to (\text{EarlyAbort})] + 4 \cdot \epsilon_{SDL} \\
&\leq \Pr[\mathcal{B}''(\mathcal{A}) \to (\text{EarlyAbort})] + \epsilon_{DDH} + 4 \cdot \epsilon_{SDL} \\
&= \Pr[\mathcal{B}''(\mathcal{A}) \to (\text{II.AdvLoses})] + \Pr[\mathcal{B}''(\mathcal{A}) \to (\text{III.BadInterp})] \\
&\qquad + \epsilon_{DDH} + 4 \cdot \epsilon_{SDL} \\
&\leq \Pr[\mathcal{B}''(\mathcal{A}) \to (\text{II.AdvLoses})] + 2 \cdot \epsilon_{OMDL} + \epsilon_{DDH} + 4 \cdot \epsilon_{SDL}
\end{aligned}
$$

$\mathcal{B}''$ differs from $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}$ only in the use of simulated NIZKs. Define one final hybrid $\mathcal{B}'''$ that uses real NIZKs (i.e. $\mathsf{NIZK}_{\mathsf{KeyGen}}.\mathsf{Prove}$ and $\mathsf{NIZK}_{\mathsf{SEval}}.\mathsf{Prove}$) instead. By the simulatability of the NIZKs (Appendix B), $|\Pr[\mathcal{B}''(\mathcal{A}) \to (\text{II.AdvLoses})] - \Pr[\mathcal{B}'''(\mathcal{A}) \to (\text{II.AdvLoses})]| \leq \epsilon_{NIZK}$.

$\mathcal{B}'''$ simulates $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}$ perfectly and outputs (II.AdvLoses) iff $\mathcal{A}$ fails the game's win condition. Therefore, $\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}} = 1 - \Pr[\mathcal{B}'''(\mathcal{A}) \rightarrow (\text{II.AdvLoses})]$, and consequently:

$$\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}} \leq 2 \cdot \epsilon_{OMDL} + \epsilon_{DDH} + 4 \cdot \epsilon_{SDL} + \epsilon_{NIZK} \qquad \square$$

4.1 Threshold Blind Signature Security

Now that tOPRF security is proven, we provide a simple lemma saying that any publicly verifiable tOPRF is also a tBSig. Even though a tBSig doesn't need to be a tOPRF (specifically, it doesn't need to be deterministic and pseudorandom), our 2B–tBlindBLS scheme is a tOPRF, so this lemma enables us to quickly prove its security.

Lemma 6. *Suppose there exists a threshold oblivious pseudorandom function (Definition 8)* $\mathsf{tOPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{UEval}, \mathsf{SEval}, \mathsf{UAgg}, \mathsf{Offline})$ *and a PPT algorithm* Verify *such that for all integers* $0 \leq t < n$*, all* $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}) \leftarrow \mathsf{KeyGen}(n,t)$*, and all* $(x,y) \in \mathcal{X} \times \mathcal{Y}$*:*

$$\mathsf{Verify}(\mathsf{pk}, x, y) = 1 \iff y = \mathsf{Offline}(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{aux}, x) \tag{11}$$

where pk *is part of* aux.

It follows that $\mathsf{tBSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{UEval}, \mathsf{SEval}, \mathsf{UAgg}, \mathsf{Verify})$ *is a threshold blind signature scheme (where the output of* KeyGen *is parsed as* $(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{pk}, \mathsf{aux})$*).*

Proof. To prove Lemma 6, we must show that tBSig is verifiably correct (Definition 16), blind (Definition 17), and one-more unforgeable (Definition 18). These three properties are immediately implied by the deterministic correctness (Definition 9), blindness (Definition 12), and one-more unpredictability (Definition 14) of tOPRF, because equation (11) makes the respective definitions equivalent. $\square$

Per the definition of a bilinear pairing, 2B–tBlindBLS.Verify indeed satisfies equation (11). Therefore, by Lemma 6, 2B–tBlindBLS is a secure threshold blind signature scheme.

Note that our security proof for 2B–tBlindBLS depends upon the DDH assumption in $\mathbb{G}_1$. If the protocol is instantiated with a type 1 (i.e. symmetric) pairing setup, then the pairing function itself acts as a DDH tester. Therefore, the protocol should be instantiated only with a type 2 or 3 (i.e. asymmetric) pairing setup. We do not view this as a significant drawback; in practice, type 3 setups are more performant than types 1 or 2 anyways [47].

References

1. Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 311–341. Springer, Cham, December 2021.

2. Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 136–151. Springer, Berlin, Heidelberg, May 2001.
3. Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Berlin, Heidelberg, August 2004.
4. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572. IEEE Computer Society Press, May 2022.
5. Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Berlin, Heidelberg, May / June 2006.
6. Ghous Ali Amjad, Scott Hendrickson, Christopher A. Wood, and Kevin W. L. Yeo. Partially Blind RSA Signatures. Internet-Draft draft-irtf-cfrg-partially-blind-rsa-00, Internet Engineering Task Force, September 2024. Work in Progress.
7. iCloud Private Relay Overview, Dec 2021. `https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf`.
8. Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 193–207, New York, NY, USA, 2022. Association for Computing Machinery.
9. Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 429–459. Springer, Cham, May 2024.
10. Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008.
11. Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 433–444. ACM Press, October 2011.
12. Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022.
13. Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Paper 2022/833, 2022. `https://eprint.iacr.org/2022/833`.
14. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003.
15. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.
16. L. Brandao and Peralta R. NIST first call for multi-party threshold schemes, 2023. `https://csrc.nist.gov/publications/detail/nistir/8214c/draft`.

17. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 345–356. ACM Press, October 2008.
18. Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 388–407. Springer, Berlin, Heidelberg, August 2001.
19. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report No. 260, March 1997. `ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/`.
20. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
21. 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.
22. Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Berlin, Heidelberg, August 1999.
23. Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. SoK: Oblivious pseudorandom functions. Cryptology ePrint Archive, Report 2022/302, 2022.
24. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Cham, May 2020.
25. Sofia Celi, Alex Davidson, Steven Valdez, and Christopher A. Wood. Privacy Pass Issuance Protocols. RFC 9578, June 2024.
26. 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, Cham, August 2022.
27. Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Pairing-free blind signatures from CDH assumptions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 174–209. Springer, Cham, August 2024.
28. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.
29. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, New York, August 1990.
30. Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023.
31. Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe,

editors, *Public-Key Cryptography – PKC 2022*, pages 409–438, Cham, 2022. Springer International Publishing.
32. Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*, page 678–709, Berlin, Heidelberg, 2023. Springer-Verlag.
33. Elizabeth 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, *Advances in Cryptology – CRYPTO 2023*, pages 710–742, Cham, 2023. Springer Nature Switzerland.
34. Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Cham, August 2023.
35. Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 710–742. Springer, Cham, August 2023.
36. Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 382–400. Springer, Cham, September 2020.
37. Poulami Das, Julia Hesse, and Anja Lehmann. DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 682–696. ACM Press, May / June 2022.
38. Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 356–370. ACM Press, November 2023.
39. Sourav Das and Ling Ren. Adaptively secure bls threshold signatures from ddh and co-cdh. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 251–284, Cham, 2024. Springer Nature Switzerland.
40. Frank Denis, Frederic Jacobs, and Christopher A. Wood. RSA Blind Signatures. RFC 9474, October 2023.
41. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Berlin, Heidelberg, August 1988.
42. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990.
43. Jack Doerner, Yashvanth Kondi, Eysa Lee, abhi shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *2023 IEEE Symposium on Security and Privacy*, pages 773–789. IEEE Computer Society Press, May 2023.
44. Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 180–194. Springer, Berlin, Heidelberg, November 1999.

45. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Berlin, Heidelberg, February 2005.
46. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 1:1–49, 2018.
47. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008. Applications of Algebra to Cryptography.
48. Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. Cryptology ePrint Archive, Report 2023/567, 2023.
49. 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.
50. 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 16International Conference on Applied Cryptography and Network Security*, volume 9696 of *LNCS*, pages 156–174. Springer, Cham, June 2016.
51. Trust Tokens, Apr 2024. `https://developer.chrome.com/docs/privacy-sandbox/trust-tokens/`.
52. Panagiotis Grontas, Aris Pagourtzis, Alexandros Zacharakis, and Bingsheng Zhang. Towards everlasting privacy and efficient coercion resistance in remote electronic voting. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 210–231. Springer, Berlin, Heidelberg, March 2019.
53. Y. Gu, S. Jarecki, P. Kedzior, P. Nazarian, and J. Xu. Threshold PAKE with security against compromise of all servers. In *Advances in Cryptology – ASIACRYPT 2024*, 2024.
54. Christian Hanser. *Signatures on Equivalence Classes: A New Tool for Privacy-Enhancing Cryptography*. PhD thesis, Graz University of Technology, Austria, February 2016.
55. Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Rai-choo! Evolving blind signatures to the next level. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 753–783. Springer, Cham, April 2023.
56. Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, Berlin, Heidelberg, March 2008.
57. Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 43–60. Springer, Berlin, Heidelberg, February 2016.
58. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Berlin, Heidelberg, December 2014.

59. Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17International Conference on Applied Cryptography and Network Security*, volume 10355 of *LNCS*, pages 39–58. Springer, Cham, July 2017.
60. Stanislaw Jarecki, Hugo Krawczyk, and Jason K. Resch. Updatable oblivious key management for storage systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 379–393. ACM Press, November 2019.
61. Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Cham, April / May 2018.
62. Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Berlin, Heidelberg, March 2009.
63. Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 418–435. Springer, Berlin, Heidelberg, September 2010.
64. Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, page 398–406, New York, NY, USA, 1995. Association for Computing Machinery.
65. Julia Kastner, Ky Nguyen, and Michael Reichle. Pairing-free blind signatures from standard assumptions in the ROM. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 210–245. Springer, Cham, August 2024.
66. Jinho Kim, Kwangjo Kim, and Chulsoo Lee. An efficient and provably secure threshold blind signature. In *4th International Conference on Information Security and Cryptology – ICISC'01*, Berlin, Heidelberg, 2001. Springer-Verlag.
67. Sungwook Kim, Hyeonbum Lee, and Jae Hong Seo. Efficient zero-knowledge arguments in discrete logarithm setting: Sublogarithmic proof or sublinear verifier. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 403–433, Cham, 2022. Springer Nature Switzerland.
68. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.
69. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.
70. Veronika Kuchta and Mark Manulis. Rerandomizable threshold blind signatures. In *Trusted Systems- 6th International Conference – INTRUST 2014*, 2014.
71. 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.

72. Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Cham, August 2017.

73. 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.

74. Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Berlin, Heidelberg, December 2001.

75. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, mar 2004.

76. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, Berlin, Heidelberg, August 1992.

77. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

78. Víctor Reyes-Macedo, Akinori Kawachi, Gina Gallegos-García, and Moisés Salinas-Rosales. A threshold-blind signature scheme and its application in blockchain-based systems. *IEEE Access*, 12:138239–138251, 2024.

79. Alfredo Rial and Ania M. Piotrowska. Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive, Report 2022/011, 2022.

80. Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, November 2001.

81. Markus Stadler. Publicly verifiable secret sharing. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 190–199, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

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

83. Masayuki Tezuka and Keisuke Tanaka. Redactable signature with compactness from set-commitment. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E104.A(9):1175–1187, September 2021.

84. Duc-Liem Vo, Fangguo Zhang, and Kwangjo Kim. A new threshold blind signature scheme from pairings. In *Symposium on Cryptography and Information Security – SCIS 2023*, 2003.

85. Joachim von zur Gathen and Daniel Panario. Factoring polynomials over finite fields: A survey. *Journal of Symbolic Computation*, 31(1):3–17, 2001.

86. Pengfei Wang, Xiangyu Su, Mario Larangeira, and Keisuke Tanaka. Auditable attribute-based credentials scheme and its application in contact tracing. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 88–118, Cham, 2024. Springer Nature Switzerland.

A Proof of Lemma 1

Proof. Suppose there exists an adversary $\mathcal{A}$ that runs in time T and solves N-RDL with probability ϵ_{RDL}. We can use $\mathcal{A}$ to solve for ξ given a DL input (g, g^{ξ}). For each $i \in [N]$, pick $a_i, b_i \leftarrow_{\$} \mathbb{Z}_m$ and define $h_i := g^{a_i} \cdot (g^{\xi})^{b_i}$. Run $\mathcal{A}$ on RDL challenge $(h_1, \ldots, h_N)$. Notice that, in effect, $\xi_i = a_i + b_i\xi$ for all i.

If $\mathcal{A}$ is successful, it outputs $(c_1, \ldots, c_N)$ such that $\prod_{i\in[N]}(h_i)^{c_i} = 1$. In the exponent, we have the equation $\sum_{i\in[N]}(a_i + b_i\xi)c_i = 0$. We can isolate $\xi = (-\sum_{i\in[N]} a_i c_i)/(\sum_{i\in[N]} b_i c_i)$. As long as the denominator $\sum_{i\in[N]} b_i c_i \neq 0$, we solve DL.

Since $\mathcal{A}$ is successful, $\exists_{i\in[N]}\ c_i \neq 0$. Notice that b_i is information-theoretically hidden from $\mathcal{A}$, i.e. h_i is independent of b_i. It follows by a one-time pad argument that $\sum_{i\in[N]} b_i c_i$ is a uniformly random element of $\mathbb{Z}_m$ and $\Pr[\sum_{i\in[N]} b_i c_i = 0] = \frac{1}{m}$. Overall, then, our probability of solving DL is $(1 - \frac{1}{m}) \cdot \epsilon_{RDL}$. Our hardness assumption therefore implies that $\epsilon_{RDL} \leq \frac{m}{m-1} \cdot \epsilon$. □

B Non-Interactive Zero Knowledge Proofs

The protocols in Figure 3 require two non-interactive zero knowledge proof of knowledge systems (NIZKs), denoted $\mathsf{NIZK}_{\mathsf{KeyGen}}$ and $\mathsf{NIZK}_{\mathsf{SEval}}$. Intuitively, the former ensures that public keys are well-formed, and the latter ensures that servers behave honestly. We represent a NIZK as a pair of algorithms $(\mathsf{Prove}, \mathsf{Verify})$ for a relation $\mathcal{R}$ of statement/witness pairs. $\mathsf{Prove}(s, w) \rightarrow \pi$ produces a proof of knowledge π for $(s, w) \in \mathcal{R}$. $\mathsf{Verify}(s, \pi) \rightarrow 0/1$ verifies whether π proves knowledge of a witness w such that $(s, w) \in \mathcal{R}$.

- $\mathsf{NIZK}_{\mathsf{KeyGen}}$ is a proof system for statements of the form $\mathsf{pk} \in \mathbb{G}_1$ (for 2B–HashTDH) or $\mathsf{pk} \in \mathbb{G}_2$ (for 2B–tBlindBLS) and witnesses of the form $k \in \mathbb{Z}_m$. Relation $\mathcal{R}_{\mathsf{KeyGen}}$ is the set of (pk, k) pairs such that $\mathsf{pk} = (g_k)^k$ (for 2B–HashTDH) or $\mathsf{pk} = (g_2)^k$ (for 2B–tBlindBLS).
- $\mathsf{NIZK}_{\mathsf{SEval}}$ is a proof system for statements of the form $(p, h_1, h_2, \mathsf{pk}_i, q_i) \in (\mathbb{G}_1)^5$ and witnesses of the form $(k_i, z_i, \hat{z}_i) \in (\mathbb{Z}_m)^3$. Relation $\mathcal{R}_{\mathsf{SEval}}$ is the set of $((p, h_1, h_2, \mathsf{pk}_i, q_i), (k_i, z_i, \hat{z}_i))$ pairs such that $\mathsf{pk}_i = (g_k)^{k_i} \cdot (g_z)^{z_i} \cdot (g_{\hat{z}})^{\hat{z}_i}$ and $q_i = (p)^{k_i} \cdot (h_1)^{z_i} \cdot (h_2)^{\hat{z}_i}$.

These NIZKs are required to satisfy Completeness, Simulatability, and Strong Soundness. We consider NIZKs in the random oracle model, where the Prove and Verify algorithms have oracle access to a function H_{RO} randomly sampled from some distribution $\mathcal{H}$.

Definition 19 (Completeness). *Non-interactive zero knowledge proof of knowledge system* NIZK *for relation* $\mathcal{R}$ *is* complete *iff, for every* $(s, w) \in \mathcal{R}$, $\mathsf{NIZK.Verify}(s, \mathsf{NIZK.Prove}(s, w))$ *always produces 1.*

Definition 20 (Simulatability). *The advantage of adversary $\mathcal{A}$ against protocol* NIZK *for relation $\mathcal{R}$ in the simulatability game is defined as:*

$$\mathbf{Adv}^{\mathsf{simulatable}}_{\mathcal{A},\mathsf{NIZK},\mathsf{SIM}}(\kappa) = |\Pr[\mathcal{A}^{\mathsf{NIZK.Prove}(\cdot,\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\mathsf{SIM}(\cdot)}(1^\kappa) = 1]|$$

where $\mathcal{A}$ can only query its oracle on $(s, w) \in \mathcal{R}$ and SIM *is an algorithm that takes only the s part as its input. In the first probability, H_{RO} is randomly sampled from $\mathcal{H}$. In the second probability,* SIM *programs the H_{RO} queries of $\mathcal{A}$.*

Non-interactive zero knowledge proof of knowledge system NIZK *is* simulatable *iff there exists an efficient simulator* SIM *such that, for all PPT adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{simulatable}}_{\mathcal{A},\mathsf{NIZK},\mathsf{SIM}}$ is negligible.*

Definition 21 (Strong Soundness). *The acceptance event $acc_{\mathcal{A}}$ of prover $\mathcal{A}$ against protocol* NIZK *for relation $\mathcal{R}$ is defined as:*

$$\mathcal{A}(1^\kappa; \rho) \to (\{(s_i, \pi_i)\}_{i\in[\omega]}, \phi) \ \textit{s.t.}\ \forall_{i\in[\omega]}\ \mathsf{NIZK.Verify}(s_i, \pi_i) = 1$$

where $\mathcal{A}$ is a probabilistic algorithm that runs on randomness ρ sampled from some distribution $P_{\mathcal{A}}$. $\mathcal{A}$ outputs ω proofs $\{(s_i, \pi_i)\}_{i\in[\omega]}$ and a side output ϕ.

The acceptance event acc_{EXT} of extractor EXT *against prover $\mathcal{A}$ is defined as:*

$$\begin{aligned} &\mathcal{A}(1^\kappa; \rho) \to (\{(s_i, \pi_i)\}_{i\in[\omega]}, \phi) \ \textit{s.t.}\ \forall_{i\in[\omega]}\ \mathsf{NIZK.Verify}(s_i, \pi_i) = 1 \\ \wedge\ &\mathsf{EXT}^{\mathcal{A}(1^\kappa;\rho)}(1^\kappa) \to (\{w_i\}_{i\in[\omega]}) \ \textit{s.t.}\ \forall_{i\in[\omega]}\ (s_i, w_i) \in \mathcal{R} \end{aligned}$$

In this experiment, EXT *and the "main" execution of $\mathcal{A}$ have oracle access to random function H_{RO}. For the executions of $\mathcal{A}$ run by* EXT*,* EXT *programs H_{RO}. Every execution of $\mathcal{A}$ runs on the same randomness ρ.*

Non-interactive zero knowledge proof of knowledge system NIZK *is* sound *iff, for every PPT prover $\mathcal{A}$, if $\Pr_{\rho\leftarrow\$P_{\mathcal{A}}, H_{\mathsf{RO}}\leftarrow\$\mathcal{H}}[acc_{\mathcal{A}}]$ is non-negligible, then there exists a PPT extractor* EXT *such that $\Pr_{\rho\leftarrow\$P_{\mathcal{A}}, H_{\mathsf{RO}}\leftarrow\$\mathcal{H}}[acc_{\mathsf{EXT}}]$ is non-negligible.*

The extraction loss $L_{\mathsf{EXT}}(\cdot)$ is a lower-bound on the probability of acc_{EXT} as a function of the probability of $acc_{\mathcal{A}}$. The extraction time $T_{\mathsf{EXT}}(\cdot)$ is an upper-bound on the runtime of $\mathsf{EXT}^{\mathcal{A}}$ as a function of the runtime of $\mathcal{A}$.

Note that we are using a "multi-point" version of strong soundness that allows for (offline) extraction of multiple witnesses generated by a prover. This property is achieved by a NIZK created via the Fiat-Shamir transform applied to a Sigma protocol, per the generalized forking lemma of Bagherzandi et al. [10]. The Sigma protocols for relations $\mathcal{R}_{\mathsf{KeyGen}}$ and $\mathcal{R}_{\mathsf{SEval}}$ are known as proofs of discrete-logarithm respresentation: They are a straightforward extension of Schnorr's proof of discrete-logarithm knowledge, and their cost is comparable to Schnorr's signature, see e.g. the report by Camenisch and Stadler [19].

B.1 Performance Considerations

The security of 2B–HashTDH depends upon NIZK soundness only in its proof of blindness (Section 4). Even in that proof the NIZKs are only relevant to the

analysis of UAgg outputs, which are part of the blindness game but are excluded from the obliviousness game. Therefore, in applications where obliviousness is sufficient, NIZKs can be omitted from 2B–HashTDH entirely. In that case, users can evaluate the tOPRF without needing any public data (i.e. $\mathsf{pk}, \mathsf{aux}$). As mentioned in Section 2.2, obliviousness is the more typical security expectation for OPRFs. Only those applications that demand a robust (i.e. verifiable) tOPRF need incur the overhead of public keys and NIZKs.

For 2B–tBlindBLS, the situation is even better. NIZK soundness does not appear at all in the proofs of security. The only feature that the NIZKs provide, then, is "partial verifiability," i.e. the ability for a user to check whether each SEval response is correctly formed. This check is only important after an aggregated signature fails to verify. Therefore, we propose that NIZKs can and should be omitted from the normal operation of 2B–tBlindBLS. Instead, servers can provide an "Audit" interface by which a user can request a proof that a previous SEval response was correct. Since SEval is deterministic, servers can support this feature without even maintaining any state. In practice, the inevitability of being caught should strongly disincentivize dishonest server behavior, and NIZKs should be rarely needed.

C Proof of Blindness

We now prove that 2B–HashTDH and 2B–tBlindBLS are blind. We argue that, under expected conditions, the view of $\mathcal{A}$ in $\mathbf{Game}^{\mathsf{blind}}$ is identically distributed regardless of the secret bit θ. In particular, the $\mathcal{O}^{\mathsf{UEval}}$ oracle always outputs two uniformly random $\mathbb{G}_1$ elements. If $\mathcal{A}$ sends honest server responses to $\mathcal{O}^{\mathsf{UAgg}}$, then that oracle outputs $(H_0(x_{0,sid})^k, H_0(x_{1,sid})^k)$, which does not depend on θ. If $\mathcal{A}$ sends dishonest responses, then UAgg is expected to abort and output $(\perp, \perp)$, which obviously reveals no information about θ.

Thus, blindness is perfect unless, for some sid, $\mathcal{O}^{\mathsf{UAgg}}$ produces a (non-$\perp$) output that is not of the form $(H_0(x_{0,sid})^k, H_0(x_{1,sid})^k)$ for some k. For 2B–tBlindBLS, this event is impossible due to the pairing-based verification check that UAgg performs; if y_0 and y_1 are not correctly formed (for some given public key pk), UAgg is guaranteed to abort. For 2B–HashTDH, the event is not impossible, but it occurs with negligible probability.

Theorem 2. *If DL is (ϵ_{DL}, T_{DL})-hard in $\mathbb{G}_1$, $\mathsf{NIZK}_{\mathsf{KeyGen}}$ and $\mathsf{NIZK}_{\mathsf{SEval}}$ are sound with extraction loss L_{EXT} and extraction time T_{EXT} (Definition 21), $\mathcal{A}$ runs in time $\mathcal{O}(T^{-1}_{\mathsf{EXT}}(T^{-1}_{\mathsf{EXT}}(T_{DL})))$, and $\mathcal{A}$ makes at most q_{UAgg} queries to $\mathcal{O}^{\mathsf{UAgg}}$, then $\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B–HashTDH}} \leq 2q_{\mathsf{UAgg}} \cdot L^{-1}_{\mathsf{EXT}}(L^{-1}_{\mathsf{EXT}}(\epsilon_{DL}))$. Thus, 2B–HashTDH is blind (Definition 12).*

Proof. Given an RDL input $\mathsf{challenge} = (g_k, g_z, g_{\hat{z}})$, we can use $\mathcal{A}$ to find $c_k, c_z, c_{\hat{z}}$ such that $(g_k)^{c_k} \cdot (g_z)^{c_z} \cdot (g_{\hat{z}})^{c_{\hat{z}}} = 1$ and $c_k \neq 0 \vee c_z \neq 0 \vee c_{\hat{z}} \neq 0$.

Define $\mathcal{B}$ to be an algorithm that runs $\mathbf{Game}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B–HashTDH}}$ but uses challenge to set the public parameters. As argued above, $\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B–HashTDH}}$ is

upper-bounded by the probability that, for some sid, $\mathcal{O}^{\mathsf{UAgg}}$ produces an unexpected output. In particular, an output is unexpected if at least one of $y_{0,sid}$ and $y_{1,sid}$ is not correctly formed for the pk given in aux_{sid}.

$\mathcal{B}$ selects a sid used by $\mathcal{A}$ at random and $\theta \leftarrow_{\$} \{0,1\}$. $y_{\theta,sid}$ is not correctly formed with probability at least $\frac{1}{2q_{\mathsf{UAgg}}} \cdot \mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}$. This event implies that all NIZKs $\pi_{\mathsf{KeyGen},sid}$ and $\{\pi_{i,\theta,sid}\}_{i \in \mathbf{S}_{\theta,sid}}$ successfully verify. $\mathcal{B}$ outputs these NIZKs.

Per the supposition that $\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}$ is non-negligible, the soundness of $\mathsf{NIZK}_{\mathsf{KeyGen}}$ implies the existence of an efficient extractor $\mathsf{EXT}_{\mathsf{KeyGen}}$ such that $\mathsf{EXT}^{\mathcal{B}}_{\mathsf{KeyGen}}$ outputs witness k for $\pi_{\mathsf{KeyGen},sid}$ with probability at least $L_{\mathsf{EXT}}(\frac{1}{2q_{\mathsf{UAgg}}} \cdot \mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}})$. Similarly, the soundness of $\mathsf{NIZK}_{\mathsf{SEval}}$ implies the existence of an efficient extractor $\mathsf{EXT}_{\mathsf{SEval}}$ such that $\mathsf{EXT}^{\mathsf{EXT}^{\mathcal{B}}_{\mathsf{KeyGen}}}_{\mathsf{SEval}}$ outputs witnesses $(k_i, z_i, \hat{z}_i)$ for all $\{\pi_{i,\theta,sid}\}_{i \in \mathbf{S}_{\theta,sid}}$ with probability at least $L_{\mathsf{EXT}}(L_{\mathsf{EXT}}(\frac{1}{2q_{\mathsf{UAgg}}} \cdot \mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}))$.

UAgg aborts unless public keys correctly interpolate, i.e. $\mathsf{pk} = (g_k)^k = \prod_{i \in \mathbf{S}_{\theta,sid}} \mathsf{pk}_i^{\lambda_i}$. Representing pk_i using the extracted $(k_i, z_i, \hat{z}_i)$, we have $(g_k)^k = (g_k)^{\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i k_i} \cdot (g_z)^{\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i z_i} \cdot (g_{\hat{z}})^{\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i \hat{z}_i}$. If $\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i k_i = k$, $\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i z_i = 0$, and $\sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i \hat{z}_i = 0$, then $y_{\theta,sid}$ would be correctly formed for public key $(g_k)^k$. It follows that one of these equalities does not hold, and we therefore have an RDL solution: $c_k = \sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i k_i - k$, $c_z = \sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i z_i$, $c_{\hat{z}} = \sum_{i \in \mathbf{S}_{\theta,sid}} \lambda_i \hat{z}_i$.

Putting it together, we have an $\mathcal{O}(T_{\mathsf{EXT}}(T_{\mathsf{EXT}}(T_{\mathcal{A}})))$ algorithm that wins RDL with probability $L_{\mathsf{EXT}}(L_{\mathsf{EXT}}(\frac{1}{2q_{\mathsf{UAgg}}} \cdot \mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}}))$. Our hardness assumption therefore implies that, if $T_{\mathcal{A}} = \mathcal{O}(T^{-1}_{\mathsf{EXT}}(T^{-1}_{\mathsf{EXT}}(T_{DL})))$, then $\mathbf{Adv}^{\mathsf{blind}}_{\mathcal{A},\mathsf{2B\text{–}HashTDH}} \leq 2q_{\mathsf{UAgg}} \cdot L^{-1}_{\mathsf{EXT}}(L^{-1}_{\mathsf{EXT}}(\epsilon_{DL}))$. □

D Universally Composable tOPRF

The UC *threshold* OPRF (tOPRF) functionality was first proposed by Jarecki et al. [59], based on the prior notion of UC OPRF [58]. Recently, Gu et al. [53] pointed out some ambiguities in the UC tOPRF model of [59] and proposed a fixed model, which we adopt here, as well as a protocol that securely realizes it in the static adversary setting under OMDH and DDH assumptions in ROM. We note that the UC (t)OPRF models of [58, 59, 53] are used as building blocks for applications such as password-protected secret sharing [59, 37], password-authenticated key exchange [58, 61], and a *threshold* pasword-authenticated key exchange (PAKE) [53], and that realizing the same functionality in the adaptive adversary setting, can lead to adaptive security of these applications. (This is the case e.g. in tPAKE of [53].)

In this section we present a variant $\mathsf{UC\text{–}2B\text{–}HashTDH}$ of our tOPRF protocol $\mathsf{2B\text{–}HashTDH}$, and we prove that it is adaptively secure under the UC model $\mathcal{F}_{\mathsf{tOPRF}}$ of [53]. We thus extend our earlier game-based results and

provide an adaptively-secure tOPRF that can be used as a building block into any of the aforementioned higher-level constructions. Since our proofs are in the AGM, we use the adaptation of the UC model to AGM by Abdalla et al. [1]. Roughly speaking, Abdalla et al. [1] show that UC composition theorem applies to the AGM model, as long as the simulator used in the security proof is also *algebraic*, i.e. that it uses the group like the AGM algorithm. Technically [1] require the functionality itself to be algebraic, but that's immediate in the case of $\mathcal{F}_{\mathsf{tOPRF}}$ since this functionality is oblivious to the group used in the protocol that implements it. Finally (and naturally), the composition holds only for environments which are algebraic with respect to the protocol, i.e. they use the group elements output by the homest parties only as an AGM adversary would. In the theorem below we use the [1] terminology that protocol π AGM-*emulates* functionality F, but this is an AGM model counterpart to the standard notion that protocol π UC-*realizes* functionality F of Canetti [20].

D.1 Security Model

Figure 4 is the $\mathcal{F}_{\mathsf{tOPRF}}$ functionality of [53] (with some minor syntactic changes).

$\mathcal{F}_{\mathsf{tOPRF}}$ is in some ways stronger than the game-based definition of tOPRF given in Section 2.2. For example, $\mathcal{F}_{\mathsf{tOPRF}}$ requires that a tOPRF be "one-more pseudorandom", i.e. until $t + 1$ server responses are collected, the function output is not just unpredictable but completely pseudorandom. Our game-based definition separately requires one-more unpredictability and pseudorandomness, but those properties do not imply one-more pseudorandomness. Consider a protocol where every server can independently compute just the first bit of the tOPRF output for any input. Such a protocol could be one-more unpredictable (as long as the rest of the bits are unpredictable without $t + 1$ servers' participation) and pseudorandom (as long as that first bit and the rest of the bits are pseudorandom), but it would not be one-more pseudorandom and it would not realize $\mathcal{F}_{\mathsf{tOPRF}}$.

$\mathcal{F}_{\mathsf{tOPRF}}$ also requires that, at the time of each evaluation, the ideal adversary commits to the instance *sid*, server set $\mathbf{S}$, and server context data ctx under which that evaluation is occuring. Somehow, a simulator must be able to "extract" that information from observing a real-world adversary's behavior. This property is also not guaranteed by the game-based tOPRF definition of Section 2.2.

D.2 Protocol

Figure 6 is $\mathsf{UC–2B–HashTDH}$, a tOPRF protocol realizing $\mathcal{F}_{\mathsf{tOPRF}}$. $\mathsf{UC–2B–HashTDH}$ makes use of $\mathcal{F}_{\mathsf{channel}}$, a functionality modeling a secure and authenticated communication channel, but only during initialization. $\mathcal{F}_{\mathsf{channel}}$ is defined in Figure 5.

$\mathsf{UC–2B–HashTDH}$ is essentially our main tORPF protocol $\mathsf{2B–HashTDH}$ augmented with an "outer" hash function $\tilde{H}$ applied to the function's final output y. This technique generically boosts one-more unpredictability to one-more pseudorandomness. By also including the function input x in the

Notation

The functionality interacts with a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}^*$. $\mathcal{P}^* = \mathcal{P} \cup \{\mathcal{A}^*\}$. **Corr** is the initial set of corrupted parties. By convention, $\mathcal{A}^* \in$ **Corr**. We assume strings sid of form $sid = (\ldots, \mathcal{S})$ where $\mathcal{S} = (\mathsf{S}_1, \ldots, \mathsf{S}_n) \in (\mathcal{P}^*)^n$. $\mathcal{S}_{sid}$ denotes the list $\mathcal{S}$ specified by string sid.

Initially $\mathsf{tx}[sid, \mathsf{ctx}, i] := 0$ and $F_{sid}(x)$ are undefined for all sid,ctx, i, x. When $F_{sid}(x)$ is first referenced $\mathcal{F}_{\mathsf{tOPRF}}$ assigns $F_{sid}(x) \leftarrow_{\$} \{0,1\}^l$.

Initialization

1. On $(\mathsf{toprf.init}, sid)$ from $\mathsf{P}_0 \in \mathcal{P}^*$, if sid is new (abort otherwise):
   - send $(\mathsf{toprf.init}, sid, \mathsf{P}_0)$ to $\mathcal{A}^*$
   - save $(\mathsf{toprf.init}, sid, \mathsf{P}_0)$ and mark it COMPR if $\mathsf{P}_0 \in$ **Corr**
2. On $(\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$ from S where $\mathsf{S} = \mathcal{S}_{sid}[i]$ or ($\mathsf{S} = \mathcal{A}^*$ and $\mathcal{S}_{sid}[i] \in$ **Corr**), save record $(\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$ marked INACTIVE
3. On $(\mathsf{toprf.fininit}, sid, i)$ from $\mathcal{A}^*$ where $\exists$ record $\mathsf{urec} = (\mathsf{toprf.init}, sid, \mathsf{P}_0)$ and record $\mathsf{srec} = (\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$ marked INACTIVE:
   - send $(\mathsf{toprf.sinit}, sid, i)$ to $\mathcal{S}_{sid}[i]$
   - if urec is COMPR or $\mathcal{S}_{sid}[i] \in$ **Corr** then mark srec COMPR
   - else (i.e. urec is not COMPR and $\mathcal{S}_{sid}[i] \notin$ **Corr**) mark srec ACTIVE

Corruption

4. On $(\mathsf{toprf.corrupt}, \mathsf{P})$ from $\mathcal{A}^*$ (with permission from $\mathcal{Z}$):
   - set **Corr** := **Corr** $\cup \{\mathsf{P}\}$
   - mark every ACTIVE record $(\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$ COMPR where $\mathsf{P} = \mathcal{S}_{sid}[i]$

Evaluation

5. On $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, x)$ from $\mathsf{U} \in \mathcal{P}^*$, if this is the first call from U for sid and $ssid_{\mathsf{U}}$:
   - send $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, \mathsf{U})$ to $\mathcal{A}^*$
   - save $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, \mathsf{U}, x)$ marked FRESH
6. On $(\mathsf{toprf.sndrcomplete}, sid, i, \mathsf{ctx})$ from S where $\exists$ record $\mathsf{srec} = (\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$ not marked INACTIVE and ($\mathsf{S} = \mathcal{S}_{sid}[i]$ or ($\mathsf{S} = \mathcal{A}^*$ and srec is marked COMPR)):
   - send $(\mathsf{toprf.sndrcomplete}, sid, i, \mathsf{ctx})$ to $\mathcal{A}^*$
   - set $\mathsf{tx}[sid, \mathsf{ctx}, i]{+}{+}$
7. On $(\mathsf{toprf.rcvcomplete}, sid, ssid_{\mathsf{U}}, sid^*, \mathsf{ctx}^*, \mathbf{S})$ from $\mathcal{A}^*$ where $|\mathbf{S}| = t + 1$ and $\exists$ record $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, \mathsf{U}, x)$ marked FRESH:
   - if $\exists j \in \mathbf{S}$ such that $\mathsf{tx}[sid^*, \mathsf{ctx}^*, j] = 0$ then abort
   - otherwise mark the record COMPLETED, set $\mathsf{tx}[sid^*, \mathsf{ctx}^*, j]{-}{-}$ for all $j \in \mathbf{S}$, and send $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, F_{sid^*}(x))$ to U

Fig. 4. $\mathcal{F}_{\mathsf{tOPRF}}$: threshold OPRF functionality, parameterized by threshold t, number of servers n, and output length l.

Notation

The functionality interacts with a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}^*$.

Secure Channel

1. On $(\mathsf{channel.send}, sid, \mathsf{R}, m)$ from $\mathsf{S} \in \mathcal{P}$:
   - save $(\mathsf{channel.message}, sid, \mathsf{S}, \mathsf{R}, m)$ marked PENDING
   - send $(\mathsf{channel.send}, sid, \mathsf{S}, \mathsf{R}, |m|)$ to $\mathcal{A}^*$
2. On $(\mathsf{channel.deliver}, sid, \mathsf{S}, \mathsf{R})$ from $\mathcal{A}^*$ where $\exists$ record $(\mathsf{channel.message}, sid, \mathsf{S}, \mathsf{R}, m)$ marked PENDING:
   - mark the record COMPLETED
   - send $(\mathsf{channel.deliver}, sid, \mathsf{S}, m)$ to R

Fig. 5. $\mathcal{F}_{\mathsf{channel}}$: secure and authenticated communication functionality

Setup

$\mathbb{G}_1$ is a group of prime order m with generator g_1. $H_0 : \{0,1\}^* \to \mathbb{G}_1$, $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \{0,1\}^* \to \mathbb{G}_1$, and $\tilde{H} : \{0,1\}^* \to \{0,1\}^l$ are hash functions.

Initialization

1. On input $(\mathsf{toprf.init}, sid)$, initializer P_0 does:
   - pick $a_0, ..., a_t, b_1, \ldots, b_t, \hat{b}_1, \ldots, \hat{b}_t \leftarrow_\$ \mathbb{Z}_m$
   - define polynomials $k(i) := \sum_{\iota=0}^{t} a_\iota i^\iota$, $z(i) := \sum_{\iota=1}^{t} b_\iota i^\iota$, $\hat{z}(i) := \sum_{\iota=1}^{t} \hat{b}_\iota i^\iota$
   - $\forall_{i \in [n]}\ k_i := k(i), z_i := z(i), \hat{z}_i := \hat{z}(i)$
   - for each $i \in [n]$, send $(\mathsf{channel.send}, (sid, i), \mathcal{S}_{sid}[i], (i, k_i, z_i, \hat{z}_i))$ to $\mathcal{F}_{\mathsf{channel}}$
2. On input $(\mathsf{toprf.sinit}, sid, i, \mathsf{P}_0)$, server $\mathcal{S}_{sid}[i]$ does:
   - await $(\mathsf{channel.deliver}, (sid, i), \mathsf{P}_0, (i, k_i, z_i, \hat{z}_i))$ from $\mathcal{F}_{\mathsf{channel}}$;
   - then save record $(\mathsf{toprf.share}, sid, i, k_i, z_i, \hat{z}_i)$
   - output $(\mathsf{toprf.fininit}, sid, i)$

Evaluation

3. On input $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, x)$, evaluator U does:
   - pick $r \leftarrow_\$ \mathbb{Z}_m$ and compute $p := H_0(x)^r$
   - for each $i \in [n]$, send $(sid, i, ssid_{\mathsf{U}}, p)$ to $\mathcal{S}_{sid}[i]$
   - await responses $(sid, i, ssid_{\mathsf{U}}, q_i)$ from $\mathcal{S}_{sid}[i]$ for all $i \in \mathbf{S}$, for any set $\mathbf{S} \subseteq [n]$ of size $t+1$;
   - then compute $q := \prod_{i \in \mathbf{S}} q_i^{\lambda_i}$ where λ_i is the Lagrange interpolation coefficient for index i and index set $\mathbf{S}$
   - output $(\mathsf{toprf.eval}, ssid_{\mathsf{U}}, \tilde{H}(x, q^{1/r}))$
4. On input $(\mathsf{toprf.sndrcomplete}, sid, i, \mathsf{ctx})$, server $\mathcal{S}_{sid}[i]$ does:
   - retrieve record $(\mathsf{toprf.share}, sid, i, k_i, z_i, \hat{z}_i)$ (abort if not found)
   - await $(sid, i, ssid_{\mathsf{U}}, p)$ from any U (if it hasn't already been received);
   - then compute $q_i := p^{k_i} \cdot H_1(\mathsf{ctx}, p)^{z_i} \cdot H_2(\mathsf{ctx}, p)^{\hat{z}_i}$
   - send response $(sid, i, ssid_{\mathsf{U}}, q_i)$ to U

Fig. 6. Protocol UC–2B–HashTDH which realizes $\mathcal{F}_{\mathsf{tOPRF}}$ in the $\mathcal{F}_{\mathsf{channel}}$-hybrid world.

input to $\tilde{H}$, we solve our protocol's *sid* extractability issue. (**S** and ctx are extractable due to the blinding factors used by the servers.)

D.3 Proof of Security

Theorem 3. *Protocol* UC–2B–HashTDH $\mathbb{G}_1$-AGM *emulates functionality* $\mathcal{F}_{\mathsf{tOPRF}}$ *with parameters* t *and* n *in the* $(\mathcal{F}_{\mathsf{channel}}, \mathcal{F}_{\mathsf{RO}})$*-hybrid model, assuming hash functions* $H_0, H_1, H_2, \tilde{H}$ *are modeled as random oracles, and assuming the hardness of DL on* $\mathbb{G}_1$ *and one-more unpredictability of* 2B–HashTDH*.*

Specifically, we show a simulator SIM *s.t. for any efficient adversary* $\mathcal{A}$ *against protocol* $\Pi =$ UC–2B–HashTDH *and any environment* $\mathcal{Z}$ *s.t.* $(\mathcal{Z}, \mathcal{A})$ *are* $(\mathbb{G}_1, \Pi)$*-algebraic, then* $(\mathcal{Z}, \mathsf{SIM})$ *are* $(\mathbb{G}_1, \mathcal{F}_{\mathsf{tOPRF}})$*-algebraic, and* $\mathcal{Z}$*'s advantage in distinguishing the view of* $\mathcal{A}$ *interacting with the real* UC–2B–HashTDH *protocol and the view of* SIM *interacting with the ideal functionality* $\mathcal{F}_{\mathsf{tOPRF}}$*, is upper-bounded by* $2q^2 \cdot \frac{m}{m-1} \cdot \epsilon_{DL} + \mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{R} \leftrightarrow \mathcal{Z}, \text{2B–HashTDH}}$*, where* q *is the number of* H_0 *queries,* $m = |\mathbb{G}_1|$*, and* ϵ_{DL} *and* $\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{R} \leftrightarrow \mathcal{Z}, \text{2B–HashTDH}}$ *are the advantages of algorithms (that run in time approximately equal to* $\mathcal{Z}$*) in the DL and* 2B–HashTDH *one-more unpredictability games, respectively.*

Proof. For any adversary $\mathcal{A}^*$, we construct simulator SIM as shown in Figures 7, 8, and 9. Without loss of generality, we assume that $\mathcal{A}^*$ is a "dummy" adversary that merely passes messages to and from the environment $\mathcal{Z}$.

First, note that simulator SIM is $\mathbb{G}_1$-algebraic. For simplicity of notation, we describe SIM's way of sampling random $\mathbb{G}_1$ elements in Fig. 9 using an unspecified sampling function, but this sampler can be trivially implemented in the algebraic way, by picking random exponent r and outputting $(g_1)^r$.

We now show that, for any efficient (i.e. PPT) $\mathcal{Z}$, the distinguishing advantage of $\mathcal{Z}$ between the real and simulated worlds is negligible. The argument proceeds by a series of game changes, starting from the real world $\mathcal{G}_0$ and ending at the simulated world $\mathcal{G}_4$. By $\mathbf{Dist}^{\mathcal{G},\mathcal{G}'}_{\mathcal{Z}}$ we denote distinguisher $\mathcal{Z}$'s distinguishing advantage between world $\mathcal{G}$ and world $\mathcal{G}'$. Specifically, $\mathbf{Dist}^{\mathcal{G},\mathcal{G}'}_{\mathcal{Z}} = |\Pr_{\mathcal{Z} \leftrightarrow \mathcal{G}}[\mathcal{Z} \to 1] - \Pr_{\mathcal{Z} \leftrightarrow \mathcal{G}'}[\mathcal{Z} \to 1]|$.

Game $\mathcal{G}_0$: The real world. The distinguisher $\mathcal{Z}$ interacts with UC–2B–HashTDH (Figure 6) in the role of the honest parties and in the role of the adversary. H_0, H_1, H_2 and $\tilde{H}$ are all true random oracles.

As a purely conceptual change from the true real world, one can imagine all the computational processes of the honest parties (i.e. UC–2B–HashTDH and $\mathcal{F}_{\mathsf{channel}}$) abstracted into a single monolithic component, which we call the simulator. This component also simulates the code of $\mathcal{F}_{\mathsf{tOPRF}}$ in parallel to its "real world" functions. By the end of the following sequence of game changes, all interactions with the honest parties will exclusively occur through the interface of $\mathcal{F}_{\mathsf{tOPRF}}$.

Game $\mathcal{G}_1$: $\tilde{H}$ returns PRF outputs. $\mathcal{G}_1$ is $\mathcal{G}_0$ with the following changes:

Notation

Notation is as in $\mathcal{F}_{\mathsf{tOPRF}}$. $\mathbb{G}_1$ is a group of prime order m with generator g_1. Initially, $\mathsf{evalset}_{sid}(\mathsf{ctx}, p) := \emptyset$ for all sid, ctx, and p. Values t, n, l are parameters.

Honest Initialization

1. On $(\mathsf{toprf.init}, sid, \mathsf{P}_0)$ from $\mathcal{F}_{\mathsf{tOPRF}}$ where $\mathsf{P}_0 \neq \mathcal{A}^*$:
   - pick $a_0, ..., a_t, b_1, \ldots, b_t, \hat{b}_1, \ldots, \hat{b}_t \leftarrow_{\$} \mathbb{Z}_m$ (such that a_0 has never been picked before)
   - define polynomials $k(i) := \sum_{\iota=0}^{t} a_\iota i^\iota$, $z(i) := \sum_{\iota=1}^{t} b_\iota i^\iota$, $\hat{z}(i) := \sum_{\iota=1}^{t} \hat{b}_\iota i^\iota$
   - save $(\mathsf{toprf.init}, sid, \mathsf{P}_0, (g_1)^{k(0)})$
   - for each $i \in [n]$, save $(\mathsf{toprf.share}, sid, i, \mathsf{P}_0, k(i), z(i), \hat{z}(i))$ marked INACTIVE
   - for each $i \in [n]$, send $(\mathsf{channel.send}, (sid, i), \mathsf{P}_0, \mathcal{S}_{sid}[i], |(i, k(i), z(i), \hat{z}(i))|)$ to $\mathcal{A}^*$
2. On $(\mathsf{channel.deliver}, (sid, i), \mathsf{P}_0, \mathcal{S}_{sid}[i])$ from $\mathcal{A}^*$ where $\exists$ record $(\mathsf{toprf.share}, sid, i, \mathsf{P}_0, k_i, z_i, \hat{z}_i)$ marked INACTIVE:
   - mark the record ACTIVE (or if $\mathsf{P}_0 = \mathcal{A}^*$ mark it COMPR)
   - send $(\mathsf{toprf.fininit}, sid, i)$ to $\mathcal{F}_{\mathsf{tOPRF}}$

Dishonest Initialization

3. On $(\mathsf{channel.send}, (sid, i), \mathcal{S}_{sid}[i], (i, k_i, z_i, \hat{z}_i))$ from $\mathcal{A}^*$ on behalf of $\mathsf{P}_0 \in \mathbf{Corr}$:
   - if this is the first such message for sid, then send $(\mathsf{toprf.init}, sid)$ to $\mathcal{F}_{\mathsf{tOPRF}}$ on behalf of P_0
   - save $(\mathsf{toprf.share}, sid, i, \mathcal{A}^*, k_i, z_i, \hat{z}_i)$ marked INACTIVE

Corruption

4. On corruption by $\mathcal{A}^*$ of party $\mathcal{P}$ (with permission from $\mathcal{Z}$):
   - send $(\mathsf{toprf.corrupt}, \mathsf{P})$ to $\mathcal{F}_{\mathsf{tOPRF}}$
   - set $\mathbf{Corr} := \mathbf{Corr} \cup \{\mathsf{P}\}$
5. If ever $\exists$ record $(\mathsf{toprf.share}, sid, i, \mathsf{P}_0, k_i, z_i, \hat{z}_i)$ marked ACTIVE where $\mathcal{S}_{sid}[i] \in \mathbf{Corr} - \{\mathcal{A}^*\}$, mark it COMPR and send $(\mathsf{toprf.share}, sid, i, k_i, z_i, \hat{z}_i)$ to $\mathcal{A}^*$

Fig. 7. Simulator SIM for protocol UC–2B–HashTDH, part 1: Notation, Honest Initialization, Corrupt Initialization, and Corruption

Honest Evaluation

6. On $(\mathsf{toprf.eval}, sid, ssid_{\mathsf{U}}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{tOPRF}}$ where $\mathsf{U} \neq \mathcal{A}^*$:
   - pick $r \leftarrow_{\$} \mathbb{Z}_m$ and $x' \leftarrow_{\$} \{0,1\}^{\lambda}$ and define $p := H_0(x')^r$
   - for each $i \in [n]$, send $(sid, i, ssid_{\mathsf{U}}, p)$ to $\mathcal{A}^*$ (addressed from U to $\mathcal{S}_{sid}[i]$)
   - and await responses $(sid, i, ssid_{\mathsf{U}}, q_i)$ from $\mathcal{A}^*$ (addressed from $\mathcal{S}_{sid}[i]$ to U) for all $i \in \mathbf{S}$, for any set $\mathbf{S} \subseteq [n]$ of size $t+1$;
   - then compute $q := \prod_{i \in \mathbf{S}} q_i^{\lambda_i}$ where λ_i is the Lagrange interpolation coefficient for index i and index set $\mathbf{S}$
   - if the algebraic representation of y is not of the form $y = H_0(x')^k$, then pick random $k \leftarrow_{\$} \mathbb{Z}_m$
   - run $\mathsf{FindEvalset}((g_1)^k)$, which returns $(sid^*, \mathsf{ctx}^*, \mathbf{S}')$ (see routine 9)
   - send $(\mathsf{toprf.rcvcomplete}, sid, ssid_{\mathsf{U}}, sid^*, \mathsf{ctx}^*, \mathbf{S}')$ to $\mathcal{F}_{\mathsf{tOPRF}}$
7. On $(\mathsf{toprf.sndrcomplete}, sid, i, \mathsf{ctx})$ from $\mathcal{F}_{\mathsf{tOPRF}}$ and $(sid, i, ssid'_{\mathsf{U}}, p)$ from $\mathcal{A}^*$ (addressed from U to $\mathcal{S}_{sid}[i]$) where $\mathcal{S}_{sid}[i] \neq \mathcal{A}^*$:
   - set $\mathsf{evalset}_{sid}(\mathsf{ctx}, p) := \mathsf{evalset}_{sid}(\mathsf{ctx}, p) \cup \{i\}$
   - retrieve $(\mathsf{toprf.share}, sid, i, \mathsf{P}_0, k_i, z_i, \hat{z}_i)$
   - compute $q_i := p^{k_i} \cdot H_1(\mathsf{ctx}, p)^{z_i} \cdot H_2(\mathsf{ctx}, p)^{\hat{z}_i}$
   - send $(sid, i, ssid'_{\mathsf{U}}, \mathsf{ctx}, q_i)$ to $\mathcal{A}^*$ (addressed from $\mathcal{S}_{sid}[i]$ to U)
8. Define subroutine $\mathsf{FindEvalset}(k^*)$:
   - find record $(\mathsf{toprf.init}, sid^*, \mathsf{P}_0, k^*)$
   - if no such record exists, create it as follows:
     - choose an unused sid^* such that $\mathcal{S}_{sid^*} = (\mathcal{A}^*)^n$
     - send $(\mathsf{toprf.init}, sid^*)$ to $\mathcal{F}_{\mathsf{tOPRF}}$
     - receive $(\mathsf{toprf.init}, sid^*, \mathcal{A}^*)$ in response
     - for each $i \in [n]$, send $(\mathsf{toprf.sinit}, sid^*, i, \mathcal{A}^*)$ and $(\mathsf{toprf.fininit}, sid^*, i)$ to $\mathcal{F}_{\mathsf{tOPRF}}$
     - save $(\mathsf{toprf.init}, sid^*, \mathcal{A}^*, k^*)$
   - if $\mathsf{P}_0 = \mathcal{A}^*$, then define $\mathbf{S}^* := [n]$
   - if $\mathsf{P}_0 \neq \mathcal{A}^*$, then define $\mathbf{S}^* := \{i : \exists$ record $(\mathsf{toprf.share}, sid^*, i, \mathsf{P}_0, k_i, z_i, \hat{z}_i)$ marked COMPR$\}$
   - for each $i \in \mathbf{S}^*$:
     - pick an unused $ssid'$
     - send $(\mathsf{toprf.sndrcomplete}, sid^*, i, ssid')$ to $\mathcal{F}_{\mathsf{tOPRF}}$
     - receive the same message in response (don't run routine 8)
   - pick any (ctx^*, p) such that $|\mathsf{evalset}_{sid^*}(\mathsf{ctx}^*, p)| \geq (t+1) - |\mathbf{S}^*|$ (if no such (ctx^*, p) exists, then emit (FAIL, sid^*) and halt the entire simulator)
   - pick any evaluation set $\mathbf{S}' \subseteq \mathbf{S}^* \cup \mathsf{evalset}_{sid^*}(\mathsf{ctx}^*, p)$ of size $|\mathbf{S}'| = t+1$
   - set $\mathsf{evalset}_{sid^*}(\mathsf{ctx}^*, p) := \emptyset$
   - return $(sid^*, \mathsf{ctx}^*, \mathbf{S}')$

Fig. 8. Simulator SIM for protocol UC–2B–HashTDH, part 2: Honest Evaluation

Dishonest Evaluation

9. On fresh query x to $H_0(\cdot)$, simply set $H_0(x) \leftarrow_\$ \mathbb{G}_1$ and return it
10. On fresh query (ctx, p) to $H_1(\cdot)$, simply set $H_1(\mathsf{ctx}, p) \leftarrow_\$ \mathbb{G}_1$ and return it
11. On fresh query (ctx, p) to $H_2(\cdot)$, simply set $H_2(\mathsf{ctx}, p) \leftarrow_\$ \mathbb{G}_1$ and return it
12. On fresh query (x, y) to $\tilde{H}(\cdot)$:
    - if the algebraic representation of y is not of the form $y = H_0(x)^k$, then simply set $\tilde{H}(x, y) \leftarrow \{0,1\}^*$ and return it
    - otherwise, run $\mathsf{FindEvalset}((g_1)^k)$, which returns $(sid^*, \mathsf{ctx}^*, \mathbf{S}')$ (see routine 9)
    - pick an unused $ssid_{\mathsf{U}}$ and send $(\mathsf{toprf.eval}, sid^*, ssid_{\mathsf{U}}, x)$ to $\mathcal{F}_{\mathsf{tOPRF}}$
    - send $(\mathsf{toprf.rcvcomplete}, sid^*, ssid_{\mathsf{U}}, sid^*, \mathsf{ctx}^*, \mathbf{S}')$ to $\mathcal{F}_{\mathsf{tOPRF}}$
    - receive $(\mathsf{toprf.eval}, ssid_{\mathsf{U}}, \tilde{y})$ in response
    - set $H_3(x, y) := \tilde{y}$ and return it

Fig. 9. Simulator SIM for protocol UC–2B–HashTDH, part 3: Dishonest Evaluation

- Subroutine FindEvalset is introduced (Figure 8). Given parameter $k^* \in \mathbb{G}_1$, this subroutine finds a PRF instance sid^* that was initialized with key $k(0)$ such that $k^* = (g_1)^{k(0)}$. If no such instance exists, it uses the interface of $\mathcal{F}_{\mathsf{tOPRF}}$ to create one, with $\mathcal{A}^*$ taking the nominal role of all n servers. In $\mathcal{G}_1$, FindEvalset does not yet search for a (ctx^*, p) pair under which a sid^* evaluation is legal. It therefore never fails and always outputs (only) sid^*.
- The simulator must answer $\tilde{H}$ queries that occur during honest user evaluation and that are made directly by the adversary. In both cases, $\tilde{H}$ uses FindEvalset to set their outputs (Figure 9). In particular, upon fresh query (x, y), $\tilde{H}$ checks whether y is algebraically represented as $H_0(x)^k$ for some $k \in \mathbb{Z}_m$. If not, then $\tilde{H}$ simply samples a random output. If so, then $\tilde{H}$ calls $\mathsf{FindEvalset}((g_1)^k)$, which returns sid^*. It then outputs $\tilde{y} := F_{sid^*}(x)$, where $F_{sid^*}(\cdot)$ is as defined in $\mathcal{F}_{\mathsf{tOPRF}}$ (Figure 4).

In the previous game, $\tilde{H}(x, y)$ outputs a randomly sampled value corresponding to (x, y). In this game, it outputs a randomly sampled value corresponding to (x, sid), where sid is a PRF instance identifier that FindEvalset maps to (x, y). In the case that y is not represented as $H_0(x)^k$ for some $k \in \mathbb{Z}_m$, sid is effectively a one-time value that is never used for any other (x, y). If there exists a perfect bijection between pairs (x, y) and pairs (x, sid), then $\mathcal{Z}$'s views of $\mathcal{G}_0$ and $\mathcal{G}_1$ are identical. Consider the two directions of this bijection:

1. Suppose there exist x, y, sid, and sid' such that $\tilde{H}(x, y)$ maps to both (x, sid) and (x, sid').
   $\tilde{H}$ only consults FindEvalset for fresh queries (x, y). Regardless of their representations, repeated (x, y) queries are guaranteed to produce the same output (i.e. map to the same sid). Therefore, $sid = sid'$.
2. Suppose there exist x, y, y', and sid such that $\tilde{H}(x, y)$ and $\tilde{H}(x, y')$ both map to (x, sid).

y and y' must be represented as $H_0(x)^k$ and $H_0(x)^{k'}$, respectively, for some $k, k' \in \mathbb{Z}_m$ (otherwise, they are guaranteed to map to different *sid*s). Also, $\mathsf{FindEvalset}((g_1)^k)$ and $\mathsf{FindEvalset}((g_1)^{k'})$ must both map to *sid*. Per the procedure of $\mathsf{FindEvalset}$, it follows that $k = k' = k(0)$, where $k(0)$ is the secret key for the PRF instance identified by *sid*. Therefore, $y = y'$.

Thus, this bijection between pairs (x, y) and pairs (x, sid) does indeed hold.

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathcal{G}_0,\mathcal{G}_1} = 0$$

Game $\mathcal{G}_2$: Honest user evaluation is oblivious. $\mathcal{G}_2$ is $\mathcal{G}_1$ with the following changes:

- After picking random exponent $r \leftarrow_\$ \mathbb{Z}_m$, the honest user evaluation procedure picks any new x' and sets $p := H_0(x')^r$ rather than $p := H_0(x)^r$ for the actual PRF input x. This ensures that the same x' is never used twice (and x' is never sent directly to $\tilde{H}$ by the adversary).
- Honest user evaluation proceeds as before, with the aggregation of q and computation of $y = q^{1/r}$. Then, however, the simulator checks whether y is algebraically represented as $H_0(x')^k$ for some $k \in \mathbb{Z}_m$. If not, it simply samples a random output $\tilde{y}$. If so, it calls $\mathsf{FindEvalset}((g_1)^k)$, which returns sid^*. It then outputs $F_{sid^*}(x)$ for the actual input x (i.e. not x').

In both $\mathcal{G}_2$ and $\mathcal{G}_1$, p is a uniformly random group element. In both $\mathcal{G}_2$ and $\mathcal{G}_1$, honest user evaluation leads to subroutine call $\mathsf{FindEvalset}((g_1)^k)$ where k is the discrete log $\mathsf{DL}_p(q)$. There is only one difference between $\mathcal{G}_1$ and $\mathcal{G}_2$ that may be visible to $\mathcal{Z}$: in $\mathcal{G}_2$, honest user evaluations do not contribute to $\tilde{H}$'s "cache" of previous queries.

Even still, as long as all y values (i.e. those appearing during honest user evaluation and those sent to $\tilde{H}$ by the adversary) have acceptable algebraic representations, the games will be identical to $\mathcal{Z}$. Thus, $\mathcal{Z}$'s probability of distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$ is upper-bounded by the probability that $\mathcal{Z}$ causes two evaluations (x, y) and (x', y') such that $y = H_0(x)^k$ and $y' = H_0(x')^k$ for some $k \in \mathbb{Z}_m$, and yet either y or y' is not algebraically represented as such. At least one of these (x, y) pairs must occur during honest user evaluation; the other may come from honest user evaluation or from a direct adversarial query to $\tilde{H}$. In either case, this event corresponds to a failure by the simulator to recognize that the two evaluations are using the same key (and should therefore be mapped to the same *sid*). We will denote this event E and prove by reduction to DL that its probability is negligible.

Given a DL input $\mathsf{challenge} = (g_1, (g_1)^\xi)$, we construct reduction $\mathcal{R}$, which uses $\mathcal{G}_2 \leftrightarrow \mathcal{Z}$ to solve for ξ. Denote by (x, y) and (x', y') the two evaluations that satisfy E. $\mathcal{R}$ begins by guessing $J, J' \leftarrow_\$ [q]$ (where the H_0 queries by $\mathcal{G}_2 \leftrightarrow \mathcal{Z}$ are denoted $x_1, \ldots, x_q$). $\mathcal{R}$ succeeds only if $x = x_J$ and $x' = x_{J'}$. For all $j \in [q]$, we denote $h_j := H_0(x_j)$.

Event E states that $y = (h_J)^k$ and $y' = (h_{J'})^k$ for some $k \in \mathbb{Z}_m$, yet either y or y' is not algebraically represented as such. Label the y and y' representations

as:

$$y = (g_1)^{c_0} \cdot \prod_{j \in [q]} (h_j)^{c_j}$$
$$y' = (g_1)^{c'_0} \cdot \prod_{j \in [q]} (h_j)^{c'_j}$$

H_1 and H_2 are also sources of $\mathbb{G}_1$ elements for $\mathcal{G}_2 \leftrightarrow \mathcal{Z}$. However, $\mathcal{R}$ simply programs those random oracles using group elements with known trapdoors (i.e. known discrete logarithms with respect to g_1); therefore any involvement of those group elements is included in c_0 and c'_0 in these representations.

Event E states that $c_0 \neq 0 \vee \exists_{j \in [q]-\{J\}}\ c_j \neq 0 \vee c'_0 \neq 0 \vee \exists_{j \in [q]-\{J'\}}\ c'_j \neq 0$.

Define $\gamma = \mathsf{DL}_{h_J}(h_{J'})$, i.e. $(h_J)^\gamma = h_{J'}$. Using γ, we can combine the representations of $y = (h_J)^k$ and $y' = (h_{J'})^k$ into one equation:

$$\begin{aligned}
1 &= (g_1)^{\gamma c_0 - c'_0} \cdot \prod_{j \in [q]} (h_j)^{\gamma c_j - c'_j} \\
&= (h_J)^{\gamma c_J - c'_J} \cdot (h_{J'})^{\gamma c_{J'} - c'_{J'}} \cdot (g_1)^{\gamma c_0 - c'_0} \cdot \prod_{j \in [q]-\{J,J'\}} (h_j)^{\gamma c_j - c'_j} \\
&= (h_J)^{\gamma^2 c_{J'} + \gamma(c_J - c'_{J'}) - c'_J} \cdot (g_1)^{\gamma c_0 - c'_0} \cdot \prod_{j \in [q]-\{J,J'\}} (h_j)^{\gamma c_j - c'_j}
\end{aligned}$$

Define F to be the event that some exponent in this equation is non-zero. Specifically, $\gamma^2 c_{J'} + \gamma(c_J - c'_{J'}) - c'_J \neq 0 \vee \gamma c_0 - c'_0 \neq 0 \vee \exists_{j \in [q]-\{J,J'\}}\ \gamma c_j - c'_j \neq 0$. $\mathcal{R}$ follows one of two reduction strategies uniformly at random. Strategy #1 succeeds if event F occurs, and strategy #2 succeeds if it doesn't. (Also, both strategies only succeed if event E occurs and J, J' are correct guesses.)

Strategy #1

This strategy follows the Relational DL approach (Definition 3). $\mathcal{R}$ picks $\gamma \leftarrow_\$ \mathbb{Z}_m$ and programs $h_{J'} := (h_J)^\gamma$. For all other $j \in [q] - \{J'\}$ (including $j = J$), $\mathcal{R}$ picks $a_j, b_j \leftarrow_\$ \mathbb{Z}_m$ and programs $h_j := (g_1)^{a_j} \cdot ((g_1)^\xi)^{b_j}$.

If E occurs, then $\mathcal{R}$ has values d_0, $\{d_j\}_{j \in [q]-\{J'\}}$ such that $(g_1)^{d_0} \cdot \prod_{j \in [q]-\{J'\}} (h_j)^{d_j} = 1$. In the exponent, we have the equation $d_0 + \sum_{j \in [q]-\{J'\}} (a_j + b_j \xi) d_j = 0$. We can isolate $\xi = (-d_0 - \sum_{j \in [q]-\{J'\}} a_j d_j)/(\sum_{j \in [q]-\{J'\}} b_j d_j)$. As long as the denominator $\sum_{j \in [q]-\{J'\}} b_j d_j \neq 0$, $\mathcal{R}$ solves DL.

If F occurs, $\exists_{j \in [q]-\{J'\}}\ d_j \neq 0$ (it is impossible that $d_0 \neq 0$ yet $\forall_{j \in [q]-\{J'\}}\ d_j = 0$). Notice that b_j is information-theoretically hidden from $\mathcal{G}_2 \leftrightarrow \mathcal{Z}$, i.e. h_j is independent of b_j. It follows by a one-time pad argument that $\sum_{j \in [q]-\{J'\}} b_j d_j$ is a uniformly random element of $\mathbb{Z}_m$ and $\Pr[\sum_{j \in [q]-\{J'\}} b_j d_j = 0] = \frac{1}{m}$. Therefore, strategy #1 solves DL with probability $1 - \frac{1}{m} = \frac{m-1}{m}$ if event E occurs, event F occurs, and J, J' are correct guesses.

Strategy #2

In this strategy, $\mathcal{R}$ picks $a \leftarrow_\$ \mathbb{Z}_m$ and programs $h_J := (g_1)^a$ and $h_{J'} := ((g_1)^\xi)^a$. The goal of $\mathcal{R}$ is to compute $\xi = \gamma$. For all other $j \in [q] - \{J, J'\}$, $\mathcal{R}$ randomly programs $h_j \leftarrow_\$ \mathbb{G}_1$.

If event F does not occur, then $\gamma^2 c_{J'} + \gamma(c_J - c'_{J'}) - c'_J = 0 \wedge \gamma c_0 - c'_0 = 0 \wedge \forall_{j \in [q]-\{J,J'\}}\ \gamma c_j - c'_j = 0$. If event E occurs, then either (a) $c_{J'} \neq 0 \vee c'_J \neq 0$, (b) $c_0 \neq 0 \vee c'_0 \neq 0$, or (c) $\exists_{j \in [q]-\{J,J'\}}\ c_j \neq 0 \vee c'_j \neq 0$. In case (a), the equation $\gamma^2 c_{J'} + \gamma(c_J - c'_{J'}) - c'_J = 0$ can be solved for γ. In case (b), the equation $\gamma c_0 - c'_0 = 0$ can be solved for γ. In case (c), the equation $\gamma c_j - c'_j = 0$ can be solved for γ. Therefore, strategy #2 solves DL with probability 1 if event E occurs, event F does not occur, and J, J' is a correct guess.

The probability of correctly guessing J, J' is $\geq \frac{1}{q^2}$ and the probability of correctly guessing whether or not F will occur is $\frac{1}{2}$. Overall, then, $\mathcal{R}$ solves DL with probability $\epsilon_{DL} \geq \Pr_{\mathcal{G}_2 \leftrightarrow \mathcal{Z}}[E] \cdot \frac{1}{q^2} \cdot \frac{1}{2} \cdot \frac{m-1}{m}$. As previously explained, $\mathcal{Z}$'s probability of distinguishing between $\mathcal{G}_1$ and $\mathcal{G}_2$ is upper-bounded by the probability of E.

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathcal{G}_1,\mathcal{G}_2} \leq 2q^2 \cdot \frac{m}{m-1} \cdot \epsilon_{DL}$$

Game $\mathcal{G}_3$: Illegal evaluations cause failure. $\mathcal{G}_3$ is $\mathcal{G}_2$ with the following changes:

- As honest servers perform evaluations (i.e. toprf.sndrcomplete) the evalset of each (ctx, p) pair is tracked (Figure 8).
- Subroutine FindEvalset is expanded. After finding sid^*, it "prints blank tickets" for as many of this instance's servers as it can (all n if the instance was just created); these blank tickets could correspond to any pair (ctx, p). Then it finds a pair (ctx^*, p) that has been evaluated by a set $\mathbf{S}'$ of at least $t + 1$ servers and "uses it up" by resetting its evalset to empty. If no such (ctx^*, p) can be found, then it is illegal to perform an evaluation on this PRF instance; the simulator immediately emits the event (FAIL, sid^*) and halts. Otherwise, the relevant instance identifier sid^*, server subsession identifier ctx^*, and evaluation set $\mathbf{S}'$ are returned. FindEvalset now fully matches its description in Figure 8.
- $\tilde{H}$ queries use $\mathcal{F}_{\mathsf{tOPRF}}$'s toprf.rcvcomplete interface to query the output $\tilde{y}$ (using the sid^*, $\mathbf{S}'$, and ctx^* from FindEvalset as parameters). $\tilde{H}$ now fully matches its description in Figure 9.
- Honest user evaluation also uses $\mathcal{F}_{\mathsf{tOPRF}}$'s toprf.rcvcomplete interface to query the output $\tilde{y}$ (using the sid^*, $\mathbf{S}'$, and ctx^* from FindEvalset as parameters). The honest user evaluation procedure now fully matches its description in Figure 8.

Observe that FindEvalset's mechanism for determining the legality of an evaluation is always at least as restrictive as the "ticketing" mechanism used by $\mathcal{F}_{\mathsf{tOPRF}}$ internally. $\mathcal{F}_{\mathsf{tOPRF}}$ allows an evaluation as long as every server in the

evaluation set has at least one unused ticket corresponding to ctx. FindEvalset only allows an evaluation if every server in the evaluation set has at least one unused ticket corresponding to ctx^* **and** the same common query p. Therefore, the interactions with toprf.rcvcomplete introduced in this game change are guaranteed to succeed (since they are always preceded by successful calls to FindEvalset).

Observe furthermore that the eventual evaluation output values $\tilde{y}$ are unaltered by this game change. Therefore, the only potential change to $\mathcal{Z}$'s view is the newly added possibility of the FAIL event, which causes all execution to immediately halt. $\mathcal{Z}$'s probability of distinguishing between $\mathcal{G}_2$ and $\mathcal{G}_3$ is upper-bounded by the probability that $\mathcal{Z}$ triggers FAIL. We will prove by reduction to the one-more unpredictability of 2B–HashTDH that the probability of FAIL is negligible.

We construct reduction $\mathcal{R}$, which is an adversary against $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathsf{2B–HashTDH}}$. $\mathcal{R}$ is $\mathcal{G}_3$ with the following changes:

- $\mathcal{R}$ begins by guessing the *sid* on which (FAIL, *sid*) will ultimately occur. Note that FAIL is only possible for honestly initialized *sid*s.
- On $(\mathsf{toprf.init}, sid, \mathsf{P}_0)$ from $\mathcal{F}_{\mathsf{tOPRF}}$, $\mathcal{R}$ does not pick secret keys. Instead, it responds using the input from $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathsf{2B–HashTDH}}$, including public key $(g_k)^k$.
- g_k is used in place of g_1 when saving public keys and when calling FindEvalset.
- $\mathcal{F}_{\mathsf{tOPRF}}$ corruption events to parties in $\mathcal{S}_{sid}$ are forwarded to the $\mathcal{O}^{\mathsf{Corrupt}}$ oracle of the $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathsf{2B–HashTDH}}$ game.
- $\mathcal{R}$ uses the $\mathcal{O}^{\mathsf{SEval}}$ oracle to respond to toprf.sndrcomplete messages targeting uncorrupted servers with *sid*.
- $\mathcal{R}$ saves a set $W = \{(x, y)\}$ of values corresponding to each FindEvalset execution that maps to *sid*. For FindEvalset calls originating from direct adversarial $\tilde{H}$ queries, these values are the input to $\tilde{H}$. For FindEvalset calls originating from honest user evaluation, the randomly chosen x' and the resulting $y = q^{1/r}$ are used.

The view of $\mathcal{Z}$ in this reduction is identical to its view in the game $\mathcal{G}_3$. The event FAIL exactly corresponds to the winning condition in the one-more unpredictability game. At the moment that FAIL occurs, the set W contains one more 2B–HashTDH evaluation pair than should be allowed. $\mathcal{R}$ outputs W and wins $\mathbf{Game}^{\mathsf{om-unpredictable}}_{\mathsf{2B–HashTDH}}$. $\mathcal{R} \leftrightarrow \mathcal{Z}$ is an algebraic algorithm that treats H_0, H_1, and H_2 as random oracles. Therefore, by Theorem 1, $\mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{R}\leftrightarrow\mathcal{Z},\mathsf{2B–HashTDH}}$ is negligible.

As previously explained, $\mathcal{Z}$'s probability of distinguishing between $\mathcal{G}_2$ and $\mathcal{G}_3$ is upper-bounded by the probability of FAIL.

$$\mathbf{Dist}^{\mathcal{G}_2,\mathcal{G}_3}_{\mathcal{Z}} \leq \mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{R}\leftrightarrow\mathcal{Z},\mathsf{2B–HashTDH}}$$

Game $\mathcal{G}_4$: The simulated world. The change from $\mathcal{G}_3$ to $\mathcal{G}_4$ is purely conceptual. All interactions between the simulator and the honest parties now

occur through the interface of $\mathcal{F}_{\mathsf{tOPRF}}$, so one can imagine the monolithic simulator now cleanly split into the two components $\mathcal{F}_{\mathsf{tOPRF}}$ and SIM.

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathcal{G}_3,\mathcal{G}_4} = 0$$

Summing up the distinguishing advantage bounds for each incremental game change yields an overall bound on $\mathcal{Z}$'s distinguishing advantage between the real and simulated worlds.

$$\mathbf{Dist}_{\mathcal{Z}}^{\mathcal{G}_0,\mathcal{G}_4} \leq 2q^2 \cdot \frac{m}{m-1} \cdot \epsilon_{DL} + \mathbf{Adv}^{\mathsf{om-unpredictable}}_{\mathcal{R}\leftrightarrow\mathcal{Z},\mathsf{2B\text{–}HashTDH}}$$

The distinguishing advantage of any efficient, algebraic $\mathcal{Z}$ between the real and simulated worlds is negligible. Thus, $\mathsf{UC\text{–}2B\text{–}HashTDH}$ realizes $\mathcal{F}_{\mathsf{tOPRF}}$.

□