A Fast and Simple Partially Oblivious PRF, with Applications

Nirvan Tyagi
Cornell University

Sofía Celi
Cloudflare

Thomas Ristenpart
Cornell Tech

Nick Sullivan
Cloudflare

Stefano Tessaro
University of Washington

Christopher A. Wood
Cloudflare

Abstract

We build the first construction of a partially oblivious pseudorandom function (POPRF) that does not rely on bilinear pairings. Our construction can be viewed as combining elements of the 2HashDH OPRF of Jarecki, Kiayias, and Krawczyk with the Dodis-Yampolskiy PRF. We analyze our POPRF's security in the random oracle model via reduction to a new one-more gap strong Diffie-Hellman inversion assumption. The most significant technical challenge is establishing confidence in the new assumption, which requires new proof techniques that enable us to show that its hardness is implied by the q-DL assumption in the algebraic group model.

Our new construction is as fast as the current, standards-track OPRF 2HashDH protocol, yet provides a new degree of flexibility useful in a variety of applications. We show how POPRFs can be used to prevent token hoarding attacks against Privacy Pass, reduce key management complexity in the OPAQUE password authenticated key exchange protocol, and ensure stronger security for password breach alerting services.

Contents

1 Introduction **3**

2 Preliminaries **6**
- 2.1 Algebraic Group Model 6
- 2.2 Random Oracle Model 6
- 2.3 Non-interactive Zero Knowledge Proofs 7

3 Partially Oblivious Pseudorandom Functions **8**

4 The 3HashSDHI POPRF **12**

5 Security Analysis **14**
- 5.1 Pseudorandomness 14
- 5.2 Request Privacy 19

6 Performance Evaluation **21**

7 Applications **21**
- 7.1 Privacy Pass 22
- 7.2 Private Set Membership and Breach Alerting 23
- 7.3 OPAQUE 23

A Security of (m, n)-OM-Gap-SDHI **27**
- A.1 Simulation of (m, n)-OM-Gap-SDHI environment 27
- A.2 Linear independence of winning elements 28

B Security Proofs for 3H **35**

C Security with Restricted Tag Space **37**

D Security of 2HashDH OPRF **38**

E (Partially) Blind Signatures Preliminaries **40**
- E.1 Syntax and Semantics 41
- E.2 Security 41

F (P)OPRFs from Unique (Partially) Blind Signatures **42**

G Security of ZSS Partially Blind Signature **43**

H Uniqueness from Request Privacy **47**

I Multiplicative Blinding **48**

1 Introduction

An oblivious pseudorandom function (OPRF) [FIPR05, JL09] allows a client holding a private input x and a server holding a key sk for a PRF f to engage in a protocol to *obliviously* evaluate f_{sk} on x. The client learns (and optionally verifies) the evaluation $f_{sk}(x)$ while the server learns nothing. *Partially*-oblivious PRFs (POPRF), first introduced by Everspaugh et al. in the context of the Pythia password hardening system [ECS+15], extend this functionality to include a public input (or metadata tag) t for the PRF evaluation. A client learns (and, optionally, verifies) $f_{sk}(t, x)$ where t is known by both server and client; the private input x remains hidden.

OPRFs are increasingly becoming a critical cryptographic tool for privacy-preserving protocols. Examples include one-time use anonymous credentials for spam prevention [DGS+18, HIJ+21], private set intersection (PSI) for checking compromised credentials [LPA+19, TPY+19], de-identified authenticated logging [HIJ+21], and password-authenticated key exchange [JKX18, JKK14]. In all these applications, we observe that there is a need to "partition" the PRF in a productive manner, i.e., allowing computation of $f_{sk}(t, x)$ using domain separation on some public value t. OPRF blinding protocols do not support this in a secure manner, because the server cannot verify what t is used within a client's oblivious request. Most OPRF applications therefore use a separate key instance for each t, with an associated increase in key management complexity. POPRFs directly provide this functionality, but the only known POPRF [ECS+15] relies on bilinear pairings, which slows performance relative to the best known OPRF and also complicates deployment given the lack of widespread implementation support for pairings.

In this work, we introduce a new POPRF that combines aspects of the 2HashDH OPRF of Jarecki et al. [JKK14], that is the de facto standard used in practice, with the Dodis-Yampolskiy (DY) verifiable random function [DY05]. Our POPRF is also closely related to a signature scheme suggested by Zhang, Safavi-Naini, and Susilo (ZSS) [ZSS04, ZSS03]. Our new POPRF, called 3HashSDHI, is essentially as performant as 2HashDH and does not rely on pairings, thereby enabling support for a public input virtually *for free*. While 3HashSDHI's protocol is simple, its analysis is not, requiring a new interactive discrete log (DL) assumption whose security we reduce to q-DL in the algebraic group model [FKL18]. We also provide new formal security notions for POPRFs and (as a special case) OPRFs, which we believe will be of independent interest.

Formal syntax and security notions for POPRFs. We start with the latter contribution. We provide a new formalization for POPRFs, including syntax, semantics, and security definitions. Our formal syntax builds off of [ECS+15] and previous OPRF formalizations [JL09, JKK14]. In terms of security, we propose new property-based security definitions that cover pseudorandomness (in the face of malicious clients) as well as request privacy and verifiability (in the face of malicious servers). Our property-based security games avoid the ideal function based formulations inherited from 2PC and used in prior works on OPRFs; they also avoid the non-standard "one-more" PRF security definition of [ECS+15].

Our *pseudorandomness* notion for POPRFs guarantees that the evaluation outputs look random to a malicious client, even when the malicious client has access to a blinded evaluation oracle. It is formalized with a simulation-based indistinguishability game that takes rough inspiration from the UC-style all-in-one OPRF security definition of [JKK14] and prior notions for partially blind signatures [AF96]. Here an adversary must distinguish between real evaluations of the PRF given access to a blind evaluation oracle, and evaluations of a random function given access to a simulated blind evaluation oracle. The simulator can receive random function evaluations on a limited number of points for any given public input t, where the limit is determined by the number of times the adversary has queried the blind evaluation oracle for that t. This restriction captures that only one random function evaluation is learned for each blind evaluation. Note that our accounting is more granular than the general "ticketing" approaches of blind UC protocols [JKK14, KZ08, Fis06]), due to the need of tying invocations to particular t values.

Our next notion is *request privacy* which captures that nothing about a client message x should leak to a malicious server during an oblivious evaluation, and, moreover, the server should not be able to link an output $f_{sk}(t, x)$ to particular oblivious request transcripts. The latter is often referred to as a linking attack, and is problematic in various applications of POPRFs. Our request privacy notion comes in two flavors, depending on whether the malicious server behaves passively or actively. The former allows us to analyze the privacy of schemes that do not allow verification that a server legitimately computed the blinded evaluation protocol; the latter requires schemes to allow client-side verification of the server's response.

Finally we formalize a notion of *uniqueness*. It ensures that a malicious server cannot trick clients into accepting inconsistent evaluations, relative to a shareable public key associated to the secret key sk.

The 3HashSDHI construction. The main contribution of this work is a new construction of a POPRF, which we call 3HashSDHI. The name refers to its use of three hashes and its reliance on the strong Diffie-Hellman inversion assumption. Its starting point is the 2HashDH construction of Jarecki et al. [JKK14], whose full PRF evaluation we define as $\mathsf{2HashDH.Ev}(sk, x) = \mathsf{H}_2(x, \mathsf{H}_1(x)^{sk})$. The blinded evaluation protocol has the client send $B = \mathsf{H}_1(x)^r$ for random r, and the server respond with $B' = B^{sk}$. The client can unblind to $(B')^{1/r} = \mathsf{H}_1(x)^{sk}$ in order to complete the evaluation of the function. Here operations are over a prime-order group (written multiplicatively) such as an elliptic curve. Proof of evaluation consists of a simple Chaum-Pedersen proof of discrete log equality [CP92] proving $\log_g pk = \log_B B'$ where $pk = g^{sk}$ is the server's public key. As mentioned, 2HashDH is already in use in practice [DGS+18, TPY+19, HIJ+21] and is on track to become a standard [DFHSW20].

We want a way to extend 2HashDH to allow public tags. To do so, we take inspiration from the Dodis-Yampolskiy PRF, whose evaluation is defined as $\mathsf{DY.Ev}(sk, t) = g^{1/(sk+t)}$. Put together, the 3HashSDHI scheme gives a PRF evaluated as:

$$\mathsf{3H.Ev}(sk, t, x) = \mathsf{H}_2\left(t, x, \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}\right).$$

It can therefore be interpreted as evaluating the Dodis-Yampolskiy PRF on the public input t over a random generator determined by the private input x, followed by a final hashing step. The basic structure of $\mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}$ was also described in an attempt to build secure partially blind signatures by ZSS [ZSS03]. Their analysis is incorrect, as we discuss further below and in Section 4.

To perform a blind evaluation, the client hashes and blinds their private input as $B = \mathsf{H}_1(x)^r$ using a random scalar r and sends B to the server holding sk. The server computes and sends back to the client the strong Diffie-Hellman inversion $B' = B^{1/(sk+\mathsf{H}_3(t))}$ of the blinded element using the secret key and public hash of the public input t. The client can unblind by computing $(B')^{1/r} = \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}$ and then complete the evaluation by hashing appropriately. To provide verifiability, the server uses a Chaum-Pedersen zero-knowledge proof (ZKP) of discrete log equality to prove $\log_g pk' = \log_{B'} B$ where $pk' = pk \cdot g^{\mathsf{H}_3(t)}$ which can be easily computed from public values by the client.

Our protocol incurs minimal overhead on top of the OPRF blind evaluation of 2HashDH, requiring only an extra hash computation, group operation, and scalar inversion. It makes use of the same Chaum-Pedersen proof for verifiability, which, as has been observed for 2HashDH, allows for evaluation of a batch of inputs whilst only constructing one Chaum-Pedersen proof [DGS+18, DFHSW20] (provided the batch is for the same public metadata tag t).

We formally show request privacy against passive adversaries (without ZKP) holds based just on the randomness of the blinding, and that request privacy against malicious adversaries holds additionally assuming the ZKP is sound. The key technical challenge is proving the new POPRF is pseudorandom.

As is seemingly requisite for schemes with blinded evaluation protocols, we prove the pseudorandomness security of our scheme with respect to a one-more gap style assumption [BNPS03, Bol03]. In fact the algebraic structure exposed to adversarial clients by the 3HashSDHI blinded evaluation protocol — raising an arbitrary group element Y to $1/(sk + \mathsf{H}_3(t))$ for adversarial t — requires new proof techniques compared to prior approaches. We start by introducing a new one-more gap strong Diffie-Hellman inversion (OM-Gap-SDHI) assumption, based on the perceived hardness of computing $Y^{1/(x+c)}$ for any base Y and (restricted) scalars c. We show via a relatively straightforward proof that this assumption is sufficient to prove POPRF pseudorandomness for 3HashSDHI, modeling the hash functions as random oracles. Additionally, the verifiable version requires that the ZKP is zero-knowledge.

The main difficulty is analyzing the security of our new computational assumption. In particular, for given distinct constants $c_1, \ldots, c_n$, the assumption considers a setting with an oracle SDH returning $B^{1/(x+c_i)}$ on input (B, i). Given some additional random group elements $Y_1, \ldots, Y_m$, it requires it to be hard to compute ℓ elements $Y_{i_1}^{1/(x+c_i)}, \ldots, Y_{i_\ell}^{1/(x+c_i)}$, for any $i \in [n]$ and for distinct $i_1, \ldots, i_\ell \in [m]$, using fewer than ℓ queries SDH$(\cdot, i)$. The challenge is that we do *not* restrict the number of queries SDH(X, j) for $j \neq i$, and this could be for group elements of X that depend on c_i (e.g., X is a prior output of an SDH$(\cdot, i)$ query). Ultimately, we show in the algebraic group model (AGM) [FKL18] that the assumption reduces to one of the uber assumptions from Bauer, Fuchsbauer, and Loss [BFL20], and therefore, in turn, is implied by the q-DL assumption, where q is a bound on the number of oracle queries. This AGM analysis implies hardness of the new assumption in the generic group model (GGM) [Sho97, Mau05].

In terms of concrete security, our analyses shows that, roughly speaking, 3HashSDHI is as hard as breaking the q-DL problem. Actually our main AGM proof is loose by a factor that is the maximum number of blind evaluation queries made by an adversary. Whether this AGM analysis can be tightened is an open question, but we observe in the body that a slight alternative to our AGM analysis gives a tight reduction in the GGM. We suggest using this tighter analysis to drive parameter selection: the best known attack against q-DL is due to Cheon [Che06] and indicates that a 256-bit group suffices for 80-bit security and a 384-bit group for 128-bit security. Importantly this matches the situation for 2HashDH, and so moving to 3HashSDHI does not require changing group parameters to achieve the desired security levels.

Partially-blind signatures. Our techniques provide a new approach to building partially-blind signatures [AF96]. Whereas an OPRF requires access to the private key to verify a given input, a blind signature protocol only requires the public key. This property is useful for a number of applications and deployment settings. For example, in settings where multiple instances of a verifier may check the output of the OPRF, each instance would either (a) require access to the private key or (b) request verification from an entity which holds the private key. The former may be problematic if instances that verify outputs do not mutually trust one another or cannot otherwise share private key material, and the latter may be problematic because it incurs a network performance penalty. Blind signatures avoid both problems by allowing each instance to use the public key for verification.

Blind signatures are used in one-time use anonymous credentials, and are also being proposed as a tool for private click measurement (PCM) in the W3C [WTKW20]. One limitation in these use cases is that the protocols do not admit public metadata in the signature computation. PCM, for example, would benefit from binding additional context to signature computations [WTKW20].

As previously mentioned, the 3HashSDHI construction is closely related to the ZSS partially blind signature scheme [ZSS03], which uses pairings. As we explain in Section 4, the original unforgeability proof is however incorrect. We provide the first (correct) formal analysis of the security of ZSS using our new techniques in Appendices E and G. To the best of our knowledge, this result provides the most efficient partially-blind signature supporting arbitrary public metadata; previous RSA-based constructions [AF96, AO00] require the set of public metadata tags to be incorporated during parameter setup, previous Schnorr-based constructions [AO00, FPS20] are vulnerable in the concurrent signing setting [BLL+21], and other existing constructions are more heavyweight as they are tailored for the anonymous credential setting [CL04].

Finally, we also show how any unique (partially) blind signature scheme can be used to generically construct a POPRF by hashing the signature using a random oracle. This is apparently a folklore result for OPRFs, and we are unaware of any formal treatment it. We provide one that also covers partial obliviousness/blindness. See Appendix F.

Applications of our POPRF. Equipped with our new POPRF and the underlying design of 3HashSDHI, we return to our motivating applications and show how swapping in a POPRF for the existing OPRF can lead to various benefits for deployments.

One-time use anonymous credentials. Privacy Pass [DGS+18, CDFH21] is a protocol in which clients may be issued one-time use tokens that can later be redeemed anonymously to authenticate themselves. It has been proposed for use in the context of content distribution networks and web advertising, requiring users to authenticate with a token, and thereby reducing malicious web requests, protecting against, e.g., denial-of-service attacks and fraudulent advertisement conversions. Tokens are issued to users that prove trustworthiness, e.g., through a CAPTCHA challenge, The protocol is being considered for standardization by both the IETF and the World Wide Web Consortium (W3C), and a prototype deployment is already in production use by Cloudflare, hCaptcha, and others.

An OPRF is the core component of the protocol. Tokens are issued via an OPRF in which users obtain evaluations at random points, storing the point x and evaluation y. Redeeming a token simply involves showing the pair (x, y), which the server can check is valid, but cannot link x back to an issuance due to the oblivious evaluation. The server stores a strikelist of used tokens to prevent double spending. Additionally, all servers perform a global double-spend check to avoid clients from exploiting the possibility of spending tokens more than once against distributed token checking systems. The use of an OPRF leads to a more efficient issuance protocol than alternate approaches for keyed-verification anonymous credentials that support attributes and proofs over attributes [CMZ14, CPZ20].

An abuse of the protocol that has been observed in its early use is individual users (or groups of users) gathering tokens over a long period of time and redeeming them all at once, e.g., in an attempt to overwhelm a website. We refer to such behavior as a hoarding attack. A conceptually easy way to mitigate the damage of a hoarding attack is to expire old unspent tokens after an amount of time: the way to do this with an OPRF is by rotating the OPRF key. But key rotations are complex, limiting their frequency: establishing trust in a frequently-rotating key is a challenging problem. Trustworthy keys are important in this context, as a server that equivocates on their public key can link token issuances and redemptions, by, for example, using a unique public key for each issuance. As we show, POPRFs address the issue of expiring tokens without the need of rotating keys by using the public metadata input to encode an expiration epoch.

Bucketized PSI for checking compromised credentials. Password breach alerting protocols [TPY⁺19, LPA⁺19] allow a user to query to determine if their username, password pair (u, pw) has appeared in a dataset D of known breaches. If so, the user is vulnerable to credential stuffing attacks and should change their password. Current services for breach alerting rely on an ad hoc 2HashDH-based private-set membership protocol that achieves scalability via bucketization: the user sends a truncated hash $\mathsf{H}(u)$ of their username to identify a subset $B \subseteq D$ that have matching truncated username hash. A 2HashDH-based protocol is then performed over B: the client obliviously evaluates $\mathsf{2HashDH.Ev}(sk, u \parallel pw)$ with sk held by server, and also obtains the OPRF outputs for all the values in the bucket B. Bucketization ensures scalability by limiting $|B|$ despite $|D|$ being on the order of billions of username, password pairs.

One issue is that currently deployed protocols provide no cryptographic binding between the bucket identifier $\mathsf{H}(u)$ and the blinded OPRF output: a malicious client can query for arbitrary usernames, not just ones that match $\mathsf{H}(u)$. Whether this is a significant security problem in practice is not clear, but we note that POPRFs easily rectify it by replacing 2HashDH above with 3HashSDHI and setting $t = \mathsf{H}(u)$.

Asymmetric password-authenticated key exchange. Password authenticated key exchange (PAKE) protocols [BM93] allow a client and server to establish a shared session key authenticated by a short password. Strong asymmetric PAKE (SaPAKE) protocols [JKX18] additionally ensure that the server can store (just) what amount to salted hashes of user passwords, thereby making it so that PAKEs can achieve the same level of security achieved by standard password-based authentication in the case of a server breach. The OPAQUE [JKX18] SaPAKE protocol uses an OPRF as one of its core components; it is currently being considered for standardization by the IETF [KLW21]. The OPRF suggested for use is 2HashDH.

In OPAQUE the server uses a separate OPRF key for each user. We show how we can instead use our 3HashSDHI POPRF to allow OPAQUE to work with a single master key pk; diversity across users can then be provided using usernames as the public input t to 3HashSDHI. We believe that this will simplify deployments and potentially improve their security, as discussed in the body.

2 Preliminaries

2.1 Algebraic Group Model

In some of our security proofs, we consider security against *algebraic* adversaries which we model using the algebraic group model, following the treatment of [FKL18]. We call an algorithm $\mathcal{A}$ *algebraic* if for all group elements Z that are output (either as final output or as input to oracles), $\mathcal{A}$ additionally provides the representation of Z relative to all previously received group elements. The previous received group elements include both original inputs to the algorithm and outputs received from calls to oracles. More specifically, if $[X]_i$ is the list of group elements $[X_0, \dots, X_n] \in \mathbb{G}$ that $\mathcal{A}$ has received so far, then, when producing group element Z, $\mathcal{A}$ must also provide a list $[z]_i = [z_0, \dots, z_n]$ such that $Z = \prod_i X_i^{z_i}$.

2.2 Random Oracle Model

We will prove security using ideal primitives, modeling hash functions as random oracles. Since our schemes will make use of more than one hash function, it will be useful to have a general abstraction for the use of ideal primitives, following the treatment of [JT20]. An ideal primitive P specifies algorithms $\mathsf{P.Init}$ and $\mathsf{P.Eval}$. The initialization algorithm has syntax $st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(1^{\lambda})$. The stateful evaluation algorithm has

```
Game $\textsc{Sound}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P}}(\lambda)$
    $pp \leftarrow\!\!\$\ \mathsf{NiZK.Setup}(\lambda)$
    $st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
    $(x, \pi) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{P}}(pp)$
    Return
    $\bigwedge \begin{pmatrix} \mathsf{NiZK.Ver}^{\mathsf{P}}(x, \pi) \\ \nexists w \ :\ (x, w) \in \mathcal{R} \end{pmatrix}$

Game $\mathrm{ZK}^{\mathcal{A},b}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda)$
    $pp \leftarrow\!\!\$\ \mathsf{NiZK.Setup}(\lambda)$
    $st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
    $st_{\mathsf{S}} \leftarrow\!\!\$\ \mathsf{S.Init}(pp)$
    $b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prim},\textsc{Prove}}(pp)$
    Return $b'$

Oracle $\textsc{Prove}(x, w)$
    Require $(x, w) \in \mathcal{R}$
    $\pi_1 \leftarrow\!\!\$\ \mathsf{NiZK.Prove}^{\mathsf{P}}(x, w)$
    $\pi_0 \leftarrow\!\!\$\ \mathsf{S.Prove}(x : st_{\mathsf{S}})$
    Return $\pi_b$

Oracle $\textsc{Prim}(x)$
    $y_1 \leftarrow\!\!\$\ \mathsf{P.Eval}(x : st_{\mathsf{P}})$
    $y_0 \leftarrow\!\!\$\ \mathsf{S.Eval}(x : st_{\mathsf{S}})$
    Return $y_b$
```

Figure 1: Soundness (left) and zero knowledge (right) security games for non-interactive zero knowledge proof systems.

syntax $y \leftarrow\!\!\$\ \mathsf{P.Eval}(x : st_{\mathsf{P}})$. We sometimes use A^{P} as shorthand for giving algorithm A oracle access to $\mathsf{P.Eval}(\cdot : st_{\mathsf{P}})$. While, the stateful formulation of the ideal primitive is used to allow for efficient instantiation in our security proofs, e.g., by "lazy sampling", ideal primitives should be *essentially stateless* [JT20] to prevent contrived behavior. For example, a random oracle can be written to be stateless, but it would inefficient to have to store a huge random table. We can combine access to multiple ideal primitives primitives $\mathsf{P} = \mathsf{P}_1 \times \ldots \times \mathsf{P}_m$ as follows:

```
$\mathsf{P.Init}(1^\lambda)$
    $[st_{\mathsf{P},i}]_i^m \leftarrow\!\!\$\ \left[\mathsf{P}_i.\mathsf{Init}(1^\lambda)\right]_i^m$
    Return $[st_{\mathsf{P},i}]_i^m$

$\mathsf{P.Eval}(x : [st_{\mathsf{P},i}]_i^m)$
    $(i, x) \leftarrow x$
    $y \leftarrow\!\!\$\ \mathsf{P}_i.\mathsf{Eval}(x : st_{\mathsf{P},i})$
    Return $y$
```

To concretize the above, we focus on random oracles. We define a random oracle that takes arbitrary input and produces random output from a sampling algorithm Samp. It is captured by the ideal primitive $\mathsf{RO}[\mathsf{Samp}] = (\mathsf{RO.Init}, \mathsf{RO.H})$ defined as follows. When the range is clear from context, Samp may be omitted.

```
$\mathsf{RO.Init}(1^\lambda)$
    $T \leftarrow [\cdot]$
    Return $T$

$\mathsf{RO.Eval}(x : T)$
    If $x \notin T$ then $T[x] \leftarrow\!\!\$\ \mathsf{Samp}()$
    Return $T[x]$
```

When clear from context and in an abuse of notation (since we will use H_i to denote a hash function as well), we will write $\mathsf{P} = \mathsf{H}_1 \times \cdots \times \mathsf{H}_m$ as the ideal primitive that gives access to m random oracles, accessible by querying directly an oracle labeled H_i.

Algebraic algorithms in the random oracle model. As in [FPS20], to support algebraic algorithms, we will require the structure of the domain and range to be specified for any random oracle RO. We assume an input can be efficiently checked to be a valid member of the domain and perform such checks implicitly returning $\bot$ if they fail. We will require that algebraic algorithms provide representations for any group element input, specified as part of the domain of RO. And similarly, any group element output of RO is included in the list of received group elements for the algebraic adversary.

2.3 Non-interactive Zero Knowledge Proofs

We define a non-interactive proof system NiZK over an efficiently computable relation $\mathcal{R}$ defined over pairs (x, w) where x is called the *statement* and w is called the *witness*. It is made up of the following algorithms. The setup algorithm produces the public parameters for execution, $pp \leftarrow\!\!\$\ \mathsf{NiZK.Setup}(\lambda)$. The proving algorithm takes a witness and statement and produces a proof, $\pi \leftarrow\!\!\$\ \mathsf{NiZK.Prove}^{\mathsf{P}}_{pp}(w, x)$. The verification algorithm verifies the proof for a statement, $b \leftarrow \mathsf{NiZK.Ver}^{\mathsf{P}}_{pp}(x, \pi)$. We define the following security properties.

Completeness. A proof system is *complete* if given a true statement, a prover with a witness can convince the verifier. We will make use of a proof system with perfect completeness. A proof system has *perfect completeness* if for all $(x, w) \in \mathcal{R}$,

$$\Pr\left[\mathsf{NiZK.Ver}^{\mathsf{P}}_{pp}(x, \mathsf{NiZK.Prove}^{\mathsf{P}}_{pp}(w, x)) = 1\right] = 1\ .$$

Knowledge soundness. A proof system is computationally *knowledge sound* if whenever a prover is able to produce a valid proof for a statement x, it is a true statement, i.e., there exists some witness w such that

```
$\Sigma_{\mathcal{R}}.\mathsf{Prove}^{\mathsf{H}}(\alpha, (g, U, V, W))$
$r \leftarrow\!\!\$\ \mathbb{Z}_p$
$s_U \leftarrow g^r$ ; $s_W \leftarrow V^r$
$c \leftarrow \mathsf{H}(g \parallel U \parallel V \parallel W \parallel s_U \parallel s_W)$
$z \leftarrow r - c\alpha$
$\pi \leftarrow (z, c)$
Return $\pi$
```

```
$\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}}((g, U, V, W), \pi)$
$(z, c) \leftarrow \pi$
$s_U \leftarrow g^z U^c$ ; $s_W \leftarrow V^z W^c$
Return $c = \mathsf{H}(g \parallel U \parallel V \parallel W \parallel s_U \parallel s_W)$

$\mathcal{R} = \{(\alpha), (g, U, V, W) : U = g^\alpha \wedge W = V^\alpha\}$
```

Figure 2: Description of Chaum-Pedersen discrete log equality Sigma protocol [CP92].

$(x, w) \in \mathcal{R}$. Knowledge soundness is defined by the security game $\text{SOUND}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P}}(\lambda)$ (Figure 1) in which an adversary is tasked with finding a verifying statement and proof where the statement is not in $\mathcal{R}$. The advantage of an adversary is defined as $\mathsf{Adv}^{\text{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P},\mathcal{A}}(\lambda) = \Pr[\text{SOUND}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P}}(\lambda) = 1]$ with respect to ideal primitive P.

Zero knowledge. A proof system is computationally *zero-knowledge* if a proof does not leak any information besides the truth of a statement. Zero knowledge is defined by the security game $\text{ZK}^{\mathcal{A},b}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda)$ (Figure 1) in which an adversary is tasked with distinguishing between proofs generated from a valid witness and simulated proofs generated without a witness. The advantage of an adversary is defined as

$$\mathsf{Adv}^{\text{zk}}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P},\mathcal{A}}(\lambda) = \left| \Pr[\text{ZK}^{\mathcal{A},1}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda) = 1] - \Pr[\text{ZK}^{\mathcal{A},0}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda) = 1] \right| ,$$

with respect to simulator algorithm S and ideal primitive P.

Fiat-Shamir heuristic for Sigma protocols. Our protocol requires a non-interactive zero knowledge proof for the relation including two pairs of group elements with equivalent discrete logs:

$$\mathcal{R} = \{(g, U, V, W), (\alpha) : U = g^\alpha \wedge W = V^\alpha\} .$$

This relation falls into a general family of relations of discrete log linear homomorphisms for which there exist so-called "Sigma protocols" [Cam98] to construct interactive proofs of knowledge. These can be made non-interactive using the Fiat-Shamir heuristic in the standard way. We denote $\Sigma_{\mathcal{R}}[\mathsf{GGen}]$ (shortened to $\Sigma_{\mathcal{R}}$ for simplicity) as the resulting non-interactive proof system for $\mathcal{R}$ known as the Chaum-Pedersen protocol [CP92] (shown in Figure 2); it is perfectly complete, computationally sound, and perfectly zero-knowledge in the random oracle model. We refer readers to [BS17, Figure 19.7] for construction of a simulator S_Σ, which leads to the following well-known result that we state for completeness:

Theorem 1 *The simulator S_Σ is such that for any RO-model adversary $\mathcal{A}_{\text{zk}}$ against* ZK *of $\Sigma_{\mathcal{R}}$,*

$$\mathsf{Adv}^{\text{zk}}_{\Sigma_{\mathcal{R}},\mathcal{R},\mathsf{S}_\Sigma,\mathsf{RO},\mathcal{A}_{\text{zk}}}(\lambda) \le q_{\mathsf{P}} \cdot (q_{\mathsf{P}} + q_{\mathsf{H}})/2^\lambda ,$$

where $\mathcal{A}_{\text{zk}}$ makes at most q_{P} and q_{H} queries to PROVE *and the random oracle $\mathsf{RO} : \mathbb{G}^6 \to \mathbb{Z}_p$.*

3 Partially Oblivious Pseudorandom Functions

We provide a new formalization for POPRFs, including syntax, semantics, and security. Our formalization builds off that from [ECS+15], but we offer new security notions that cover simulation-based security as a PRF (in the presence of a blinded evaluation oracle), client input privacy, and verifiability.

Syntax and semantics. A partially-oblivious pseudorandom function (POPRF) scheme, Fn, is a tuple of algorithms

$$(\mathsf{Fn.Setup}, \mathsf{Fn.KeyGen}, \mathsf{Fn.Req}, \mathsf{Fn.BlindEv}, \mathsf{Fn.Finalize}, \mathsf{Fn.Ev}) .$$

The setup and key generation algorithm generate public parameters *pp* and a public key, secret key pair (pk, sk), respectively. Oblivious evaluation is carried out as an interactive protocol run between client and server. The protocols we consider in this work make use of only a single round of interaction, so we simplify the syntax of the interactive oblivious evaluation protocol into algorithms ($\mathsf{Fn.Req}$, $\mathsf{Fn.BlindEv}$, $\mathsf{Fn.Finalize}$) that work as follows:

(1) First, a client runs the algorithm $\mathsf{Fn.Req}^{\mathsf{P}}_{pp}(pk, t, x)$, which takes input a public key pk, tag (or public input) t, and private input x, and outputs a local state st and a request message req. The message req is sent to a server.

(2) A server runs algorithm $\mathsf{Fn.BlindEv}^{\mathsf{P}}_{pp}(sk, t, req)$, using as input a secret key, a tag t, and the request message. It produces a response message rep that should be sent back to the client.

(3) Finally, the client runs the algorithm $\mathsf{Fn.Finalize}(rep : st)$ and outputs a PRF evaluation or $\perp$ if the response message is rejected, for example, due to the verification check failing.

The unblinded evaluation algorithm $\mathsf{Fn.Ev}$ is deterministic, and takes as input a public key, secret key pair (pk, sk), an input pair (t, x), and outputs a PRF evaluation y. We also define sets $\mathsf{Fn.SK}$, $\mathsf{Fn.PK}$, $\mathsf{Fn.T}$, $\mathsf{Fn.X}$, and $\mathsf{Fn.Out}$ representing the secret key, public key, tag, private input, and output space, respectively. We define the input space $\mathsf{Fn.In} = \mathsf{Fn.T} \times \mathsf{Fn.X}$. We assume efficient algorithms for sampling and membership queries on these sets. When it is clear from context, we drop the prefix Fn and subscript pp from algorithm names.

For correctness, we require that Ev is a function, and that the blinded and unblinded evaluations are consistent. To formalize the latter: we require that for any pp output from Setup, any pk, sk output by KeyGen, and any t, x, it holds that $\Pr[\mathsf{Ev}(sk, t, x) = y] = 1$ where the probability is taken over choice of y via the following process:

$$(st, req) \leftarrow\!\!{\scriptstyle\$}\ \mathsf{Req}^{\mathsf{P}}(pk, t, x)\ ;\ rep \leftarrow\!\!{\scriptstyle\$}\ \mathsf{BlindEv}^{\mathsf{P}}(sk, t, req)\ ;\ y \leftarrow\!\!{\scriptstyle\$}\ \mathsf{Finalize}^{\mathsf{P}}(rep : st)\ .$$

Security. We introduce three new security definitions for POPRFs. We use code-based games mostly following the framework of Bellare and Rogaway [BR06].

Pseudorandomness. The first definition captures pseudorandomness, i.e., indistinguishability of the POPRF from a random function, even for malicious clients that have access to a blinded evaluation oracle. We borrow some elements from the UC definition for standard OPRFs from [JKK14], but opt for what we believe to be a simpler, standalone formulation. We also extend to handle partial obliviousness, which has some subtleties.

A pseudocode game appears in Figure 3. The game is parameterized by a security parameter λ, an adversary $\mathcal{A}$, a challenge bit b, a POPRF Fn, a simulator $\mathsf{S} = (\mathsf{S.Init}, \mathsf{S.BlindEv}, \mathsf{S.Eval})$, and an ideal primitive P. The last will be used for random oracles in our main result. A simulator is a triple of algorithms that share state (explicitly denoted by st_{S} in the game). Algorithm $\mathsf{S.Init}$ initializes the simulator state and outputs a public key for the game. Algorithm $\mathsf{S.BlindEv}$ simulates blinded evaluation response messages while $\mathsf{S.Eval}$ simulates random oracle queries. Importantly, $\mathsf{S.BlindEv}$ and $\mathsf{S.Eval}$ can obtain $\textsc{Ev}$ outputs, but they can only do so in a circumscribed way: the simulator has oracle access to $\textsc{LimEv}$ which limits the number of full evaluations it can obtain to be at most the number of queries so far made by the adversary to the $\textsc{BlindEv}$. Importantly, this limit is per-metadata value t (indicated via the subscript): the $\textsc{LimEv}$ query on any particular t is bound by the total number of blinded evaluation queries on that particular t. This follows from similar granular restrictions in the partially blind signatures literature [AF96].

A weaker version of the game would simply cap the total number of queries to $\textsc{LimEv}$ by the total number of queries to $\textsc{BlindEv}$. This notion is, however, too weak for applications because we would like to ensure that querying, say, three times on public input t_1 cannot somehow help an adversary complete the evaluation for another public input $t_2 \neq t_1$. We note that a recent preprint [SS21] contained this weaker notion, couched in the context of Privacy Pass. (We discuss this paper further in Section 4.)

We let the advantage of a POPRF adversary $\mathcal{A}$ be defined by

$$\mathsf{Adv}^{\text{po-prf}}_{\mathsf{Fn},\mathsf{S},\mathsf{P},\mathcal{A},}(\lambda) = \left|\Pr\left[\,\mathrm{POPRF}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{S},\mathsf{P}}(\lambda) \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{POPRF}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{S},\mathsf{P}}(\lambda) \Rightarrow 1\,\right]\right|$$

where the probability spaces are taken over the random choices made in the games and the events signify that the game outputs the value one.

One could relax our definition in various ways. For example, by setting a parameter $q_{t,\max}$ that upper bounds the total number of $\textsc{BlindEv}$ queries on tag t over the course of the game and letting the simulator — at any point in the game — obtain $q_{t,\max}$ full evaluations. This would seem to still provide qualitatively the same level of security, but our schemes meet the stronger notion that restricts the simulator over the course of the game. Another relaxation that does not preserve the same level of security would be to allow

```
Game $\mathrm{POPRF}^{\mathcal{A},b}_{\mathsf{Fn},\mathsf{S},\mathsf{P}}(\lambda)$
$\mathsf{RandFn} \leftarrow\!\!\$\ \mathsf{FnGen}(\mathsf{Fn.In}, \mathsf{Fn.Out})$
$st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
$pp \leftarrow\!\!\$\ \mathsf{Fn.Setup}(\lambda)$
$(sk, pk_1) \leftarrow\!\!\$\ \mathsf{Fn.KeyGen}^{\mathsf{P}}_{pp}()$
$(st_{\mathsf{S}}, pk_0) \leftarrow\!\!\$\ \mathsf{S.Init}(pp)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Ev},\textsc{BlindEv},\textsc{Prim}}(pp, pk_b)$
Return $b'$
```

```
Oracle $\textsc{Ev}(t, x)$
$y_1 \leftarrow \mathsf{Fn.Ev}^{\mathsf{P}}(sk, t, x)$
$y_0 \leftarrow \mathsf{RandFn}(t, x)$
Return $y_b$

Oracle $\textsc{LimEv}(t, x)$
$q_{t,s} \leftarrow q_{t,s} + 1$
If $q_{t,s} \leq q_t$ then
    Return $\textsc{Ev}(t, x)$
Return $\perp$
```

```
Oracle $\textsc{BlindEv}(t, req)$
$q_t \leftarrow q_t + 1$
$rep_1 \leftarrow \mathsf{Fn.BlindEv}^{\mathsf{P}}(sk, t, req)$
$(rep_0, st_{\mathsf{S}}) \leftarrow\!\!\$\ \mathsf{S.BlindEv}^{\textsc{LimEv}}(t, req : st_{\mathsf{S}})$
Return $rep_b$

Oracle $\textsc{Prim}(x)$
$y_1 \leftarrow\!\!\$\ \mathsf{P.Eval}(x : st_{\mathsf{P}})$
$(y_0, st_{\mathsf{S}}) \leftarrow\!\!\$\ \mathsf{S.Eval}^{\textsc{LimEv}}(x : st_{\mathsf{S}})$
Return $y_b$
```

Figure 3: Simulation-based security definition for pseudorandomness against malicious clients, with granular accounting for metadata in queries. The LimEval oracle limits the number of evaluations the simulator can make on a per-metadata tag basis.

```
Game $\mathrm{POPRIV1}^{\mathcal{A},b}_{\mathsf{Fn},\mathsf{P}}(\lambda)$
$pp \leftarrow\!\!\$\ \mathsf{Fn.Setup}(\lambda)$
$(pk, sk) \leftarrow\!\!\$\ \mathsf{Fn.KeyGen}(pp)$
$st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Trans},\mathsf{P}}(pp, pk, sk)$
Return $b'$

Oracle $\textsc{Trans}(t, x_0, x_1)$
$(st_0, req_0) \leftarrow\!\!\$\ \mathsf{Fn.Req}^{\mathsf{P}}(pk, t, x_0)$
$(st_1, req_1) \leftarrow\!\!\$\ \mathsf{Fn.Req}^{\mathsf{P}}(pk, t, x_1)$
$rep_0 \leftarrow\!\!\$\ \mathsf{Fn.BlindEv}^{\mathsf{P}}(sk, t, req_0)$
$rep_1 \leftarrow\!\!\$\ \mathsf{Fn.BlindEv}^{\mathsf{P}}(sk, t, req_1)$
$y_0 \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(rep_0; st_0)$
$y_1 \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(rep_1; st_1)$
$\tau \leftarrow (req_b, rep_b, y_0)$
$\tau' \leftarrow (req_{1-b}, rep_{1-b}, y_1)$
Return $(\tau, \tau')$
```

```
Game $\mathrm{POPRIV2}^{\mathcal{A},b}_{\mathsf{Fn},\mathsf{P}}(\lambda)$
$pp \leftarrow\!\!\$\ \mathsf{Fn.Setup}(\lambda)$
$st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
$i \leftarrow 0$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Req},\textsc{Fin},\mathsf{P}}(pp)$
Return $b'$

Oracle $\textsc{Req}(pk, t, x_0, x_1)$
$i \leftarrow i + 1$
$(st_{i,0}, req_0) \leftarrow\!\!\$\ \mathsf{Fn.Req}^{\mathsf{P}}(pk, t, x_0)$
$(st_{i,1}, req_1) \leftarrow\!\!\$\ \mathsf{Fn.Req}^{\mathsf{P}}(pk, t, x_1)$
Return $(req_b, req_{1-b})$

Oracle $\textsc{Fin}(j, rep, rep')$
If $j > i$ then return $\perp$
$y_b \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(st_{j,b}, rep)$
$y_{1-b} \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(st_{j,1-b}, rep')$
If $y_0 = \perp$ or $y_1 = \perp$ then
    Return $\perp$
Return $(y_0, y_1)$
```

Figure 4: Security definitions for honest-but-curious server unlinkability **(left)** and malicious server unlinkability **(right)**.

the simulator more queries than $q_{t,\max}$, for example, $2 \cdot q_{t,\max}$. But this degrades the security guarantee as it means that in q queries to BlindEv on some t a malicious client can potentially compute up to $2q$ POPRF outputs for that tag t.

Request privacy and unlinkability. Our second goal is to capture privacy for clients. This means not only that requests should hide the private input portion x, but also that request/response transcripts and output POPRF values should be unlinkable. We formalize two models for this goal, corresponding to the level of maliciousness by a misbehaving server.

Game POPRIV1 (Figure 4, top game) captures an indistinguishability experiment in which the adversary can query to obtain full transcripts (including output) resulting from honest blinded evaluation of a POPRF. The transcripts are either returned properly ($b = 0$) or with the request, response pairs swapped relative to the outputs ($b = 1$). Intuitively, if the adversary cannot distinguish between these two worlds, then there is no way to link a POPRF output value to a particular blinded evaluation, despite the adversary knowing the secret POPRF key. This captures also input privacy security: if a request reveals some information about the input x this can be used to win the POPRIV1 game. We sometimes refer to this as request privacy against passive adversaries, because the adversary cannot interfere with the server's proper execution.

The advantage of a POPRIV1 adversary $\mathcal{A}$ in the P-model is defined by

$$\mathsf{Adv}^{\text{po-priv1}}_{\mathsf{Fn},\mathsf{P},\mathcal{A}}(\lambda) = \left| \Pr\left[\mathrm{POPRIV1}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1 \right] - \Pr\left[\mathrm{POPRIV1}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1 \right] \right|$$

where the probability spaces are taken over the random choices made in the games and the events signify that the game outputs the value one. We say a Fn scheme is *perfectly private* if $\mathsf{Adv}^{\text{po-priv1}}_{\mathsf{Fn},\mathsf{P},\mathcal{A}}(\lambda) = 0$ for all adversaries $\mathcal{A}$.

POPRIV1 security does not capture malicious servers that deviate from the protocol. So, for example, it doesn't rule out attacks in which the server replies with garbage to a blinded evaluation request.

Our next game POPRIV2 allows the adversary to choose the public keys used for request generation and leaves to the adversary how to reply to requests. The game therefore splits transcript generation across two oracles, a request oracle (REQ) and finalize oracle (FIN). The first oracle replies with a randomly ordered pair of request messages based on the challenge bit, and the second oracle can be queried with adversarially chosen response messages. The game requires that neither y_0 nor y_1 is equal to $\bot$ — if either is then the finalize oracle returns $\bot$. This prevents the trivial attack of corrupting one reply but not the other.

The advantage of a POPRIV2 adversary $\mathcal{A}$ in the P-model is defined by

$$\mathsf{Adv}^{\text{po-priv2}}_{\mathsf{Fn},\mathsf{P},\mathcal{A}}(\lambda) = \left|\Pr\left[\text{POPRIV2}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right] - \Pr\left[\text{POPRIV2}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right]\right|$$

where the probability spaces are taken over the random choices made in the games and the events signify that the game outputs the value one.

POPRIV2 is strictly stronger than POPRIV1. Looking ahead our new POPRF meets POPRIV1 when verification is omitted, and POPRIV2 when verification is required.

Uniqueness. Lastly, we discuss an additional property that is relevant in the verifiable setting when clients want to ensure that servers honestly perform blind evaluations. This means that the output of the blind evaluation protocol should be consistent relative to the public key pair, i.e., consistent with the output of unblinded evaluation using the secret key. Our correctness definition requires this is the case for honest execution of the algorithms. We formalize this correctness property for malicious servers as a uniqueness definition POUNIQ, taking inspiration from definitions used previously for verifiable random functions (c.f., [DY05]). In short, no malicious server should be able to convince a client into accepting two different outputs for the same (pk, t, x). We show that uniqueness is implied by correctness and POPRIV2. The complete definition for POUNIQ, theorem statement, and proof are deferred to Appendix H.

Relation to partially blind signatures. POPRFs are related to two-move partially blind signatures, which were introduced by Abe and Fujisaki [AF96]. A partially blind signature is a tuple of algorithms

$$\mathsf{DS} = (\mathsf{DS.Setup}, \mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Ver}, \mathsf{DS.Req}, \mathsf{DS.BlindSign}, \mathsf{DS.Finalize})$$

where the first four algorithms define a standard digital signature scheme for message space consisting of pairs (t, m), called the public input (or tag) and private message, respectively. Signatures can also be generated via an interactive protocol which, like we did for POPRFs, we formalize simply as a single round trip protocol initiated by a client running $\mathsf{DS.Req}(pk, t, m)$ to generate a request message req and client state st, sending the former to the server which runs $\mathsf{DS.BlindSign}(sk, req)$ to generate and send a response rep back to the client, which then computes a signature via $\mathsf{DS.Finalize}(st, rep)$. This protocol should achieve blindness, which can be defined similarly to our request privacy definition above for POPRFs.

The main security property targeted is one-more unforgeability, which, roughly speaking, states that an adversarial client can't generate $q + 1$ unique message-signature pairs $(m_1, \sigma_1), \ldots, (m_{q+1}, \sigma_{q+1})$ that all verify under a public key pk and public tag t even when given the ability to query a blind signing oracle with the only restriction being that only q queries can be made for the chosen public tag t. This intuitively enforces that each query to the blind signing oracle only results in one learned signature, and queries for a different public tag do not help in forging a signature for the target tag. We present a complete formal treatment of partially blind signatures in Appendix E.

A partially blind signature is unique if $\mathsf{DS.Sign}$ is deterministic and its output on (pk, t, m) matches that of the interactive protocol when initiated on the same triple. A blind signature is just a partially blind signature with t omitted. JKK observed that one can transform unique blind signatures into OPRFs by hashing the signature. A similar transform exists to build a POPRF from a unique partially blind signature. We provide details and proof of this transform in Appendix F for both cases, with and without public input). (As far as we are aware there has been no formal treatment of this observation.)

Most prior partially blind signature schemes are not unique, e.g., [AF96, AO00]. The only unique scheme we are aware of is due to Zhang, Safavi-Naini, and Susilo (ZSS) [ZSS03], but it relies on bilinear pairings and so this generic transformation will not achieve our goals for a POPRF. Moreover as mentioned in the introduction, the security analysis in ZSS is wrong. That said, our construction shares much of the underlying structure from the ZSS one. Using our new proof techniques, we furthermore provide the first complete proof of the ZSS partially blind signature scheme (see Appendix G).

4 The 3HashSDHI POPRF

We now turn to our main result: providing a new POPRF. Our construction combines elements of the 2HashDH construction with a technique used by Dodis and Yampolskiy for their verifiable PRF; it is also related to a partially blind signature scheme suggested by Zhang, Safavi-Naini, and Susilo. We call our construction 3HashSDHI, which we often abbreviate to 3H. The name refers to its use of three hashes and reliance on the strong inverse Diffie-Hellman assumption.

Algorithms. Our protocol relies on a group $\mathbb{G}$ of prime order p and with generator g. As mentioned in the introduction, the 3HashSDHI protocol computes a PRF output as

$$\mathsf{3H.Ev}(sk, t, x) = \mathsf{H}_2\left(t, x, \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}\right)$$

where $\mathsf{H}_1\colon \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_2\colon \{0,1\}^* \to \{0,1\}^{\gamma_2}$, and $\mathsf{H}_3\colon \{0,1\}^* \to \{0,1\}^{\gamma_3}$ are the titular hash functions. Note that H_1 has range the group $\mathbb{G}$, whereas the second and third hashes output bit strings of length γ_2 and γ_3. By default we set $\gamma_2 = \lambda$ and $\gamma_3 = 2\lambda$. The third hash must be collision resistant for security to hold. Looking ahead to the security analysis, we will model the hash functions as random oracles. The setup, key generation, and full evaluation algorithms are shown in pseudocode below.

```
$\mathsf{3H.Setup}(\lambda)$
    $(p, g, \mathbb{G}) \leftarrow\!\$\ \mathsf{GGen}(\lambda)$
    $pp \leftarrow (p, g, \mathbb{G})$
    Return $pp$

$\mathsf{3H.KeyGen}^{\mathsf{H}_1\times\mathsf{H}_2\times\mathsf{H}_3}(pp)$
    $(p, g, \mathbb{G}) \leftarrow pp$
    $sk \leftarrow\!\$\ \mathbb{Z}_p$ ; $pk \leftarrow g^{sk}$
    Return $(pk, sk)$

$\mathsf{3H.Ev}^{\mathsf{H}_1\times\mathsf{H}_2\times\mathsf{H}_3}(sk, t, x)$
    $Y \leftarrow \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}$
    $Z \leftarrow \mathsf{H}_2(t, x, Y)$
    Return $Z$
```

Here, GGen denotes a group parameter generator outputting a triple $(p, g, \mathbb{G})$ consisting of a prime p, (the description of) a group $\mathbb{G}$ of order p, and a generator g of $\mathbb{G}$.

The blind evaluation protocol has a client compute $\mathsf{H}_1(x)$ and mask the resulting group element by raising it to a random scalar r. The client can send the resulting blinded value B to the server, who can then raise B to $1/(sk + \mathsf{H}_3(t))$ and return the result. The client then finalizes by raising the returned value to $1/r$ in order to remove the blinding, followed by the final step of computing the final hash H_2. The blinding ensures request privacy.

We optionally can extend this blinded evaluation protocol to include a proof that the server properly exponentiated B. This is necessary to have the protocol enjoy POPRIV2 security, which is important in some (but not all) applications. At first, it may not be obvious how to prove to the client that the server is returning $B' = B^{1/(sk+\mathsf{H}_3(t))}$ relative to the public key g^{sk}, because the sum appears in the denominator. However, we can use the following trick: the server generates a standard DL proof that $B = (B')^k$ for some k. The client runs verification by explicitly reconstructing $g^k = pk \cdot g^{\mathsf{H}_3(t)} = g^{sk+\mathsf{H}_3(t)}$. This means that the verification procedure checks the special structure of the exponent k.

The full protocol, including the NIZK (which uses its own hash H_4), is shown in Figure 5. Here we show that t is sent from the client to server, though in some applications the server may receive t out-of-band. Execution requires just one round trip. It requires just two group exponentiations on the client side (and, when using the NIZK, those used for its verification). The server uses one exponentiation plus one for the NIZK proof.

Relation to partially blind signatures. 3HashSDHI is closely related to a partially blind signature suggested by Zhang, Safavi-Naini, and Susilo [ZSS03]. It uses groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ each of order p, with generators g_1, g_2, g_T, and that come equipped with an efficient-to-compute pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that for any $\alpha, \beta \in \mathbb{Z}_p$ it holds that $e(g_1^{\alpha}, g_2^{\beta}) = g_T^{\alpha\beta}$. Their signature is defined as

$$\mathsf{ZSS1.Ev}(sk, t, x) = \mathsf{H}_{\mathbb{G}_2}(x)^{1/(sk+\mathsf{H}_3(t))}$$

where $\mathsf{H}_{\mathbb{G}_2}\colon \{0,1\}^* \to \mathbb{G}_2$ hashes onto the group $\mathbb{G}_2$ and H_3 is as defined above for 3HashSDHI. We use $\mathsf{ZSS1}$ to differentiate from our suggested modifications, which we call $\mathsf{ZSS2}$ and discuss in Appendix G. As can be seen, 3HashSDHI uses essentially the same structure, combined with a final hash but we dispense with

```
$\underline{\mathsf{3H.Req}^{\mathsf{H}_1\times\mathsf{H}_2\times\mathsf{H}_3\times\mathsf{H}_4}(pk,t,x)}$
$r \leftarrow\!\!\$\ Z_p$ ; $B \leftarrow \mathsf{H}_1(x)^r$
Return $((pk,r,t,x),B)$
                                   $\xrightarrow{B,\,t}$
                                        $\underline{\mathsf{3H.BlindEv}^{\mathsf{H}_1\times\mathsf{H}_2\times\mathsf{H}_3\times\mathsf{H}_4}(sk,t,B)}$
                                        $k \leftarrow sk + \mathsf{H}_3(t)$
                                        $B' \leftarrow B^{1/k}$
                                        $\pi \leftarrow\!\!\$\ \Sigma_{\mathcal{R}}.\mathsf{Prove}^{\mathsf{H}_4}(k,(g,g^k,B',B))$
                                   $\xleftarrow{B',\,\pi}$
                                        Return $(B',\pi)$
$\underline{\mathsf{3H.Finalize}^{\mathsf{H}_1\times\mathsf{H}_2\times\mathsf{H}_3\times\mathsf{H}_4}(B',\pi;(pk,r,t,x))}$
$Y \leftarrow (B')^{1/r}$
Require $\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}_4}((g,g^{\mathsf{H}_3(t)}\cdot pk,B',B),\pi)$
$Z \leftarrow \mathsf{H}_2(t,x,Y)$
Return $Z$
```

Figure 5: Blind evaluation for our 3H POPRF construction. All three algorithms have implicit input the parameters $pp = (p, g, \mathbb{G})$ that describe the group used. The NIZK uses relation $\mathcal{R} = \{(g, U, V, W), (\alpha) : U = g^\alpha \wedge W = V^\alpha\}$.

the use of bilinear pairings using instead NIZKs to provide verifiability. We also comment on two aspects of the original security analysis of the partially blind signature scheme in [ZSS03]: (1) the analysis of one-more unforgeability is incorrect; and (2) it contains an incorrect claim that so-called "exponential" blinding (see below for discussion in the context of OPRFs) is insecure. More precisely, for (1), the claimed security proof relies on a lemma (stated without a proof) which says that if the scheme is secure, in terms of one-more unforgeability, for any fixed public input, then it is secure as a partially-blind signature – such lemma does not appear to be provable; for (2), the claimed distinguishing test does not work, rather it identifies every pair of signature and signing transcript as a match, regardless of whether they are associated. Our new techniques, in particular a variant of the new gap assumption discussed in the next section, enable a new proof of unforgeability for the ZSS partially blind signature, and we also provide a proof that exponential blinding is secure. See Appendix G for the details.

Comparison to prior (O)PRFs. Recall that the 2HashDH OPRF is defined by $\mathsf{2HashDH.Ev}(sk, x) = \mathsf{H}_2(x, \mathsf{H}_1(x)^{sk})$. On the other hand, the DY PRF [DY05] is evaluated on a message t via $\mathsf{DY.Ev}(sk, t) = h^{1/(sk+t)}$ for generator h. Thus, our 3HashSDHI can be seen as blending of the two approaches, basically defining, for each x, a separate instance of the DY PRF with generator $h = \mathsf{H}_1(x)$ and input message t. The way we combine them retains the simple blinding mechanism of 2HashDH to allow hiding x. Despite the similarity to the prior constructions, analyzing security requires new techniques (see the next section).

2HashDH is often formulated using an alternative "multiplicative" blinding strategy as opposed to the "exponential" blinding presented here. Multiplicative blinding enables client-side performance improvements of fixed-base exponentiation with precomputation over variable-base exponentiation, but has been shown to have some security drawbacks in the non-verified setting [JKX21]. 3HashSDHI is not compatible with the multiplicative blinding protocol used by 2HashDH, however Appendix I presents an alternative multiplicative blinding protocol [ZSS03] that enjoys the same performance benefits.

Miao et al. construct an oblivious evaluation protocol for the DY PRF [MPR+20]. Their approach makes use of the additive-homomorphic Camenisch-Shoup encryption scheme [CS03] and related proofs of discrete log representation. In contrast, 3HashSDHI uses the more efficient oblivious evaluation approach of 2HashDH and uses the structure of DY to tie in the public metadata, meaning the more expensive DY oblivious evaluation techniques are avoided.

Pythia [ECS+15] provided the first POPRF. It uses pairing-friendly groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ each of order p, with generators g_1, g_2, g_T. Then, the Pythia POPRF is defined by

$$\mathsf{Pythia.Ev}(sk, t, x) = e(\mathsf{H}_{\mathbb{G}_1}(t), \mathsf{H}_{\mathbb{G}_2}(x))^{sk}$$

where $\mathsf{H}_{\mathbb{G}_2}$ and $\mathsf{H}_{\mathbb{G}_2}$ are hash functions that map to the groups $\mathbb{G}_1, \mathbb{G}_2$. This construction enables blinded evaluation by sending $B \leftarrow \mathsf{H}_{\mathbb{G}_2}(x)^r$, and having the server respond with $e(\mathsf{H}_{\mathbb{G}_1}(t)^{sk}, B)$. It also has some other features that were desirable in the password hardening context for which Pythia was designed, specifically, that one can have compact key rotation tokens of the form $\Delta = sk'/sk$. The token Δ can be shared with a client to help it update previously computed POPRF values for any t, x. Compared to Pythia's POPRF, 3HashSDHI avoids use of pairings. This makes it faster to compute and saves bandwidth.

That said, the 3HashSDHI construction does not support key rotations even if one omits the final H_2

evaluation. To expand, consider using a (non-compact) key rotation in a way analogous to Pythia, i.e., distributing to a client $\Delta_{t_1} = (sk + \mathsf{H}_3(t_1))/(sk' + \mathsf{H}_3(t_1))$ and $\Delta_{t_2} = (sk + \mathsf{H}_3(t_2))/(sk' + \mathsf{H}_3(t_2))$. This would allow rotating (unhashed) POPRF outputs on public inputs t_1 and t_2 from an old key sk to a new key sk'. But this also trivially reveals sk and sk', given that $\mathsf{H}_3(t_1)$ and $\mathsf{H}_3(t_2)$ are publicly computable. While in the applications we explore in Section 7 we do not need key rotation tokens, the question of finding a POPRF that avoids pairings yet supports key rotations remains open.

We also compare to the recent attribute-based verifiable OPRF (AB-VOPRF) suggested by Huang et al. of Facebook [HIJ+21] for use with Privacy Pass. An attribute-based VOPRF is a POPRF that separates out an explicit algorithm for converting a secret key and attribute t (what we call a tag) into a tag-specific public key, secret key pair. As with other VOPRFs, there is a verifiable, blinded evaluation protocol by which a client can obtain an output on some (t, x) pair without revealing x. Any AB-VOPRF gives a POPRF, and vice versa.

The Facebook construction of an AB-VOPRF combines the 2HashDH approach with the Naor-Reingold PRF [NR97]. Evaluation is defined by

$$\mathsf{FB.Ev}(sk, t, x) = \mathsf{H}_1(x)^{a_0 \cdot \prod_i a_i^{t[i]}}$$

where $sk = a_0, a_1, \ldots, a_{|t|}$ and $t[i]$ indicates the i^{th} bit of t. To make this verifiable, the scheme must provide a more complex NIZK involving $|t|$ group elements, making it expensive to transmit and verify, particularly in applications where a wide variety of tags t will be used. In comparison 3HashSDHI is as efficient as 2HashDH.

Finally, a concurrent, independent work by Silde and Strand [SS21] describe what we call the 3HashSDHI protocol and how it could be useful for Privacy Pass and the Facebook de-identified logging application. They formalize a notion of anonymous token security that is more tailored to Privacy Pass style applications (compared to our general POPRF definitions), but this definition contains the aforementioned problem (see Section 3) of not performing query accounting on a per public input basis, making it too weak of a security notion for their applications. In addition, the security analysis relative to this notion is incomplete, and so the paper does not yet provide a proof even of this weaker security notion. Nevertheless, their work underscores the benefits of the 3HashSDHI protocol in the applications they explore and our proof techniques (in particular, the new one-more gap SDHI assumption discussed in the next section) should enable improvements to their analysis.

Extension to private metadata bit. Recall a primary application of OPRFs is in the construction of anonymous tokens. We have thus far been concerned with adding support for public metadata, but there are also settings that benefit from being able to associate *private* metadata to tokens that can only be identified by the issuer. To prevent trivial linking attacks by a malicious server, it is necessary that the private metadata space remain small. Kreuter et al. propose a variant of Privacy Pass (based on the 2HashDH OPRF) that supports a single private metadata bit [KLOR20]. The high level approach is to simply maintain two keys and prove in zero knowledge that the token is issued under one of the two keys. However, they observe that a deterministic primitive (like a PRF) is insufficient to achieve indistinguishability between private metadata bits. Therefore, the core of their construction is a new anonymous token protocol that can be considered as a randomized variant of 2HashDH. It is likely similar techniques can be applied to construct a randomized version of 3HashSDHI to support public metadata as well as a private metadata bit; we leave the details of such a construction to future work. Silde and Strand propose a construction along these lines, but as mentioned before, the security analysis is incomplete [SS21].

5 Security Analysis

We show formally that the 3HashSDHI PO-PRF enjoys pseudorandomness and request privacy. The former is the more complex analysis; we start with it.

5.1 Pseudorandomness

The main technical challenge is showing that 3HashSDHI meets our pseudorandomness definition, captured by game POPRF (Figure 3 in Section 3). We start with an overview of our proof strategy, and then state our main result.

Proof strategy. Our proof of pseudorandomness proceeds in several steps. First, we introduce a new discrete log (DL) type cryptographic hardness assumption: the one-more gap strong Diffie-Hellman inversion problem, denoted (m,n)-OM-GAP-SDHI for parameters m,n that we will explain. The new assumption is a generalization of prior one-more DL assumptions, but extended with two oracles and a more involved one-more winning condition which depends on the number of queries with a specific form to one of the oracles. We show that we can build a POPRF simulator such that, in the ROM, distinguishing between the real $(b=1)$ and ideal worlds $(b=0)$ reduces to breaking an instance of (m,n)-OM-GAP-SDHI where m is the number of H_3 queries made by $\mathcal{A}$ and n is the number of H_1 queries.

We also analyze the security of our new assumption, showing that, in the Algebraic Group Model (AGM) [FKL18] it reduces to one of the uber assumptions from Bauer, Fuchsbauer, and Loss (BFL) [BFL20]. In turn, we can use a result from BFL to finally show that our new assumption is implied (again, in the AGM) by the q-DL assumption. This provides good evidence of the difficulty of the problem, and allows us to derive precise concrete security bounds.

The one-more gap SDHI assumption. Game (m,n)-OM-GAP-SDHI is shown in Figure 7. The game generates a group instance and a challenge secret sk. The adversary $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ runs in two stages. In the first stage it receives the group description $p,\mathbb{G}$ and outputs a sequence of n scalar values $c_1,\ldots,c_n$. Importantly $\mathcal{A}_1$ does not receive g, forcing it to commit to the c_i values in a way independently of the generator g. We assume that g is randomly chosen; this will be important in our analysis. Then, the second stage $\mathcal{A}_2$ is run on input the generator g, g^{sk}, and a vector of m group elements $g^{y_1},\ldots,g^{y_n}$. The adversary is given access to two oracles. The SDH oracle returns $B^{1/(sk+c_i)}$ for arbitrary B and one of the previously specified c_i values. The SDDH oracle is a decision oracle that helps the adversary determine whether $Z=Y^{1/(sk+c_i)}$ for arbitrary Y,Z and one of the previously specified c_i values.

The adversary outputs a distinguished index γ indicating a c_γ value, as well as a set of ℓ pairs $(Z_i,\alpha_i)\in\mathbb{G}\times[0..m]$. The adversary wins if $\ell>q_\gamma$ and $Z_i=Y_{\alpha_i}^{1/(sk+c_\gamma)}$ for all $1\le i\le\ell$. Here q_γ is the number of queries to the SDH with second input set to γ. Without the "one more" restriction of $\ell>q_\gamma$, it is trivial to win. We define the (m,n)-OM-GAP-SDHI-advantage of an adversary $\mathcal{A}$ by

$$\mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}}(\lambda)=\Pr\left[\,(m,n)\text{-OM-GAP-SDHI}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)\Rightarrow\mathsf{true}\,\right].$$

An adversary $\mathcal{A}$ has query budget $(\vec{q},q_{\mathsf{SDDH}})$ for $\vec{q}=[\vec{q}_1,\ldots,\vec{q}_n]$ if at the end of the game $\mathcal{A}$ has made at most $\vec{q}_i$ queries to SDH with index i and has made at most q_{SDDH} queries to SDDH.

We note that a weakening of the assumption dispenses with the more granular per-c-value accounting, instead just asking that the adversary can't come up with $\ell>q$ solutions for any mixture of Y_i and c_j values. This variant is much easier to analyze in the AGM, but is not sufficient for our analysis.

Reducing (m,n)-OM-GAP-SDHI to q-DL. In two steps, we show how to reduce this assumption, in the AGM, to the difficulty of q-DL. The latter involves a game q-DL (Figure 6) that generates a group instance $p,g,\mathbb{G}$ for security parameter λ, and gives an adversary $g,g^x,g^{x^2},\ldots,g^{x^q}$ for a random scalar x. The adversary must output x. We define the advantage of a q-DL-adversary $\mathcal{A}$ to be $\mathsf{Adv}^{q\text{-dl}}_{\mathsf{GGen},\mathcal{A}}(\lambda)=\Pr\left[\,q\text{-DL}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)\Rightarrow\mathsf{true}\,\right]$.

```
Game $q$-DL$^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
$(p,g,\mathbb{G}) \leftarrow\$ \mathsf{GGen}(\lambda)$
$x \leftarrow\$ \mathbb{Z}_p$
$x' \leftarrow\$ \mathcal{A}\big(p,\mathbb{G},g,\big[g^{x^i}\big]_{i=1}^{q}\big)$
Return $x = x'$
```

Figure 6: The q-type discrete log security game.

As a convenient middle layer, we rely on BFL's "Uber-assumption" [BFL20], formalized via the game m-UBER in Figure 8. It involves a game where the adversary can obtain $g^{\rho(\vec{x})}$ by querying an arbitrarily chosen m-variate polynomial $\rho(\vec{X})$ to an oracle EV, for a secret vector $\vec{x}\leftarrow\$\,\mathbb{Z}_p^m$. The adversary wins if it outputs successfully $g^{\mu(\vec{x})}$ for some polynomial $\mu(\vec{X})$ which is *independent* of the polynomials $\rho_1(\vec{X}),\ldots,\rho_q(\vec{X})$ queried to EV, i.e., $\mu(\vec{X})$ cannot be expressed as an affine combination $\mu(\vec{X})=\alpha_1\rho_1(\vec{X})+\cdots+\alpha_q\rho_q(\vec{X})+\beta$. The adversary can also query an additional DECIDE oracle with a polynomial $\rho(\vec{X})$, as well as group elements $g^{y_1},\ldots,g^{y_m}$, and learn whether $g^{\rho(y_1,\ldots,y_m)}=0$ or not. We denote the corresponding advantage as $\mathsf{Adv}^{m\text{-uber}}_{\mathsf{GGen},\mathcal{A}}(\lambda)=\Pr\left[\,m\text{-UBER}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)\Rightarrow\mathsf{true}\,\right]$.

We are going to prove the following theorem in Appendix A. Here and subsequently we use '$\approx$' to denote that runtimes are equal up to small constant factors.

```
Game $(m,n)$-OM-GAP-SDHI$^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
$(p, g, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(\lambda)$
$sk \leftarrow\$ \mathbb{Z}_p$ ; $[y_i]_i^m \leftarrow\$ [\mathbb{Z}_p]_i^m$
$(st_{\mathcal{A}}, [c_i]_i^n) \leftarrow\$ \mathcal{A}_1(p, \mathbb{G})$
Require $\forall_{i \neq j}^n c_i \neq c_j$
$(\gamma, [Z_i, \alpha_i]_i^\ell) \leftarrow\$ \mathcal{A}_2^{\text{SDH},\text{SDDH}}\left(g, g^{sk}, [g^{y_i}]_i^m : st_{\mathcal{A}}\right)$
Require $q_\gamma < \ell \wedge \forall_{i \neq j}^\ell \alpha_i \neq \alpha_j$
Return $[Z_i]_i^\ell = \left[g^{y_{\alpha_i}/(sk+c_\gamma)}\right]_i^\ell$

Oracle SDH$(B, i)$
Require $i \notin [1, n]$
$q_i \leftarrow q_i + 1$
$Z \leftarrow B^{1/(sk+c_i)}$
Return $Z$

Oracle SDDH$(Y, Z, i)$
Return $Z = Y^{1/(sk+c_i)}$
```

Figure 7: The one-more gap strong Diffie-Hellman inversion security game.

```
Game $m$-UBER$^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)$
$(p, g, \mathbb{G}) \leftarrow\$ \mathsf{GGen}(\lambda)$
$Q \leftarrow \{\}$
$\vec{x} = [x_i]_i^m \leftarrow\$ [\mathbb{Z}_p]_i^m$
$(U, \mu(\vec{X})) \leftarrow\$ \mathcal{A}^{\text{Ev},\text{Decide}}(p, \mathbb{G}, g)$
Return $\left(U = g^{\mu(\vec{x})} \wedge Q \perp\!\!\!\perp \{\mu(\vec{X})\}\right)$

Oracle Ev$(\rho(\vec{X}))$
$Q \leftarrow Q \cup \{\rho(\vec{X})\}$
Return $g^{\rho(\vec{x})}$

Oracle Decide$(\rho(\vec{X}), [Y_i]_i^n)$
$\vec{y} = [y_i]_i^n \leftarrow [\log_g Y_i]_i^n$
Return $\rho(\vec{y}) \equiv_p 0$
```

Figure 8: The interactive, flexible-output, polynomial uber assumption with decision oracle. Here, $\perp\!\!\!\perp$ denotes algebraic independence.

Theorem 2 *For any algebraic adversary $\mathcal{A}_{\text{sdhi}}$ of (m,n)-OM-GAP-SDHI with query budget ($\vec{q} = [q_1, \ldots, q_n]$, q_{SDDH}), and any GGen outputting $(p, g, \mathbb{G})$, where g is a uniformly chosen element of $\mathbb{G}$, we give adversary $\mathcal{A}_{\text{uber}}$ such that*

$$\mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\text{sdhi}}}(\lambda) \leq (q_{\max} + 1) \cdot \mathsf{Adv}^{(m+1)\text{-uber}}_{\mathsf{GGen},\mathcal{A}_{\text{uber}}}(\lambda) + \frac{q}{p},$$

where $q = \sum_i^n q_i$ and $q_{\max} = \max\{q_i\}_i^n$. Also, $\mathcal{A}_{\text{uber}}$ makes at most q queries to its polynomial evaluation oracle with maximum degree $q+1$, and outputs a polynomial of degree at most q. Further, $T(\mathcal{A}_{\text{sdhi}}) \approx T(\mathcal{A}_{\text{uber}})$.

It is important here to note that the theorem assumes that the query budgets q_i corresponding to different i's are *fixed* a priori, rather than being chosen adaptively.

Combined with a basic reduction from [BFL20], this gives us the following immediate corollary.

Corollary 3 *For any algebraic adversary $\mathcal{A}_{\text{sdhi}}$ of (m,n)-OM-GAP-SDHI, with query budget ($\vec{q} = [q_1, \ldots, q_n]$, q_{SDDH}), and any GGen outputting $(p, g, \mathbb{G})$, where g is a uniformly chosen element of $\mathbb{G}$, we give adversary $\mathcal{A}_{\text{dl}}$ such that*

$$\mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\text{sdhi}}}(\lambda) \leq (q_{\max} + 1) \cdot \mathsf{Adv}^{(q+1)\text{-dl}}_{\mathsf{GGen},\mathcal{A}_{\text{dl}}}(\lambda) + \frac{q}{p},$$

where $q = \sum_i^n q_i$ and $q_{\max} = \max\{q_i\}_i^n$. Further, $T(\mathcal{A}_{\text{sdhi}}) \approx T(\mathcal{A}_{\text{dl}})$.

The main difficulty of the proof of Theorem 2 in Appendix A stems from the one-more requirement $\ell > q_\gamma$ in the winning condition, which is defined in a way that depends on the specific number of queries q_γ to SDH$(\cdot, \gamma)$. To gain some intuition, it is convenient to think of the game in algebraic terms (and this point of view is also accurate when casting our proof in the AGM).[1] Specifically, let us describe exponents of the elements provided to $\mathcal{A}_2$ as formal polynomials X_0 (standing for the secret key) and $X_1, \ldots, X_m$ (for the values $y_1, \ldots, y_m$). Initially, the adversary has these polynomials available, and now a call to SDH(P, i) can also be thought of as dividing some polynomial P (or more generally, a rational function) by $(X_0 + c_i)$. The rational function P can be any affine combination of the functions obtained so far, and SDH(P, i) adds a new rational function to this set of available rational functions. In other words, consecutive queries induce

[1] We ignore the SDDH oracle in this discussion, and it will be easy to handle in the actual proof via the Decide oracle.

a *transcript* τ consisting of the initial functions $X_0, X_1, \ldots, X_m$, and the functions returned by SDH. The goal of the adversary is to ensure that, for some γ, the span[2] of τ contains $\ell > q_\gamma$ functions of the form

$$\frac{X_{\alpha_1}}{X_0 + c_\gamma}, \ldots, \frac{X_{\alpha_\ell}}{X_0 + c_\gamma}.$$

An adversary cannot achieve this goal naively by querying $\mathrm{SDH}(X_{\alpha_j}, \gamma)$ for $j \in [\ell]$ without violating the query budget. Still, the key difficulty here is that the adversary *could*, after learning (say) $X_1/(X_0 + c_\gamma)$ make a further query that would give $X_1/(X_0+c_\gamma)(X_0+c_{\gamma'})$ for some $\gamma' \neq \gamma$. This second query would *not* count towards q_γ, and could potentially be helpful, as it *does* involve c_γ.

The bulk of our proof shows that arbitrary queries to SDH cannot, in fact, help the adversary. We do so via a careful inductive analysis which shows that the transcript τ can be rewritten in an equivalent way, call it τ', without affecting its span. In particular, τ' only involves rational functions whose denominators have form $(X_0 + c_i)^k$ for some i and k, but no products involving multiple c_i's appear in the denominators. We leverage this structure to show that the span of such τ' can include at most q_γ rational functions of the form $\frac{X_i}{X_0+c_\gamma}$.

Now, given the above algebraic game cannot be won, an adversary winning the game must necessarily produce an output $(\gamma, [Z_i, \alpha_i]_i^\ell)$ where for at least one $i \in [\ell]$, we have that the polynomial $X_{\alpha_i}/(X_0 + c_i)$ is not in the span of the queries to SDH. This lends itself naturally to a reduction to the Uber-assumption, which we describe in full in the proof.

Reducing to (m, n)**-OM-Gap-SDHI.** We now turn to showing that we can reduce the pseudorandomness security of 3HashSDHI to our new assumption. We focus on the verifiable version of 3HashSDHI; an analysis for the non-verifiable version is easily derived from our analysis here. Our analysis is in the RO model; we model all four hash functions as ROs.

We start by describing the simulator used in the proof. The simulator's goal is to respond to blind evaluation and RO queries so that the resulting transcript of values is indistinguishable from real responses. Importantly, the simulator must do this without making too many calls to the full evaluate oracle for each BlindEval-queried public input t. Intuitively, achieving this security enforces that a malicious client can not exploit the blinded evaluation oracle to do more than help it compute a single POPRF output for the particular requested t.

The simulator works as follows. It chooses its own secret key sk and answers the H_1 and H_3 queries with random group elements and scalars, respectively. To answer a blinded evaluation query, it runs the scheme's blind evaluation algorithm $\mathsf{Fn.BlindEv}(sk, B)$, except that it uses the NIZK's simulator to generate the proof π (and to simulate any ideal primitive underlying the NIZK, i.e., H_4). The key challenge is in simulating H_2 queries, that which enables the adversary to "complete" a blinded evaluation. The simulator must arrange that the value it returns in response to H_2 queries is consistent with the random value returned by Ev. To do so, the simulator checks whether a queried point (t, x, Y) is such that $Y = \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}$ and, if so, it queries $\mathrm{LimEv}(t, x)$ and returns the output. Otherwise, it chooses a random point to return. The simulator can perform this check because it chose sk. See Figure 10 in Appendix B for the full details of the simulator.

The simulation can fail should the adversary be able to query it on a point $Y = \mathsf{H}_1(x)^{1/(sk+\mathsf{H}_3(t))}$ when the simulator cannot make another call to LimEv for that value t. This can only arise should the adversary query H_2 on more such values t, Y than queries it so far made to BlindEv on that t. We show that an adversary, that can do so, can also win the (m, n)-OM-Gap-SDHI game where m, n are the total number of queries involving a distinct x value and distinct t value, respectively. (We define this more precisely below.) This step also relies on the collision resistance of H_3, which holds in the ROM.

To formalize this, we state below a theorem using the ideal primitive model in which $\mathsf{P} = \mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3 \times \mathsf{H}_4$ for random oracles over $\mathsf{H}_1 : * \to \mathbb{G}$, $\mathsf{H}_2 : * \times * \times \mathbb{G} \to \{0,1\}^\lambda$, $\mathsf{H}_3 : * \to \mathbb{Z}_p$, $\mathsf{H}_4 : \mathbb{G}^6 \to \mathbb{Z}_p$ for $(p, \mathbb{G})$ determined by $\mathsf{GGen}(\lambda)$. Here '$*$' denotes the set of arbitrary inputs. We define the query budget for an adversary $\mathcal{A}_{\mathrm{prf}}$ in the P model to be a tuple $(m, n, q_{\mathsf{E}}, \vec{q}, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4})$ where:

- m is the maximum number of distinct x values queried by $\mathcal{A}_{\mathrm{prf}}$ to H_1 or H_2;
- n is the maximum number of distinct t values queried by $\mathcal{A}_{\mathrm{prf}}$ to BlindEv, H_2, or H_3;

[2]By "span" we mean the set of rational functions that can be obtain by taking *affine* combinations of the functions in τ.

- $\vec{q} = [\vec{q}_1, \ldots, \vec{q}_n]$ is a vector where each $\vec{q}_i$ is the maximum number of queries by $\mathcal{A}_{\rm prf}$ to $\textsc{BlindEv}(t_i, \mathit{req})$ for any req and where we $t_1, \ldots, t_n$ are the (at most) n values t_i queried in the course of the game in the order of when they are queried. (That is, t_1 is the first t value queried, t_2 is the second, etc.) In words, the adversary is limited to some number n of public inputs t that it can target, and makes a limited number of blinded evaluation queries for each of those inputs t.
- q_{E}, q_{H_1}, q_{H_2}, q_{H_3}, and q_{H_4} are the maximum number of queries made by $\mathcal{A}_{\rm prf}$ to the $\textsc{Ev}$, H_1, H_2, H_3, and H_4 oracles, respectively.

Note that our query budget requirement $\vec{q}$ does not restrict *which* values t the adversary can use; these can be picked adaptively. But the number of times each t value is queried is restricted by the order in which they are queried. The granular accounting of blinded evaluation queries via $\vec{q}$ will be important when combining the following theorem with Theorem 2.

Theorem 4 *Let $\mathcal{A}_{\rm prf}$ be a P-model POPRF adversary against $\mathsf{3H}$ with query budget $(m, n, q_{\mathsf{E}}, \vec{q}, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4})$. Then we give a H_4-model adversary $\mathcal{A}_{\rm zk}$ and adversary $\mathcal{A}_{\rm sdhi}$ such that*

$$\mathsf{Adv}^{\rm po\text{-}prf}_{\mathsf{3H},\mathsf{S}[\mathsf{S}_\Sigma],\mathsf{P},\mathcal{A}_{\rm prf}}(\lambda) \le \mathsf{Adv}^{\rm zk}_{\Sigma_{\mathcal{R}},\mathcal{R},\mathsf{H}_4,\mathsf{S}_\Sigma,\mathcal{A}_{\rm zk}}(\lambda) + \mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\rm sdhi}}(\lambda) + \frac{n^2}{p} ,$$

where S is the simulator defined in Figure 10 that makes use of NIZK simulator S_Σ. Adversary $\mathcal{A}_{\rm zk}$ makes q_{H_4} queries to its random oracle and $\mathcal{A}_{\rm sdhi}$ has query budget $(\vec{q}, q_{\mathsf{H}_2})$. Further, $T(\mathcal{A}_{\rm prf}) \approx T(\mathcal{A}_{\rm zk}) \approx T(\mathcal{A}_{\rm sdhi})$.

A detailed proof is given in Appendix B. It proceeds via a sequence of games, starting with the real world $\mathrm{POPRF}^{\mathcal{A}_{\rm po\text{-}prf},1}_{\mathsf{3H},\mathsf{P},\mathsf{S}}(\lambda)$ and first transitioning to a game that replaces the NIZK π with one generated by the NIZK simulator S_Σ. Then we change how $\textsc{Ev}$ queries are handled. Instead of computing the POPRF using sk, we pick a random value and add it to a table R. We also modify the handling of H_2 queries to check if R has been set on a relevant value and, if so, patch up H_2's response so that it maintains consistency. This does not change the distribution of responses to the adversary. Finally, we are in position to perform a reduction to $(q_{\mathsf{H}_1}, q_{\mathsf{H}_3})$-OM-Gap-SDHI: the only difference between this game and the ideal world $\mathrm{POPRF}^{\mathcal{A}_{\rm prf},0}_{\mathsf{3H},\mathsf{P},\mathsf{S}[\mathsf{S}_\Sigma]}(\lambda)$ is when H_2 needs to repair a R value more often than queries to $\textsc{BlindEv}$. This reduction step is made relatively simple by our new assumption, which provides the values and oracles necessary to simulate $\mathcal{A}_{\rm prf}$'s view in a straightforward way.

We can combine the two main theorems with a standard result about the NIZK that we use (restated in Section 2) to give the following corollary.

Corollary 5 *Let $\mathcal{A}_{\rm prf}$ be a P-model POPRF adversary against $\mathsf{3H}$ with query budget $(m, n, q_{\mathsf{E}}, \vec{q}, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4})$ and GGen any group parameter generator outputting $(p, g, \mathbb{G})$, where p is a prime g is a uniformly chosen element of $\mathbb{G}$. Then, we give adversary $\mathcal{A}_{\rm dl}$ such that*

$$\mathsf{Adv}^{\rm po\text{-}prf}_{\mathsf{3H},\mathsf{S}[\mathsf{S}_\Sigma],\mathsf{P},\mathcal{A}_{\rm prf}}(\lambda) \le (q_{\max} + 1) \cdot \mathsf{Adv}^{(q+1)\text{-dl}}_{\mathsf{GGen},\mathcal{A}_{\rm dl}}(\lambda) + \frac{q + n^2}{p} + \frac{3q^2 + q(q_{\mathsf{H}_4} + 4) + 2}{2^\lambda} ,$$

where $q = \sum_i^n q_i$, $q_{\max} = \max\{q_i\}_i^n$. Further, S is the simulator defined in Figure 10 that makes use of NIZK simulator S_Σ, and $T(\mathcal{A}_{\rm prf}) \approx T(\mathcal{A}_{\rm dl})$.

Concrete security and parameter selection. Corollary 5 is interpreted best in the generic-group model (GGM) [Sho97, Mau05], as this yields an absolute bound in terms of $\mathcal{A}_{\rm prf}$'s resources. The advantage of a generic algorithm $\mathcal{A}_{\rm dl}$ running in time T (or more precisely, making T queries to the generic-group oracle) against $(q+1)$-DL in a group of order p is $\mathsf{Adv}^{(q+1)\text{-dl}}_{\mathsf{GGen},\mathcal{A}_{\rm dl}}(\lambda) \le (T + q + 2)^2(q+1)/(p-1)$ (see, e.g., [BFL20] for a proof). This advantage is multiplied by $q_{\max}$ to obtain the dominating term in our final bound. We conjecture however that the bound is somewhat pessimistic, and that the factor $q_{\max}$ is an artifact of the proof. In fact, as we discuss below, a different interpretation of our proof flow, which is particularly meaningful in the GGM, avoids this factor altogether. This improved bound omitting $q_{\max}$ is also essentially tight, since Cheon's attack [Che06] extracts[3] the secret key from q $\textsc{BlindEval}$ queries in time $\sqrt{p/q}$, as long as q divides $p-1$ or $p+1$.

[3] Define $x = (\mathit{sk} + \mathsf{H}_3(t))^{-1}$ for some fixed t. Then, the attacker can just obtain, via consecutive iterative queries, the values $g^x, g^{x^2}, \ldots, g^{x^q}$, and then recover x via Cheon's attack. Finally, $\mathit{sk} = x^{-1} - \mathsf{H}_3(t)$.

Cheon's attack can therefore guide parameter selection, as is also done for 2HashDH deployments. For example, a 256-bit group may be sufficient to achieve security for up $T = 2^{80}$, as this would still accommodate up to $q \approx 2^{96}$ blind evaluations without violating our bounds. In contrast, to ensure security up to $T = 2^{128}$, moving to a 384-bit curve appears necessary. The conclusion being that our choice of parameters is consistent with that for 2HashDH, meaning we achieve the same group operation performance while adding public inputs.

We also note that our reduction to the uber assumption requires the generator to be uniformly chosen. We cannot envision any security issues when the generator is instead fixed, and the need for uniformly chosen generators is likely just an artifact of our proof technique.

Tighter GGM bound. We only sketch the main idea behind the tighter GGM proof, as it is the result of a minor modification of our AGM proof flow. First, note that the $(q_{\max} + 1)$ factor in Corollary 5 is inherited from Theorem 2 and is due to our inability to *efficiently* find, within the adversary $\mathcal{A}_{\text{uber}}$, a good index $j \in [\ell]$ that leads to a break of the uber-assumption. Therefore, we are left with guessing. However, an alternative is to find such j by computing all ℓ possible polynomials $\mu(\vec{X})$, and outputting the one which is independent from those input to EV. Unfortunately, this is computationally expensive, and requires time at least $\Omega(q_{\max}^2)$. In other words, we could make the proof tight with respect to advantage while losing tightness with respect to time complexity. While in our proof flow this needs to be taken into account, in the GGM only the number of group operations matters (i.e., the number of oracle calls), whereas "additional" running time is for free. Thus, if $\mathcal{A}_{\text{prf}}$ makes T queries to its GGM oracles, our proof flow yields (with the proposed modification) an adversary $\mathcal{A}_{\text{dl}}$ with roughly the same number of GGM queries and advantage against $(q + 1)$-DL.

5.2 Request Privacy

We now turn to request privacy, which is simpler to analyze. Intuitively, 3HashSDHI client requests leak no information because the blinding makes them independent of other requests and finalized outputs. The following theorem formalizes this for the case of POPRIV1 for the non-verifiable version of 3HashSDHI.

Theorem 6 *For any* POPRIV1 *adversary* $\mathcal{A}_{\text{po-priv1}}$ *against* 3H *(without client verification) we have that* $\mathsf{Adv}^{\text{po-priv1}}_{\mathsf{3H},\mathcal{A}_{\text{po-priv1}}}(\lambda) = 0$.

Note that the theorem makes no assumptions about the hash functions or group, instead privacy derives directly from the information-theoretic blinding.

Proof: Let G be the same as game $\text{POPRIV1}^{\mathcal{A},b}_{\mathsf{3H},\mathsf{P}}(\lambda)$ except that we replace $(req_d, rep_d) = (\mathsf{H}_1(x_d)^{r_d}, \mathsf{H}_1(x_d)^{r_d \cdot sk})$ with $(req_d, rep_d) = (g^{r_d}, g^{r_d \cdot sk})$ for $d \in \{0,1\}$ and where r_0, r_1 are the random exponents chosen in the two invocations of 3H.Req. Observe that in game G the values returned by REQ are independent of the challenge bit b. Then we have that

$$\Pr\left[\,\text{POPRIV1}^{\mathcal{A},1}_{\mathsf{3H},\mathsf{P}}(\lambda) \Rightarrow 1\,\right] = \Pr\left[\,\text{G} \Rightarrow 1\,\right] = \Pr\left[\,\text{POPRIV1}^{\mathcal{A},0}_{\mathsf{3H},\mathsf{P}}(\lambda) \Rightarrow 1\,\right] .$$

■

Non-verifiable PO-PRFs, including the non-verifiable version of 3HashSDHI, cannot achieve our stronger notion of malicious request privacy. The attack is straightforward since the adversary can simply replace one of the two responses with garbage, and determine the challenge bit. In detail for the case of 3H, adversary $\mathcal{A}_{\text{po-priv2}}$ can pick $sk \in \mathbb{G}$ arbitrarily, let $pk = g^{sk}$, and then query $\text{REQ}(pk, t, x_0, x_1)$ for some arbitrary t, x_0, x_1. It obtains back from the oracle req, req', and then parses req as a pair (B, t). It then queries $\text{FIN}(B^{1/(sk+\mathsf{H}_3(t))}, g)$ to get back reply (y_0, y_1). It checks if $y_0 = \mathsf{H}_2(\mathsf{H}_3(x_0)^{1/(sk+\mathsf{H}_3(t))})$ and returns 0 if so. Otherwise it returns one. This adversary wins with probability 1.

The verifiable version of 3HashSDHI achieves our stronger notion of malicious request privacy, due to the ZKP forcing the malicious server to respond honestly to blinded requests (relative to the public key being used). The following theorem formalizes this, where we model the hash used by the ZKP as a random oracle, and all other hashes as standard model.

Theorem 7 *Let $\mathcal{A}_{\text{po-priv2}}$ be a* POPRIV2 *adversary in the* P*-model against* 3H *that makes at most q queries to* Fin*. We give in the proof below a* $\text{SOUND}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{H}_4}$ *adversary $\mathcal{B}_{\text{sound}}$ such that*

$$\mathsf{Adv}^{\text{po-priv2}}_{\mathsf{3H},\mathsf{P},\mathcal{A}_{\text{po-priv2}}}(\lambda) \leq 4q \cdot \mathsf{Adv}^{\text{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{H}_4,\mathcal{B}_{\text{sound}}}(\lambda)) \ .$$

Further, $T(\mathcal{B}_{\text{sound}}) \approx T(\mathcal{A}_{\text{po-priv2}})$.

Proof: Consider game $\text{POPRIV2}^{\mathcal{A}_{\text{po-priv2}},1}_{\mathsf{3H},\mathsf{P}}(\lambda)$. We consider the event that, in the course of the game, a $\text{Fin}(j,(B'_1,\pi_1),(B'_2,\pi_2))$ query is made such that either

1. $B'_1 \neq B^{1/(sk+\mathsf{H}_3(t_j))}_{j,b}$ but $\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}_4}((g, g^{\mathsf{H}_3(t_j)} \cdot pk_j, B'_1, B_{j,b}), \pi_1) = 1$; or
2. $B'_2 \neq B^{1/(sk+\mathsf{H}_3(t_j))}_{j,1-b}$ but $\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}_4}((g, g^{\mathsf{H}_3(t_j)} \cdot pk_j, B'_2, B_{j,1-b}), \pi_2) = 1$.

Here pk_j, t_j are the values queried to the j^{th} call to Req and we let $sk_j = \text{dlog}_g\, pk_j$. Recall that here $\mathcal{R} = \{(g,U,V,W),(\alpha) \,:\, U = g^\alpha \wedge W = V^\alpha\}$, and verification is therefore checking, in case (1), that

$$B'_2 = B^{\mathsf{H}_3(t_j)+sk_j}_{j,b} \Leftrightarrow (B'_2)^{1/(sk_j+\mathsf{H}_3(t_j))} = B_{j,b}$$

and a similar equality for case (2). So if this event occurs, this means the adversary has violated the soundness of the ZKP: only a single value $\alpha = sk_j + \mathsf{H}_3(t_j)$ can be the witness for $\mathcal{R}$.

To formally reduce to ZKP soundness, first let game G0 be the same as $\text{POPRIV2}^{\mathcal{A}_{\text{po-priv2}}}_{\mathsf{3H},\mathsf{P},b}(\lambda)$ but the bit b is chosen at random from $\{0,1\}$. Let "G0 $\Rightarrow b$" be the event in game G0 that the game returns the value b. (We use this event notation for subsequent games analogously.) Further we let G0_{bad} be the same as G0 except that within each Fin it first computes $sk_j = \text{dlog}_g\, pk_j$ and checks if conditions (1) and (2) hold. If either does not, then it sets a flag bad. Clearly G0_{bad} is not computationally efficient; our reduction will avoid this computationally inefficient step. Finally we let G1 be the same as game G0 except that all $\text{Fin}(j, rep, rep')$ queries are handled by first replacing rep and rep' with the correct values, i.e., $rep \leftarrow B^{sk_j}_{j,b}$ and $rep' \leftarrow B^{sk_j}_{j,1-b}$ where $sk_j \leftarrow \text{dlog}_g(pk_j)$. Notice that G0_{bad} and G1 are identical until the first query, if any, that sets the flag bad. We have that

$$\mathsf{Adv}^{\text{po-priv2}}_{\mathsf{3H},\mathsf{P},\mathcal{A}_{\text{po-priv2}}}(\lambda) = 2 \cdot \Pr[\,\text{G0} \Rightarrow b\,] - 1 \ .$$

and that

$$\Pr[\,\text{G0} \Rightarrow b\,] = \Pr[\,\text{G0}_{\mathsf{bad}} \Rightarrow b\,] \leq \Pr[\,\text{G1} \Rightarrow b\,] + \Pr[\,\text{G0}_{\mathsf{bad}} \text{ sets } \mathsf{bad}\,] \ ,$$

where the inequality comes from the fact that G0_{bad} and G1 are identical-until-bad and application of the fundamental lemma of game playing [BR06]. We now bound the probability that $\Pr[\text{G0}_{\mathsf{bad}} \text{ sets } \mathsf{bad}]$ via reduction to the soundness of the ZKP.

Adversary $\mathcal{B}_{\text{sound}}$ works as follows. First, it randomly chooses a number $q^* \in [1, 2q]$ to serve as its guess for which ZKP π will be forged by the adversary. Here q is the maximum number of Fin queries made by $\mathcal{A}_{\text{po-priv2}}$; each such query includes two proofs. Then $\mathcal{B}_{\text{sound}}$ runs G0, stopping when $\mathcal{A}_{\text{po-priv2}}$ has made $j = \lceil q^*/2 \rceil$ queries to Fin. At this point, $\mathcal{B}_{\text{sound}}$ stops outputting $((g, g^{\mathsf{H}_3(t_j)} pk_j, B'_1, B_{j,b}), \pi_1)$ if q^* is odd and $((g, g^{\mathsf{H}_3(t_j)} pk_j, B'_2, B_{j,1-b}), \pi_2)$ otherwise. Adversary $\mathcal{B}_{\text{sound}}$ avoids computing sk_j; it simply guesses which of the proofs would have caused bad to be set to true, had sk_j been computed and the conditions (1) and (2) been checked. A standard argument yields that

$$\Pr[\,\text{G0}_{\mathsf{bad}} \text{ sets } \mathsf{bad}\,] \leq 2q \cdot \mathsf{Adv}^{\text{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P},\mathcal{B}_{\text{sound}}}(\lambda) \ .$$

To finish the proof, we can observe that G1 always correctly computes responses, and a similar argument as we used for POPRIV1 gives that the transcript observed by $\mathcal{A}_{\text{po-priv2}}$ is independent of the challenge bit b, and so $\Pr[\text{G1} \Rightarrow b] = 1/2$. Combining all the above yields the advantage statement in the theorem.

■

Scheme	**L** (B)	**T** (B)	KeyGen	KeyVerify	Req	BlindEv	Finalize	Ev
2HashDH	16	1	0	0	73	222	392	77
2HashDH	16	8	0	0	75	229	404	79
2HashDH	16	64	0	0	84	256	447	89
3HashSDHI	16	1	0	0	85	369	527	125
3HashSDHI	16	8	0	0	76	334	477	112
3HashSDHI	16	64	0	0	75	328	471	110
Pythia	16	1	168	0	849	4068	6070	2871
Pythia	16	8	180	0	831	4099	6092	2881
Pythia	16	64	171	0	809	3922	5849	2768
ABVOPRF	16	1	294	292	74	517	684	370
ABVOPRF	16	8	1910	2386	76	2135	2789	1981
ABVOPRF	16	64	15053	19196	80	15305	19702	15163

Table 1: Average operation time for various POPRF protocols. All times are measured in μs, and a zero time represents some value much less than a single microsecond.

6 Performance Evaluation

We implemented 3HashSDHI to measure the computational cost of the protocol in comparison to related protocols, including the baseline 2HashDH VOPRF from [DFHSW20], Pythia [ECS+15], and the recent attribute-based VOPRF (ABVOPRF) from Facebook [HIJ+21]. Each protocol was implemented in a minimal fashion, e.g., by omitting domain separating hash function invocations, in order to emphasize the cost of core public key operations. Our implementations use the ristretto255 group [dVGT+20] where prime-order groups are required, and the bn256 curve for Pythia, where pairing-friendly curves are required. We implemented each protocol in Go using the CIRCL experimental cryptographic library [FHK19] and `bn256` package. These benchmarks were evaluated on a machine with a 2.6 GHz 6-Core Intel Core i7 CPU and 32 GB RAM running macOS 10.15.7. Below we report on the average time over 1000 measurements of each operation.

Our benchmarks profile the functions needed to configure a client for the protocol, including the KeyGen and KeyVerify (for ABVOPRF), as well as the functions used to carry out the protocol, including Req, BlindEv, and Finalize. We also profiled the FullEvaluation routine to compare the cost of the blinding and proof verification operations in the protocol. For each protocol scheme, we keep the input size (L) fixed but vary the metadata size (T) in bytes. The results of our analysis are given in Table 1.

We comment on two key results in this data. First, the difference between the baseline VOPRF protocol and POPRF protocol are minuscule (as a function of metadata size). In particular, the POPRF introduces approximately a 25% overhead (in terms of μs to compute), though this is likely negligible at scale. Second, there is nearly an order of magnitude difference between the POPRF and the ABVOPRF as a function of metadata size. This is primarily due to the online KeyGen process and its resulting linear cost in proof evaluation. This suggests that the POPRF construction scales better in the presence of arbitrary-size tag values.

Some of the protocols included allow certain operations to be computed offline, thereby improving online protocol performance. For example, if the tag value is known in advance, 3HashSDHI can pre-compute the private key used in the BlindEv call. Likewise, in the ABVOPRF protocol, the key generation and verification operations can be computed offline and clients can cache the results. Table 2 summarizes the cost of operations, discounting precomputation costs. Note that although the performance of the ABVOPRF construction improves, it still introduces substantially more overhead than the 3HashSDHI construction.

7 Applications

POPRFs provide a new degree of flexibility that we observe to be useful in a variety of applications. Essentially anywhere an OPRF is used we see opportunity for POPRFs to provide potential benefits in terms of increasing deployment flexibility, reducing key management challenges, and/or improving security. Here we

Scheme	T (B)	Request	Evaluate	Finalize	Ev
2HashDH	1	73	223	392	77
2HashDH	8	75	231	403	79
2HashDH	64	83	255	447	89
3HashDH	1	84	367	526	124
3HashDH	8	76	332	478	112
3HashDH	64	76	328	472	110
Pythia	1	847	3913	6074	2832
Pythia	8	824	3935	6085	2840
Pythia	64	806	3764	5843	2715
ABVOPRF	1	74	224	685	77
ABVOPRF	8	76	228	2790	79
ABVOPRF	64	75	230	19721	80

Table 2: Average times for performance, excluding precomputation cost. All times are measured in μs.

briefly discuss three previously mentioned motivating applications: anonymous one-time-use tokens, password breach alerting, and password-based authenticated key exchange.

7.1 Privacy Pass

Privacy Pass [DGS+18, CDFH21] is a protocol in which users are allowed to receive one-time-use anonymous tokens, using an issuance protocol, that can later be used to anonymously authenticate themselves using a redemption protocol. It is often used in the context of fraud prevention online: tokens are issued to users that pass integrity challenges (e.g., CAPTCHA).

The primary component underlying Privacy Pass is a verifiable OPRF (VOPRF). Current implementations use 2HashDH [JKK14]. The issuance protocol has a client request VOPRF output for a random input x under a Privacy Pass server held VOPRF secret key sk. The client must verify that the received token $y = \mathsf{2HashDH.Ev}(sk, x)$ is correct relative to the server's public key pk. A token (x, y) can then be redeemed by sending to the server $(x, \mathsf{MAC}_y(data))$ where MAC is a message authentication code such as HMAC and $data$ is some application-specific bit string (called the binding data in [DGS+18]). The server can recompute y and then use it to check the MAC value.

The core security properties achieved by Privacy Pass are unlinkability and unforgeability. Unlinkability derives from the request privacy properties of the VOPRF, which ensures that a malicious server learns nothing about a client's input x nor can they link (x, y) to a particular issuance query. Unforgeability derives from the pseudorandomness security of the VOPRF: given the ability to obtain q tokens, the adversary can at most compute q outputs of the VOPRF.

However, one abuse of Privacy Pass not prevented by the current design is what we refer to as a hoarding attack, also called a farming attack [DGS+18]. A malicious user (or group of users) can gather a large number of tokens by running the token issuance protocol as many times as possible over some period of time. Later, the malicious user can redeem all the gathered tokens at once in an effort to render the provided service unavailable in a (D)DoS attack (by, for example, overwhelming a website with expensive requests).

One potential defense against hoarding attacks is to force periodic rotation of the VOPRF secret key, such as once per week. But this is clumsy because it requires clients to verify that key rotations are made legitimately: a server that rotates too often can violate unlinkability. In the limit, a malicious server could pick a separate pk for each client issuance, thereby completely violating unlinkability. Forcing clients to use gossip protocols (to verify that the same public keys are used) or using public ledgers to record public keys for monitoring purposes require further complicating infrastructure. Furthermore, keys are often stored and need to be deleted from secure storage locations (e.g., trusted hardware) and replaced with new ones. This process is prone to failure and can lead to potential leaks.

Use of a verifiable POPRF provides a simpler, more elegant solution. Tokens can be bound to a public input t that lets the server and client agree upon a scope for token issuance. In this case, one replaces the Privacy Pass' $\mathsf{2HashDH.Ev}(sk, x)$ with $\mathsf{DY.Ev}(sk, t, x)$ in both the issuance and redemption phases. One could use as t a coarse timestamp, such as the current day or week at which an issuance occurred, and then redemption could enforce a policy about the staleness of tokens, forcing them to be redeemed within some time period. Alternatively, one could imagine using t to bind issuance and redemption to the client's general network location, e.g., it's autonomous system number (ASN).

We note that all these hoarding mitigations reduce the anonymity set of a redemption to only the clients issued tokens under t. This is true also for the key rotation approach, reducing the anonymity set to clients that were issued tokens under the particular public key. Some loss of privacy is fundamental to restricted token use, and choosing how to make use of t requires care and further work to identify best practices. Nevertheless, POPRFs provide a degree of flexibility that easily allows a variety of choices.

7.2 Private Set Membership and Breach Alerting

One widely deployed use of OPRFs is for password breach alerting (also called compromised credential checking) [TPY+19,Hun]. These use an OPRF-based bucketized private set membership protocol [TPY+19, LPA+19] that works as follows. The server generates a long-lived 2HashDH secret key sk and computes $y_{u,pw} = \mathsf{2HashDH.Ev}(sk, u, pw)$ for each username, password pair $(u, pw) \in D$. Here D is a breach database of known-compromised pairs. Then, to perform a lookup for (u^*, pw^*), the client sends a truncated hash of the username that forms a bucket identifier $\beta = \mathsf{H}(u^*)$, as well as a blind evaluation request for the client's username, password pair (u^*, pw^*) that they want to check. The server computes the OPRF response and sends it back along with the bucket $B = \{y_{u,pw} \mid \mathsf{H}(u) = \beta\}$ of values that have matching truncated username hash. The client finishes computing $y^* = \mathsf{2HashDH.Ev}(sk, u^*, pw^*)$ and checks if $y^* \in B$. If so, their credential is known to be compromised; otherwise, it is not.

In the currently deployed protocols, there is no way to enforce that the client's query indicates the correct bucket identifier B. Instead, they could query for $b' = \mathsf{H}(u')$ for $u' \neq u^*$ and complete the protocol execution checking for values in some other bucket. Whether this opens up password breach alerting services to abuse is not clear.

Nevertheless, we observe that replacing the OPRF with a POPRF provides a simple way to cryptographically bind the buckets to particular bucket identifiers, by setting $t = \beta$. One could similarly do so using per-bucket secret keys sk_β for a standard OPRF, but this would complicate key management.

7.3 OPAQUE

As mentioned in the introduction, OPAQUE [JKX18] is a strong aPAKE protocol (SaPAKE) that provides password-based mutual authentication in a client-server setting without having to rely on Public Key Infrastructure (PKI). It provides security against pre-computation attacks upon server compromise, and increases the protection against offline dictionary attacks, as an attacker will have to perform an exhaustive per-user attack upon server's data compromise. OPAQUE is a protocol also amenable to a multi-server distributed implementation where an offline dictionary attack is only possible if a threshold of servers is compromised.

OPAQUE can be thought of as a protocol that works as a "compiler" by transforming a suitable AKE protocol (resistant to key compromise impersonation attacks and forward secrecy) into a secure aPAKE protocol using an OPRF. Current implementations use 2HashDH. It consists of two phases: an offline registration phase and an online authenticated key exchange phase.

The purpose of the offline phase is to register a user's account using their unique user identity and their password. The user identity could be a username or email address. To do so, the user opaquely registers its password without the server ever knowing it. The client and server obliviously compute $\mathsf{rwd} = \mathsf{2HashDH.Ev}(sk_u, x)$, where x is the user's password and sk_u is a server's random, per-user generated OPRF secret key. The output of this functionality, rwd, is used to encrypt its AKE private key. The resulting ciphertext is stored on the server-side alongside with other user's credentials, such as its corresponding user id. The server will store, as well, the per-user OPRF key sk_u used to generate rwd.

To perform online authentication, the client and server obliviously recompute rwd, enabling the client to recover their AKE private key by decrypting the ciphertext (sent back to the client during the OPRF flows), and complete a (now password-authenticated) AKE exchange.

A complexity for deployment of OPAQUE is that the server has to maintain and keep a consistent view of the per-user OPRF keys and associated AKE private key ciphertexts. Implementation vulnerabilities may arise should servers incorrectly use the same OPRF key across multiple users, allowing potentially for cross-user attacks that allow logging in as the wrong user. Implementations are also likely to store the per-user OPAQUE secret keys in the same user database alongside other per-user data, meaning that compromises that allow exfiltrating the database (e.g., SQL injection) will reveal all information needed to brute-force recover passwords.

One potential approach instead would be to, again, replace the per-user OPRF key sk_u with a POPRF with global sk and use the user identity as t. This would allow protecting sk by storing it in a separate hardened crypto service (similar to deployment models used in password hardening, see [ECS+15]) only once. One potential complication is that performing periodic key rotations for sk would be more challenging, since it would require somehow either resetting all users passwords (not reasonable in most contexts by asking users to reset) or rolling clients to new keys as they login by maintaining old sk for some period of time. In contrast, the per-user sk_u approach can selectively rotate OPRF keys for each user as needed.

Acknowledgments

The authors would like to thank Tjerand Silde and Martin Strand for discussions about their preprint.

References

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In *ASIACRYPT*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1996.

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2000.

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In *CRYPTO (2)*, volume 12171 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2020.

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. *IACR Cryptol. ePrint Arch.*, 2021:866, 2021.

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In *EUROCRYPT (1)*, volume 12696 of *Lecture Notes in Computer Science*, pages 33–53. Springer, 2021.

[BM93] Steven Bellovin and Michael Merritt. Augmented encrypted key exchange: a password based protocol secure against dictionary attacks and password file compromise. In *CCS*, pages 244–250. ACM, 1993.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptol.*, 16(3):185–215, 2003.

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

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[BS17] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2017. Version 0.4.

[Cam98] Jan Camenisch. *Group signature schemes and payment systems based on the discrete logarithm problem*. PhD thesis, ETH Zurich, Zürich, Switzerland, 1998.

[CDFH21] Sofia Celi, Alex Davidson, and Armando Faz-Hernndez. Privacy Pass Protocol Specification. Internet-Draft draft-ietf-privacypass-protocol-00, Internet Engineering Task Force, January 2021. Work in Progress.

[Che06] Jung Hee Cheon. Security analysis of the strong diffie-hellman problem. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *CCS*, pages 1205–1216. ACM, 2014.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In *CCS*, pages 1445–1459. ACM, 2020.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

[DFHSW20] Alex Davidson, Armando Faz-Hernndez, Nick Sullivan, and Christopher A. Wood. Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups. Internet-Draft draft-irtf-cfrg-voprf-05, Internet Engineering Task Force, November 2020. Work in Progress.

[DGS$^+$18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proc. Priv. Enhancing Technol.*, 2018(3):164–180, 2018.

[dVGT$^+$20] Henry de Valence, Jack Grigg, George Tankersley, Filippo Valsorda, Isis Lovecruft, and Mike Hamburg. The ristretto255 and decaf448 Groups. Internet-Draft draft-irtf-cfrg-ristretto255-decaf448-00, Internet Engineering Task Force, October 2020. Work in Progress.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[ECS$^+$15] Adam Everspaugh, Rahul Chatterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The pythia PRF service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562. USENIX Association, 2015.

[FHK19] Armando Faz-Hernndez and Kris Kwiatkowski. *Introducing CIRCL: An Advanced Cryptographic Library*. Cloudflare, June 2019. https://github.com/cloudflare/circl.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2006.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *EUROCRYPT (2)*, volume 12106 of *Lecture Notes in Computer Science*, pages 63–95. Springer, 2020.

[HIJ$^+$21] Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei Lee, Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh Sung, and Albert Zhang. PrivateStats: De-Identified Authenticated Logging at Scale, January 2021.

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 345–375. Springer, 2019.

[Hun] Troy Hunt. Have i been pwned. https://haveibeenpwned.com/.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 2014.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486. Springer, 2018.

[JKX21] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. On the (in)security of the diffie-hellman oblivious PRF with multiplicative blinding. In *Public Key Cryptography (2)*, volume 12711 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2021.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2009.

[JT20] Joseph Jaeger and Nirvan Tyagi. Handling adaptive compromise for practical encryption schemes. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2020.

[KLOR20] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 308–336. Springer, 2020.

[KLW21] Hugo Krawczyk, Kevin Lewi, and Christopher A. Wood. The OPAQUE Asymmetric PAKE Protocol. Internet-Draft draft-irtf-cfrg-opaque-02, Internet Engineering Task Force, February 2021. Work in Progress.

[KZ08] Aggelos Kiayias and Hong-Sheng Zhou. Equivocal blind signatures and adaptive uc-security. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

[LPA$^+$19] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *CCS*, pages 1387–1403. ACM, 2019.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[MPR$^+$20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467. IEEE Computer Society, 1997.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[SS21] Tjerand Silde and Martin Strand. Anonymous tokens with public metadata and applications to private contact tracing. *IACR Cryptol. ePrint Arch.*, 2021:203, 2021.

[TPY$^+$19] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security Symposium*, pages 1556–1571. USENIX Association, 2019.

[WTKW20] John Wilander, Erik Taubeneck, Andrew Knox, and Chris Wood. Consider using blinded signatures for fraud prevention - Private Click Measurement, 2020. https://github.com/privacycg/private-click-measurement/issues/41.

[ZSS03] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In *INDOCRYPT*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2003.

[ZSS04] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 277–290. Springer, 2004.

A Security of (m, n)-OM-Gap-SDHI

Proof of Of Theorem 2: This proof proceeds in two parts. (1) First, we present $\mathcal{A}_{\text{uber}}$ and argue that it almost perfectly simulates the (m, n)-OM-Gap-SDHI game for $\mathcal{A}_{\text{sdhi}}$, up to a small statistical difference. (2) Second, we argue that if $\mathcal{A}_{\text{sdhi}}$ outputs a set of values that wins the (m, n)-OM-Gap-SDHI game, it must be that at least one of those values has a non-trivial representation in the exponent that wins the m-Uber game.

A.1 Simulation of (m, n)-OM-Gap-SDHI environment

We construct adversary $\mathcal{A}_{\text{uber}}$ as shown in Figure 9. The adversary first constructs a polynomial $G(\vec{X}) \leftarrow \prod_i^n (X_0 + c_i)^{q_i}$ from the chosen c values in the first stage of $\mathcal{A}_{\text{sdhi}}$. The adversary receives an evaluation $\hat{g} = g^{G(\vec{x})}$ from its Ev oracle that it will pass as the generator in its simulation to the second stage of $\mathcal{A}_{\text{sdhi}}$. The public key $\hat{g}^{x_0}$ is simulated by evaluating $X_0 \cdot G(\vec{X})$; and the m challenge points will each be simulated with additional variables X_i for $i \in [1, m]$ by evaluating polynomial $X_i \cdot G(\vec{X})$.

To simulate the SDH oracle on input (Y, i), first assume that $\mathcal{A}_{\text{uber}}$ knows the polynomial $P(\vec{X})$ such that $Y = g^{P(\vec{x})}$. We will show shortly how $\mathcal{A}_{\text{uber}}$ can compute $P(\vec{X})$ from the algebraic representation of Y. Given $P(\vec{X})$, $\mathcal{A}_{\text{uber}}$ simulates by computing the polynomial $P(\vec{X})/(X_0 + c_i)$ and returning its evaluation. Now to compute $P(\vec{X})$, $\mathcal{A}_{\text{uber}}$ takes the algebraic representation of Y in terms of elements given to $\mathcal{A}_{\text{sdhi}}$. The elements given to $\mathcal{A}_{\text{sdhi}}$ are the initial elements plus the elements output from previous queries to SDH. The polynomial exponents of all of these elements are known to $\mathcal{A}_{\text{uber}}$, so they can be combined via the linear combination indicated by the algebraic representation to compute $P(\vec{X})$. Furthermore, $P(\vec{X})/(X_0 + c_i)$ will always be computable due to the construction of $G(\vec{X})$ including q_i factors of $(X_0 + c_i)$ where q_i is the maximum number of queries for c_i to SDH. All initial elements have the polynomial $G(\vec{X})$ as a factor of the exponent, and while subsequent elements returned from SDH divide out different factors of $(X_0 + c_j)$, they

```
Adversary $\mathcal{A}_{\text{uber}}^{\text{Ev},\text{Decide}}(p, \mathbb{G}, g)$
    $(st_{\mathcal{A}}, [c_i]_i^n) \leftarrow\$ \mathcal{A}_1(p, \mathbb{G})$
    $G(\vec{X}) \leftarrow \prod_i^n (X_0 + c_i)^{q_i}$
    $\hat{g} \leftarrow \text{Ev}(G(\vec{X}))$ ; $\hat{X} \leftarrow \text{Ev}(X_0 \cdot G(\vec{X}))$
    $[\hat{A}]_{i=1}^m \leftarrow [\text{Ev}(X_i \cdot G(\vec{X}))]_{i=1}^m$
    $(\gamma, [Z_i, \alpha_i]_i^\ell) \leftarrow\$ \mathcal{A}_2^{\text{SDH},\text{SDDH}}(\hat{g}, \hat{X}, [\hat{A}]_i^m : st_{\mathcal{A}})$
    $j \leftarrow\$ \mathbb{N}_\ell$ ; $\mu(\vec{X}) = X_{\alpha_j}(X_0 + c_\gamma)^{q_\gamma - 1} \prod_{i \neq \gamma}^n (X_0 + c_i)^{q_i}$
    Return $(Z_j, \mu(\vec{X}))$

Oracle SDH$(Y, i)$
    Compute $P(\vec{X}) : Y = g^{P(\vec{x})}$ from representation of $Y$
    $Z \leftarrow \text{Ev}(P(\vec{X})/(X_0 + c_i))$
    Return $Z$

Oracle SDDH$(Y, Z, i)$
    $P(A_1, A_2, A_3) \leftarrow (A_1 + c_i) \cdot A_3 - A_2$
    Return Decide$(P(A_1, A_2, A_3), [\hat{X}, Y, Z])$
```

Figure 9: Adversary $\mathcal{A}_{\text{uber}}$ in the security proof of (m,n)-OM-Gap-SDHI.

never exhaust a factor $(X_0 + c_j)$ unless the maximum number of queries for that index has been reached. Lastly, $\mathcal{A}_{\text{uber}}$ simulates SDDH by passing the query on to its own Decide oracle.

As long as the maximum query counts that $\mathcal{A}_{\text{uber}}$ uses to create $G(\vec{X})$ are abided by, the environment for $\mathcal{A}_{\text{sdhi}}$ is almost perfectly simulated, as we argue below. Moreover, $\mathcal{A}_{\text{uber}}$ selects an output of $\mathcal{A}_{\text{sdhi}}$ at random and returns it along with its polynomial representation.

Concretely, let π_0 be the probability that at least one of the values in the output of $\mathcal{A}_{\text{sdhi}}$ has a non-trivial representation in the exponent that wins the m-Uber game in the original game (m,n)-OM-Gap-SDHI. Let π_1 be the probability that the same happens within the simulation of $\mathcal{A}_{\text{uber}}$. Then, we argue that

$$\pi_0 - \pi_1 \leq \frac{q}{p} \,.$$

This can be seen as follows: The only difference between the simulation and the original game is that the former uses $(\hat{g}, \hat{X})$, where $\hat{g} = g^{G(x)}$ and $\hat{X} = g^{x \cdot G(x)}$ for a $x \leftarrow\$ \mathbb{Z}_p$ and a polynomial $G(X)$ of degree q, instead of (g, g^x) in the latter, where g is a random generator. In particular, $\pi_0 - \pi_1$ is upper bounded by the statistical distance between $(y \cdot G(x), y \cdot x \cdot G(x))$ and $(y, y \cdot x)$, where $(x, y) \leftarrow\$ \mathbb{Z}_p^2$. Now, let $S \subseteq \mathbb{Z}_p^2$ be the set pairs (x, y) such that $G(x) \neq 0$. For any $(z_1, z_2) \in \mathbb{Z}_p^2$, and $(x, y) \leftarrow\$ S$, we now have

$$\begin{aligned}\Pr[\,(y, y \cdot x) = (z_1, z_2)\,] &= \Pr[\,(y, z_1 \cdot x) = (z_1, z_2)\,] \\ &= \Pr[\,(y \cdot G(x), z_1 \cdot x) = (z_1, z_2)\,] = \Pr[\,(y \cdot G(x), y \cdot x \cdot G(x)) = (z_1, z_2)\,] \,.\end{aligned}$$

Therefore, the statistical distance is upper bounded by the probability that $(x, y) \leftarrow\$ \mathbb{Z}_p$ is in S, which in turn is the probability that $G(x) = 0$. The latter is at most q/p by the Schwartz-Zippel Lemma.

In conclusion, the advantage of $\mathcal{A}_{\text{uber}}$ is

$$\mathsf{Adv}_{\mathsf{GGen},\mathcal{A}_{\text{uber}}}^{(m+1)\text{-uber}}(\lambda) \geq \frac{\pi_1}{\ell} \geq \frac{\pi_1}{q_{\max} + 1} \geq \frac{\pi_0}{q_{\max} + 1} - \frac{q}{p(q_{\max} + 1)} \,,$$

because, without loss of generality, we can assume that $\ell \leq q_{\max} + 1$. Below, we are going to prove that $\mathsf{Adv}_{\mathsf{GGen},\mathcal{A}_{\text{sdhi}}}^{(m,n)\text{-om-gap-sdhi}}(\lambda) = \pi_0$, which concludes the proof.

A.2 Linear independence of winning elements

Next, we argue that one of the strong Diffie-Hellman values output by a winning $\mathcal{A}_{\text{sdhi}}$ corresponds to a group element with a non-trivial polynomial in the exponent, i.e., a polynomial that is linearly independent from all queried polynomials. Our ultimate goal will be to claim that if the adversary produces ℓ winning group elements for a value c_γ having only queried SDH $q_\gamma < \ell$ times, then one of the winning elements must have a non-trivial (linearly-independent) polynomial exponent from all group elements given to the adversary during initialization and as output from SDH.

We will find that it is easy to argue the linear independence of the initial group elements and winning elements. The main challenge we will face is reasoning about elements output from SDH. In previous formulations of one-more assumptions [Bol03], the number of winning elements was required to be greater than the number of queries to the "one-more" oracle. In these cases, the typical proof strategy is show linear independence of the set of winning elements and initial elements – since this set is strictly larger than the set of query elements and initial elements, it must be that at least one winning element is linearly independent of the set of query elements and initial elements. However, this strategy will not work for OM-Gap-SDHI, which simply requires the number of winning elements to be greater than the number of queries *for the chosen* winning c_γ.

Intuitively, we would like to say that elements returned from SDH for other $c_{j\neq\gamma}$ will not be helpful in constructing a winning element for c_γ. If this is the case, then we can conclude by arguing that a set of $q_\gamma < \ell$ elements cannot construct a set of ℓ linearly-independent winning elements.

Unfortunately, it is not immediately clear this is the case, as elements that were previously output for queries to c_γ may be passed back in to SDH under a different c_j. This leads to the possibility that many elements returned from SDH ($> q_\gamma$) may have a "dependence" on c_γ. The main step in our proof shows that whenever a query to SDH on c_i is made, the output element can be refactored as an element that only depends on the queried c_i, regardless of whether the input element includes previous outputs from SDH dependent on other $c_{j\neq i}$.

Rewriting transcript to separate c_i dependence in SDH queries. First, we consider the polynomial exponent representation of group elements output from SDH given only queries to a single index c_i. We will notate the exponent in its quotient form (note that the simulation multiplies all polynomial exponents by a least common multiple to remove quotients). We denote by τ_i the i^{th} output from SDH in which the input may be a linear combination of any previous $\tau_{j<i}$ and initial values $X_1, \ldots, X_m$, where we denote coefficients with a. In the below we denote $\mathcal{Y}_i$ as arbitrary linear combinations of the formal variables $X_0, X_1, \ldots, X_m$ along with a constant. Without loss of generality, we will denote $c_i = c_1$.

$$
\begin{aligned}
\tau_1 &= \frac{\mathcal{Y}_1}{X_0 + c_1} && \text{(binary string representation: } \texttt{1}) \\
\tau_2 &= \frac{\mathcal{Y}_2 + a_{2,1}\tau_1}{X_0 + c_1} = \frac{a_{2,1}\mathcal{Y}_1}{(X_0 + c_1)^2} + \frac{\mathcal{Y}_2}{X_0 + c_1} && (\texttt{11, 10}) \\
\tau_3 &= \frac{\mathcal{Y}_3 + a_{3,2}\tau_2 + a_{3,1}\tau_1}{X_0 + c_1} \\
&= \frac{a_{3,2}a_{2,1}\mathcal{Y}_1}{(X_0 + c_1)^3} + \frac{a_{3,2}\mathcal{Y}_2}{(X_0 + c_1)^2} + \frac{a_{3,1}\mathcal{Y}_1}{(X_0 + c_1)^2} + \frac{\mathcal{Y}_3}{X_0 + c_1} && (\texttt{111, 110, 101, 100}) \\
\tau_4 &= \frac{\mathcal{Y}_4 + a_{4,3}\tau_3 + a_{4,2}\tau_2 + a_{4,1}\tau_1}{X_0 + c_1} \\
&= \frac{a_{4,3}a_{3,2}a_{2,1}\mathcal{Y}_1}{(X_0 + c_1)^4} + \frac{a_{4,3}a_{3,2}\mathcal{Y}_2}{(X_0 + c_1)^3} + \frac{a_{4,3}a_{3,1}\mathcal{Y}_1}{(X_0 + c_1)^3} + \frac{a_{4,3}\mathcal{Y}_3}{(X_0 + c_1)^2} && (\texttt{1111, 1110, 1101, 1100}) \\
&\quad + \frac{a_{4,2}a_{2,1}\mathcal{Y}_1}{(X_0 + c_1)^3} + \frac{a_{4,2}\mathcal{Y}_2}{(X_0 + c_1)^2} + \frac{a_{4,1}\mathcal{Y}_1}{(X_0 + c_1)^2} + \frac{\mathcal{Y}_4}{X_0 + c_1} && (\texttt{1011, 1010, 1001, 1000})
\end{aligned}
$$

We observe that each term in τ_i can be interpreted interpreted uniquely as mapping from a binary string. This leads us to the following closed form expression. We define $\omega(s)$ to take a positive integer s and return the list of indices at which the binary string representation of s has a 1. For example, $6 = \texttt{110}$ has $\omega(6) = [3, 2]$ (we use 1-indexing and reverse the list to make the expression below easier to parse). We define the mapping of positive integer s to term in τ_i as $\Omega(s)$ (defined below). The coefficients for the term are determined by the locations of the 1-bits in the binary string and the power of the denominator is determined by the number of 1-bits in the string. The $\mathcal{Y}_j$ value is determined by the least significant 1-bit set in the string. Intuitively, the locations of the 1-bits correspond to which queries to SDH the term has been passed (and repassed) into.

$$\Omega(s) = \begin{cases} \dfrac{\mathcal{Y}_{\lg(s)+1}}{X_0 + c_1} & \text{if } \exists\, i \text{ s.t. } s = 2^i \\ \dfrac{\mathcal{Y}_{\omega(s)_{|\omega(s)|}} \cdot \prod_{k=1}^{|\omega(s)|-1} a_{\omega(s)_k, \omega(s)_{k+1}}}{(X_0 + c_1)^{|\omega(s)|}} & \text{o.w.} \end{cases} \tag{1}$$

Then, we observe that the terms included in τ_i correspond to the terms for binary strings 2^{i-1} to $2^i - 1$. We get the following expression for τ_i based on the above interpretation:

$$\begin{aligned} \tau_i &= \frac{\mathcal{Y}_i + \sum_{j=1}^{i-1} a_{i,j}\tau_j}{X_0 + c_1} \\ &= \sum_{s=2^{i-1}}^{2^i - 1} \Omega(s) \end{aligned} \tag{2}$$

The above form follows from an induction argument on the form of all $\tau_{j<i}$. Multiplying τ_j by $a_{i,j}/(X_0 + c_1)$ corresponds exactly to transforming all $\Omega(s')$ for $s' \in [2^{j-1}, 2^j - 1]$ to $\Omega(s)$ for $s \in [2^{i-1} + 2^{j-1}, 2^{i-1} + 2^j - 1]$. Summing up the terms performing this transformation on all $\tau_{j<i}$ corresponds to all $s \in [2^{i-1} + 1, 2^i - 1]$, and the final term corresponding to $s = 2^{i-1}$ is added separately.

Lastly, we rearrange the terms of τ_i grouping them by $\mathcal{Y}_j$ which will be useful later on:

$$\begin{aligned} \tau_i &= \sum_{s=2^{i-1}}^{2^i - 1} \Omega(s) \\ &= \Omega(2^{i-1}) + \sum_{j=1}^{i-1} \sum_{s=2^{i-j-1}}^{2^{i-j}-1} \Omega(2^j * s + 2^{j-1}) \\ &= \frac{\mathcal{Y}_i}{X_0 + c_1} + \sum_{j=1}^{i-1} \sum_{s=2^{i-j-1}}^{2^{i-j}-1} \frac{\mathcal{Y}_j \cdot \prod_{k=1}^{|\omega(2^j * s + 2^{j-1})|-1} a_{\omega(2^j * s + 2^{j-1})_k, \omega(2^j * s + 2^{j-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^j * s + 2^{j-1})|}} \end{aligned} \tag{3}$$

Next, we consider a query made to a different $c_{j \neq 1}$, following a transcript of m queries $\tau = [\tau_i]_{i=1}^m$ made to c_1. Without loss of generality, we will denote $c_j = c_2$. This $(m+1)^{th}$ query output takes the following form:

$$\hat{\tau}_{m+1} = \frac{\mathcal{Y}_{m+1} + \sum_{i=1}^m a_{m+1,i}\tau_i}{X_0 + c_2}$$

The above is the same as τ_{m+1} (Equation 2) except the numerator is divided by $(X_0 + c_2)$ instead of $(X_0 + c_1)$. Thus, we can rewrite using the same binary string notation (again, grouped by $\mathcal{Y}_j$):

$$\begin{aligned} \hat{\tau}_{m+1} &= \frac{\mathcal{Y}_{m+1} + \sum_{i=1}^m a_{m+1,i}\tau_i}{X_0 + c_2} = \frac{X_0 + c_1}{X_0 + c_2} \cdot \tau_{m+1} \\ &= \frac{X_0 + c_1}{X_0 + c_2} \left(\sum_{s=2^{i-1}}^{2^i - 1} \Omega(s) \right) \\ &= \frac{X_0 + c_1}{X_0 + c_2} \left(\Omega(2^{i-1}) + \sum_{j=1}^{i-1} \sum_{s=2^{i-j-1}}^{2^{i-j}-1} \Omega(2^j * s + 2^{j-1}) \right) \\ &= \frac{\mathcal{Y}_{m+1}}{X_0 + c_2} + \sum_{j=1}^{m} \sum_{s=2^{m-j}}^{2^{m-j+1}-1} \frac{\mathcal{Y}_j \cdot \prod_{k=1}^{|\omega(2^j * s + 2^{j-1})|-1} a_{\omega(2^j * s + 2^{j-1})_k, \omega(2^j * s + 2^{j-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^j * s + 2^{j-1})|-1}(X_0 + c_2)} \end{aligned}$$

The query output $\hat{\tau}_{m+1}$ is a sum of terms with mixed $(X_0 + c_1)$ and $(X_0 + c_2)$ factors in the denominator. We will show that $\hat{\tau}_{m+1}$ to τ'_{m+1} of the form:

$$\tau'_{m+1} = \frac{\mathcal{Y}_{m+1} + \sum_{i=1}^{m} b_i \mathcal{Y}_i}{X_0 + c_2}\,, \tag{4}$$

for some set of coefficients $b_1, \ldots, b_m$, such that the span of the query outputs is preserved:

Claim 8 *We provide $[b_i]_{i=1}^m$ to construct τ'_{m+1} (Equation 4) such that*

$$\mathrm{Span}([\tau_1, \ldots, \tau_m, \hat{\tau}_{m+1}]) = \mathrm{Span}([\tau_1, \ldots, \tau_m, \tau'_{m+1}])\,.$$

Proof of Claim: We choose $[b_i]_{i=1}^m$ such that $\hat{\tau}_{m+1} \in \mathrm{Span}([\tau_1, \ldots, \tau_m, \tau'_{m+1}])$. This is sufficient to complete the claim that the two spans are equivalent. We solve the following system of equations:

$$\begin{aligned}\hat{\tau}_{m+1} &= \alpha_{m+1}\tau'_{m+1} + \sum_{i=1}^{m} \alpha_i \tau_i \\ &= \sum_{i=1}^{m+1} \beta_i \frac{\mathcal{Y}_i}{X_0 + c_2} + \sum_{i=1}^{m} \alpha_i \tau_i \end{aligned} \tag{5}$$

for unknowns $\alpha_1, \ldots, \alpha_{m+1}, b_1, \ldots, b_m$, or equivalently, when reformulated, for unknowns $\alpha_1, \ldots, \alpha_m$, β_1, $\ldots$, β_{m+1}, where $\beta_{m+1} = \alpha_{m+1}$ and $\beta_{i \neq m+1} = \alpha_{m+1} b_i$. Now consider the expanded Equation 5:

$$\begin{aligned}&\frac{\mathcal{Y}_{m+1}}{X_0 + c_2} + \sum_{j=1}^{m} \sum_{s=2^{m-j}}^{2^{m-j+1}-1} \frac{\mathcal{Y}_j \cdot \prod_{k=1}^{|\omega(2^j * s + 2^{j-1})|-1} a_{\omega(2^j*s+2^{j-1})_k, \omega(2^j*s+2^{j-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^j*s+2^{j-1})|-1}(X_0 + c_2)} \\ &\quad = \sum_{i=1}^{m+1} \beta_i \frac{\mathcal{Y}_i}{X_0 + c_2} + \sum_{i=1}^{m} \alpha_i \left(\frac{\mathcal{Y}_i}{X_0 + c_1} + \sum_{j=1}^{i-1} \sum_{s=2^{i-j-1}}^{2^{i-j}-1} \frac{\mathcal{Y}_j \cdot \prod_{k=1}^{|\omega(2^j*s+2^{j-1})|-1} a_{\omega(2^j*s+2^{j-1})_k, \omega(2^j*s+2^{j-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^j*s+2^{j-1})|}} \right)\end{aligned}$$

One approach for solving this equation is by solving each of the partial equations, equating the coefficients for a particular $\mathcal{Y}_i$. The partial equation for $\mathcal{Y}_{m+1}$ is trivial and is easily solvable by setting $\beta_{m+1} = 1$:

$$\frac{\mathcal{Y}_{m+1}}{X_0 + c_2} = \beta_{m+1} \frac{\mathcal{Y}_{m+1}}{X_0 + c_2}$$

The other partial equations for $1 \le i \le m$ are more complex. We refer to the partial equation for $\mathcal{Y}_i$ as PE_i, and it is defined as follows:

$$\begin{aligned}&\sum_{s=2^{m-i}}^{2^{m-i+1}-1} \frac{\mathcal{Y}_i \cdot \prod_{k=1}^{|\omega(2^i*s+2^{i-1})|-1} a_{\omega(2^i*s+2^{i-1})_k, \omega(2^i*s+2^{i-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^i*s+2^{i-1})|-1}(X_0 + c_2)} \\ &\quad = \frac{\beta_i \mathcal{Y}_i}{X_0 + c_2} + \frac{\alpha_i \mathcal{Y}_i}{X_0 + c_1} + \sum_{j=i+1}^{m} \sum_{s=2^{j-i-1}}^{2^{j-i}-1} \frac{\mathcal{Y}_i \cdot \alpha_j \cdot \prod_{k=1}^{|\omega(2^i*s+2^{i-1})|-1} a_{\omega(2^i*s+2^{i-1})_k, \omega(2^i*s+2^{i-1})_{k+1}}}{(X_0 + c_1)^{|\omega(2^i*s+2^{i-1})|}}\end{aligned}$$

Note that $\mathcal{Y}_i$ can be canceled out in PE_i. We next show that there exists (and that we can solve for) a satisfying assignment of variables $[\alpha_i]_{i=1}^m$ and $[\beta_i]_{i=1}^{m+1}$ for Equation 5 through a strong induction argument on the satisfiability of the partial equations.

Induction hypothesis. $H(i)$: There exists a satisfying assignment of variables $[\alpha_j]_{j=i}^m$ and $[\beta_j]_{j=i}^{m+1}$ for the system of equations $PE_m, PE_{m-1}, \ldots, PE_i$.

Note that $H(1)$ implies that Equation 5 is satisfiable. We proceed by proving the base case, $H(m)$, and then proving the induction step showing that $H(i+1) \Rightarrow H(i)$.

Base case. We prove $H(m)$. Consider PE_m:

$$\frac{\beta_m}{X_0 + c_2} + \frac{\alpha_m}{X_0 + c_1} = \frac{a_{m+1,m}}{(X_0 + c_1)(X_0 + c_2)}$$

We solve PE_m as follows:

$$\begin{aligned}
\beta_m(X_0 + c_1) + \alpha_m(X_0 + c_2) &= a_{m+1,m} \\
\beta_m + \alpha_m &= 0 \\
\beta_m c_1 + \alpha_m c_2 &= a_{m+1,m} \\
\beta_m &= \frac{a_{m+1,m}}{c_1 - c_2} \\
\alpha_m &= \frac{a_{m+1,m}}{c_2 - c_1}
\end{aligned}$$

Induction step. We prove $H(i + 1) \Rightarrow H(i)$. Consider the left hand side of PE_i. It contains terms for $(m + 1)$-length binary strings, where `*` is a wildcard for either `0/1`:

$$\mathtt{1} \parallel *^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$$

Now consider the left hand side of PE_j for $j > i$. It contains terms for the following binary strings:

$$\mathtt{1} \parallel *^{m-j} \parallel \mathtt{1} \parallel \mathtt{0}^{j-i} \parallel \mathtt{0}^{i-1}$$

We observe that if we flip the i^{th} least significant bit of the binary strings from PE_j, we obtain a subset of binary strings from PE_i. In fact, if we do this for all PE_j for $m \geq j > i$, we obtain all binary strings from PE_i except for $\mathtt{1} \parallel \mathtt{0}^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$.

A similar cancellation occurs on the right hand side of the equations. The right hand side of PE_i contains terms for the m-length binary strings:

$$*^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$$

And the right hand side of PE_j for $m \geq j > i$ contains terms for the m-length binary strings:

$$*^{m-j} \parallel \mathtt{1} \parallel \mathtt{0}^{j-i} \parallel \mathtt{0}^{i-1}$$

Again, we observe that if we flip the i^{th} least significant bit of binary strings for all PE_j, we obtain all binary strings from PE_i except for $\mathtt{0}^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$.

This leads us to the following approach. We sum up all PE_j for $m \geq j > i$, transforming each one appropriately to "flip the i^{th} bit" of all represented binary strings. This sum can then be subtracted from PE_i to cancel out all terms on the left hand side of the equation except for the term corresponding to $\mathtt{1} \parallel \mathtt{0}^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$, and most terms on the right hand side of the equation except for the term corresponding to $\mathtt{0}^{m-i} \parallel \mathtt{1} \parallel \mathtt{0}^{i-1}$ and a set of terms $\beta_u(X_0 + c_1)^{m-i+1}$ in each PE_u that does not correspond to a binary string.

We create the summed equation:

$$\sum_{j=i+1}^{m} \frac{a_{j,i}}{X_0 + c_1} \cdot PE_j \tag{6}$$

The following equation is what remains after taking the difference of PE_i and summed Equation 6:

$$\frac{a_{m+1,i}}{(X_0 + c_1)(X_0 + c_2)} = \frac{\alpha_i}{X_0 + c_1} + \frac{\beta_i}{X_0 + c_2} + \sum_{j=i+1}^{m} \frac{\beta_j a_{j,i}}{(X_0 + c_1)(X_0 + c_2)}$$

By induction hypothesis $H(i+1)$, we have that there exists a satisfying assignment for β_j for $j > i$. We fix those variables, then solve the above equation for α_i and β_i:

$$\begin{aligned}
\beta_i(X_0+c_1)+\alpha_i(X_0+c_2) &= a_{m+1,i} - \sum_{j=i+1}^{m} \beta_j a_{j,i} \\
\beta_i + \alpha_i &= 0 \\
\beta_i c_1 + \alpha_i c_2 &= a_{m+1,i} - \sum_{j=i+1}^{m} \beta_j a_{j,i} \\
\beta_i &= \frac{a_{m+1,i} - \sum_{j=i+1}^{m} \beta_j a_{j,i}}{c_1 - c_2} \\
\alpha_i &= \frac{a_{m+1,i} - \sum_{j=i+1}^{m} \beta_j a_{j,i}}{c_2 - c_1}
\end{aligned}$$

This concludes proof of the induction hypothesis and of the claim.

Generalizing transcript rewrite to all c_i. Next, we argue that the above claim, which holds for a transcript of queries to c_1 followed by one query to c_2, generalizes to a transcript that makes queries to arbitrary different c_i values. We assume the transcript up until this point is made up of elements that are "separated by c_j", i.e., each only have powers of a single $(X_0 + c_j)$ in the denominator. Given this, we show that a new query to any c_v (taking in a linear combination of all previous elements of the transcript) can be rewritten so that it too only depends on powers of $(X_0 + c_v)$, such that the span of the full transcript is preserved.

As such, we assume the transcript has elements of the following form, where a transcript element $\tau_{i,j}$ is the i^{th} query to c_j (following from Equation 2 and 3). Element $\tau_{i,j}$ depends only on previous $\tau_{1,j}, \ldots, \tau_{i-1,j}$ and all only have powers of $(X_0 + c_j)$ in the denominator:

$$\begin{aligned}
\tau_{i,j} &= \frac{\mathcal{Y}_{i,j} + \sum_{u=1}^{i-1} a_{u,j}\tau_{u,j}}{X_0 + c_j} \\
&= \frac{\mathcal{Y}_{i,j}}{X_0 + c_j} + \sum_{u=1}^{i-1} \sum_{s=2^{i-u-1}}^{2^{i-u}-1} \frac{\mathcal{Y}_{u,j} \cdot \prod_{k=1}^{|\omega(2^u * s + 2^{u-1})|-1} a_{\omega(2^u*s+2^{u-1})_k, \omega(2^u*s+2^{u-1})_{k+1}}}{(X_0 + c_j)^{|\omega(2^u*s+2^{u-1})|}}
\end{aligned} \tag{7}$$

A new query to c_v will have the following output form, where q_j denotes the number of queries that have been made to c_j:

$$\tau'_{q_v+1,v} = \frac{\mathcal{Y}'_{q_v+1,v} + \sum_{j=1}^{n} \sum_{i=1}^{q_j} a'_{q_v+1,v,i,j}\tau_{i,j}}{X_0 + c_v}$$

We show that we can replace this output with $\tau_{q_v+1,v}$ of the following form which matches the form from Equation 7:

$$\begin{aligned}
\tau_{q_v+1,v} &= \frac{\mathcal{Y}'_{q_v+1,v} + \sum_{\substack{j=1 \\ j\neq v}}^{n} \sum_{i=1}^{q_j} b_{i,j}\mathcal{Y}_{i,j} + \sum_{i=1}^{q_v} a'_{q_v+1,v,i,v}\tau_{i,v}}{X_0 + c_v} \\
&= \frac{\mathcal{Y}_{q_v+1,v} + \sum_{i=1}^{q_v} a_{i,v}\tau_{i,v}}{X_0 + c_v} \\
\text{where } \mathcal{Y}_{q_v+1,v} &= \mathcal{Y}'_{q_v+1,v} + \sum_{\substack{j=1 \\ j\neq v}}^{n} \sum_{i=1}^{q_j} b_{i,j}\mathcal{Y}_{i,j} \\
a_{i,v} &= a'_{q_v+1,v,i,v}
\end{aligned}$$

We want that the span of the new transcript is preserved by replacing the new output with the rewritten output, and thus we prove the following:

Claim 9 *We provide* $\big[[b_i]_{i=1}^{q_j}\big]_{j=1}^{n}$ *to construct* $\tau_{q_v+1,v}$ *such that*

$$\text{Span}\Big(\Big[\big[[\tau_{i,j}]_{i=1}^{q_j}\big]_{j=1}^{n}, \tau'_{q_v+1,v}\Big]\Big) = \text{Span}\Big(\Big[\big[[\tau_{i,j}]_{i=1}^{q_j}\big]_{j=1}^{n}, \tau_{q_v+1,v}\Big]\Big)$$

Proof of Claim: We prove the claim by providing $\alpha_{i,j}$ and $b_{i,j}$ values such that:

$$\tau'_{q_v+1,v} = \alpha_{q_v+1,v}\tau_{q_v+1,v} + \sum_{j=1}^{n}\sum_{i=1}^{q_j}\alpha_{i,j}\tau_{i,j}$$

This is implied by our previous claim which shows that for each $j \neq v$, we can find $\alpha_{i,j}$ and $b_{i,j}$ such that:

$$\sum_{i=1}^{q_j}\frac{a_{q_v+1,v,i,j}\tau_{i,j}}{X_0+c_v} = \alpha_{q_v+1,v}\left(\sum_{i=1}^{q_j}\frac{b_{i,j}\mathcal{Y}_{i,j}}{X_0+c_v}\right) + \sum_{i=1}^{q_j}\alpha_{i,j}\tau_{i,j}$$

This concludes the argument that any new query output can be rewritten to be in the form of Equation 7 in which it only has powers of the queried c_v in the denominator, while preserving the span of the query output transcript.

Using rewritten transcript to show non-trivial winning element. We will continue to use the quotient notation, however, recall that in the simulation of the adversary's environment (Section A.1), the rational fractions are multiplied by a least common multiple, $\prod_{i=1}^{n}(X_0+c_i)^{q_i}$. The notions of independence we show for polynomial rational fractions hold true for the the polynomials once multiplied by the LCM as well.

The adversary is given initial group elements which we represent as polynomial rational fractions as follows: $\hat{g} \mapsto 1$, $\hat{X} \mapsto X_0$, $[A_i]_{i=1}^{m} \mapsto [X_i]_{i=1}^{m}$. It is evident that these polynomials are linearly-independent as they each include a different formal variable (with exception of $\hat{g}$ which is the only element with a constant).

The adversary will also receive group elements from the SDH oracle. We will denote the polynomial rational fraction outputs of the $q = \sum_{i=1}^{n} q_i$ queries to SDH as $[\tau'_i]_{i=1}^{q}$.

Lastly, the adversary will output a set of ℓ winning group elements for a selected γ, $[Z_i, \alpha_i]_i^{\ell}$, where $\ell > q_\gamma$. These elements are represented by the following polynomial rational fractions: $\left[\frac{X_{\alpha_i}}{X_0+c_\gamma}\right]_{i=1}^{\ell}$. These ℓ elements are linearly-independent as they each include a different formal variable.

We argue that at least one element from $[X_{\alpha_i}/(X_0+c_\gamma)]_{i=1}^{\ell}$ is linearly independent from the initial elements and SDH elements given to the adversary. If so, then $\mathcal{A}_{\text{uber}}$ wins if it guesses correctly and the proof is complete. In other words, we want that:

$$\exists\, i \in [1,\ell] \;:\; \frac{X_{\alpha_i}}{X_0+c_\gamma} \notin \text{Span}\big(\big[1, [X_i]_{i=0}^{m}, [\tau'_i]_{i=1}^{q}\big]\big)$$

We use Claim 9 to rewrite the transcript τ' into a new transcript τ that contains elements $[[\tau_{i,j}]_{i=1}^{q_j}]_{j=1}^{n}$ of form defined in Equation 7. The transcript τ is built element by element by repeatedly applying Claim 9 to the next query in τ'. Ultimately we have that:

$$\text{Span}([\tau'_i]_{i=1}^{q}) = \text{Span}\big(\big[[\tau_{i,j}]_{i=1}^{q_j}\big]_{j=1}^{n}\big)$$
$$\text{Span}\big(\big[1, [X_i]_{i=0}^{m}, [\tau'_i]_{i=1}^{q}\big]\big) = \text{Span}\Big(\Big[1, [X_i]_{i=0}^{m}, \big[[\tau_{i,j}]_{i=1}^{q_j}\big]_{j=1}^{n}\Big]\Big)$$

So instead we will show

$$\exists\, i \in [1,\ell] \;:\; \frac{X_{\alpha_i}}{X_0+c_\gamma} \notin \text{Span}\Big(\Big[1, [X_i]_{i=0}^{m}, \big[[\tau_{i,j}]_{i=1}^{q_j}\big]_{j=1}^{n}\Big]\Big)$$

Next, consider a linear combination of the above that results in a winning element $X_\alpha/(X_0 + c_\gamma)$ with linear combination coefficients $r, s_i, t_{i,j}$:

$$r \cdot 1 + \sum_{i=0}^{m} s_i X_i + \sum_{j=1}^{n} \sum_{i=1}^{q_j} t_{i,j} \tau_{i,j} = \frac{X_\alpha}{X_0 + c_\gamma}$$

$$r \cdot 1 + \sum_{i=0}^{m} s_i X_i + \sum_{\substack{j=1 \\ j \neq \gamma}}^{n} \sum_{i=1}^{q_j} t_{i,j} \tau_{i,j} = \frac{X_\alpha}{X_0 + c_\gamma} - \sum_{i=1}^{q_\gamma} t_{i,\gamma} \tau_{i,\gamma} \tag{8}$$

Now if we multiply both side of the above Equation 8 by the LCM expression $\prod_{i=1}^{n} (X_0 + c_i)^{q_i}$, the quotients are removed and we have an equation of two polynomials. By the structure of $\tau_{i,j}$ from Equation 7, we have that none of the $\tau_{i,j}$ for $j \neq \gamma$ have a $(X_0 + c_\gamma)$ term in the denominator. Thus, the left hand side polynomial has a factor of $(X_0 + c_\gamma)^{q_\gamma}$.

On the right hand side, because every $\tau_{i,\gamma}$ term has $(X_0 + c_\gamma)$ in the denominator, $(X_0 + c_\gamma)^{q_\gamma}$ does not divide the right hand side polynomial. This implies that the two polynomials cannot be equal unless they are the zero polynomial, which is only possible if $r, [s_i]_{i=1}^{m}$ coefficients are equal to 0.

Thus, if $X_\alpha/(X_0 + c_\gamma) \in \text{Span}(\left[1, [X_i]_{i=0}^{m}, \left[[\tau_{i,j}]_{i=1}^{q_j}\right]_{j=1}^{n}\right])$, then it must be:

$$\sum_{i=1}^{q_\gamma} t_{i,\gamma} \tau_{i,\gamma} = \frac{X_\alpha}{X_0 + c_\gamma}$$

However, since there are only $q_\gamma < \ell$ $[\tau_{i,\gamma}]_{i=1}^{q_\gamma}$ terms, they can at most generate a q_γ-dimension space. Since the ℓ winning elements are linearly-independent and generate a ℓ-dimension space, it is not possible that they can all be generated from a linear combination of $[\tau_{i,\gamma}]_{i=1}^{q_\gamma}$. This concludes the proof.

B Security Proofs for 3H

For ease of reference, we restate the theorem here:

Theorem 4 *Let $\mathcal{A}_{\text{prf}}$ be a* P*-model POPRF adversary against* 3H *with query budget $(m, n, q_\mathsf{E}, \vec{q}, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4})$. Then we give a* H_4*-model adversary $\mathcal{A}_{\text{zk}}$ and adversary $\mathcal{A}_{\text{sdhi}}$ such that*

$$\mathsf{Adv}^{\text{po-prf}}_{\mathsf{3H},\mathsf{S}[\mathsf{S}_\Sigma],\mathsf{P},\mathcal{A}_{\text{prf}}}(\lambda) \leq \mathsf{Adv}^{\text{zk}}_{\Sigma_\mathcal{R},\mathcal{R},\mathsf{H}_4,\mathsf{S}_\Sigma,\mathcal{A}_{\text{zk}}}(\lambda) + \mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\text{sdhi}}}(\lambda) + \frac{n^2}{p},$$

where S *is the simulator defined in Figure 10 that makes use of NIZK simulator S_Σ. Adversary $\mathcal{A}_{\text{zk}}$ makes q_{H_4} queries to its random oracle and $\mathcal{A}_{\text{sdhi}}$ has query budget $(\vec{q}, q_{\mathsf{H}_2})$. Further, $T(\mathcal{A}_{\text{prf}}) \approx T(\mathcal{A}_{\text{zk}}) \approx T(\mathcal{A}_{\text{sdhi}})$.*

Proof: We bound the advantage of $\mathcal{A}_{\text{prf}}$ by bounding the advantage of each of a series of game hops. We define $\text{G}_0 = \text{POPRF}^{\mathcal{A}_{\text{po-prf}},1}_{\mathsf{3H},\mathsf{P},\mathsf{S}}(\lambda)$ and intermediate games $\text{G}_1, \text{G}_2, \text{G}_3, \text{G}_4, \text{G}_5$ to gradually transform the view of the adversary until $\text{G}_5 = \text{POPRF}^{\mathcal{A}_{\text{po-prf}},0}_{\mathsf{3H},\mathsf{P},\mathsf{S}}(\lambda)$. The most important games, $\text{G}_2 - \text{G}_4$ are shown in Figure 11). There we use tables R_1, R_2, R_3, R_4 for RO simulation, and we restrict collisions in R_3 (as described below) by sampling from $\mathbb{Z}_p$ without replacement, which we do by defining $\mathbb{Z}_p \setminus R_3$ to be the set of points in $\mathbb{Z}_p$ that do not appear in any entries of R_3.

The advantage bound follows from the following claims which we will justify:

(1) $|\Pr[\text{G}_0 = 1] - \Pr[\text{G}_1 = 1]| \leq \frac{n(n-1)}{2p}$

(2) $|\Pr[\text{G}_1 = 1] - \Pr[\text{G}_2 = 1]| = \mathsf{Adv}^{\text{zk}}_{\Sigma_\mathcal{R},\mathcal{R},\mathsf{RO}_4,\mathcal{A}_{\text{zk}}}(\lambda)$

(3) $|\Pr[\text{G}_2 = 1] - \Pr[\text{G}_3 = 1]| = 0$

(4) $|\Pr[\text{G}_3 = 1] - \Pr[\text{G}_4 = 1]| \leq \mathsf{Adv}^{(q_{\mathsf{H}_1},q_{\mathsf{H}_3})\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\text{sdhi}}}(\lambda)$

```
$\underline{\mathsf{S.Init}(\lambda, pp)}$
$R_1 \leftarrow [\cdot]\ ;\ \ R_2 \leftarrow [\cdot]\ ;\ \ R_3 \leftarrow [\cdot]$
$sk \leftarrow\!\!\$\ \mathbb{Z}_p$
$st_\Sigma \leftarrow\!\!\$\ \mathsf{S}_\Sigma.\mathsf{Init}(pp)$
$st_\mathsf{S} \leftarrow (pp, sk, R_1, R_2, R_3, st_\Sigma)$
Return $(st_\mathsf{S}, g^{sk})$

$\underline{\mathsf{S.BlindEv}^{\text{LimEv}}(t, req : (pp, sk, R_1, R_2, R_3, st_\Sigma))}$
$h \leftarrow\!\!\$\ \mathsf{S.H}_3^{\text{LimEv}}(t : st_\mathsf{S})$
$B \leftarrow req\ ;\ \ B' \leftarrow B^{1/(sk+h)}$
$\pi \leftarrow\!\!\$\ \mathsf{S}_\Sigma.\mathsf{Prove}((g, g^{sk+h}, B', B) : st_\Sigma)$
Return $(B', \pi)$

$\underline{\mathsf{S.H}_1^{\text{LimEv}}(x : (pp, sk, R_1, R_2, R_3, st_\Sigma))}$
If $x \notin R_1$ then $R_1[x] \leftarrow\!\!\$\ \mathbb{G}$
Return $R_1[x]$

$\underline{\mathsf{S.H}_2^{\text{LimEv}}(t, x, Y : (pp, sk, R_1, R_2, R_3, st_\Sigma))}$
If $t \notin R_3$ then $R_3[x] \leftarrow\!\!\$\ \mathbb{Z}_p$
If $x \notin R_1$ then $R_1[x] \leftarrow\!\!\$\ \mathbb{G}$
If $(t, x, Y) \notin R_2$ then
    If $Y = R_1[x]^{1/(sk+R_3[t])}$ then $R_2[t, x, Y] \leftarrow \text{LimEv}(t, x)$
    Else $R_2[t, x, Y] \leftarrow\!\!\$\ \{0,1\}^\lambda$
Return $R_2[t, x, Y]$

$\underline{\mathsf{S.H}_3^{\text{LimEv}}(x : (pp, sk, R_1, R_2, R_3, st_\Sigma))}$
If $x \notin R_3$ then $R_3[x] \leftarrow\!\!\$\ \mathbb{Z}_p$
Return $R_3[x]$

$\underline{\mathsf{S.H}_4^{\text{LimEv}}(x : (pp, sk, R_1, R_2, R_3, st_\Sigma))}$
Return $\mathsf{S}_\Sigma.\mathsf{H}(x : st_\Sigma)$
```

Figure 10: Simulator $\mathsf{S}[\mathsf{S}_\Sigma]$ for PRF security of $\mathsf{3H}$ where S_Σ is the zero knowledge simulator for $\Sigma_\mathcal{R}$.

(5) $|\Pr[\mathrm{G}_4 = 1] - \Pr[\mathrm{G}_5 = 1]| \le \frac{n(n-1)}{2p}$

Claim 1: We first transition to a game G_1 in which we disallow collisions in the output of RO_3, i.e., we replace sampling from $\mathbb{Z}_p$ for new range values with sampling without replacement from $\mathbb{Z}_p$. Recall that at most n unique t values are queried by $\mathcal{A}_{\text{prf}}$ in the course of the game, and so a standard birthday analysis establishes the claim's upper bound of $n(n-1)/2p$.

Claim 2: In G_2, the proof generated in BlindEv is simulated along with the corresponding random oracle RO_4. Since this is the only change between G_1 and G_2, we can equate the distinguishing advantage between the two games to that of the zero-knowledge security game of $\Sigma_\mathcal{R}$. We construct $\mathcal{A}_{\mathsf{zk}}$ that runs G_1 and generates proofs using its Prove oracle and responds to queries to RO_4 using its own Prim oracle.

Claim 3: In G_3, the Ev oracle generates outputs independently from the random oracles and stores its choices in table R. For consistency, it must be that for t, x and $Y = R_1[x]^{1/(sk+R_3[t])}$, the random output stored in R is the same as the one stored in R_2 for responding to RO_2 queries of (t, x, Y). G_3 checks if this is the case in RO_2, and if (t, x, Y) are of the above form, it repairs R and R_2 to be consistent. Thus, from the adversary's perspective, there is no change between G_2 and G_3.

Claim 4: In G_4, the repair between R and R_2 in RO_2 only occurs if there have been more calls to BlindEv on t than calls of valid (t, x, Y) tuples to RO_2. Otherwise, a bad flag is set and the oracle returns $\perp$ to the adversary. This matches the functionality of S in $\mathrm{POPRF}_{\mathsf{3H},\mathsf{P},\mathsf{S}}^{\mathcal{A}_{\text{po-prf}},0}(\lambda)$ since the simulator is restricted to not run LimEv on t more than calls made to BlindEv on t. By an identical-until-bad argument via the fundamental lemma of game playing [BR06],

$$|\Pr[\mathrm{G}_3 = 1] - \Pr[\mathrm{G}_4 = 1]| \le \Pr[\mathsf{bad} = 1]$$

where $\mathsf{bad} = 1$ is the event that bad is sent in game G_4. We bound the probability of this event by the advantage of an adversary $\mathcal{A}_{\text{sdhi}}$, i.e., if bad is set, $\mathcal{A}_{\text{sdhi}}$ wins the (m, n)-OM-Gap-SDHI game.

Adversary $\mathcal{A}_{\text{sdhi}} = (\mathcal{A}_1, \mathcal{A}_2)$ (shown in Figure 12) runs G_4 with the help of the (m, n)-OM-Gap-SDHI game. The $[Y_i]_i^m$ group elements are used for the values returned by RO_1. (This is often called "programming" RO_1.) The n values $[c]_i^n$ from $\mathbb{Z}_p$ output by $\mathcal{A}_1$ for use in the strong Diffie-Hellman queries are used for the return values from RO_3. By assumption on the query budget for $\mathcal{A}_{\text{prf}}$, the number of Y_i values and the number of c_i values are sufficient for simulating the queries made by $\mathcal{A}_{\text{prf}}$. And since both of these sets of values are chosen at random, they have the same distribution as the random oracle responses in G_4. Queries to BlindEv are answered by computing the strong Diffie-Hellman evaluation with the appropriate c_i value using SDH. Note that $\mathcal{A}_{\text{sdhi}}$ has query budget $\vec{q}$ because by assumption $\mathcal{A}_{\text{prf}}$ queries at most $\vec{q}_1$ times to

```
Games $G_2$ , $G_3$ , $\boxed{G_4}$
$R_1 \leftarrow [\cdot]$ ; $R_2 \leftarrow [\cdot]$ ; $R_3 \leftarrow [\cdot]$
$R \leftarrow [\cdot]$
$\boxed{i_* \leftarrow 0 \;;\; j_* \leftarrow 0 \;;\; \mathsf{bad} \leftarrow 0}$
$pp \leftarrow\$ \mathsf{3H.Setup}(\lambda)$
$sk \leftarrow\$ \mathbb{Z}_p$ ; $pk \leftarrow g^{sk}$
$st_\Sigma \leftarrow\$ \mathsf{S}_\Sigma.\mathsf{Init}(pp)$
$b' \leftarrow\$ \mathcal{A}_{\mathrm{prf}}^{\mathsf{P},\mathrm{Ev},\mathrm{BlindEv}}(pp, pk)$
Return $b'$
```

```
Oracle $\mathrm{Ev}(t, x)$
If $t \notin R_3$ then $R_3[t] \leftarrow\$ \mathbb{Z}_p \setminus R_3$
If $x \notin R_1$ then $R_1[x] \leftarrow\$ \mathbb{G}$
$Y \leftarrow R_1[x]^{1/(sk+R_3[t])}$
If $(t, x, Y) \notin R_2$ then $R_2[t, x, Y] \leftarrow\$ \{0,1\}^\lambda$
$Z \leftarrow R_2[t, x, Y]$
If $(t, x) \notin R$ then
    $R[t, x] \leftarrow\$ \{0,1\}^\lambda$
$Z \leftarrow R[t, x]$
Return $Z$
```

```
Oracle $\mathrm{BlindEv}(t, req)$
If $t \notin R_3$ then $R_3[t] \leftarrow\$ \mathbb{Z}_p \setminus R_3$
$B \leftarrow req$ ; $B' \leftarrow B^{1/(sk+R_3[t])}$
$\boxed{j_t \leftarrow j_t + 1}$
$\pi \leftarrow\$ \mathsf{S}_\Sigma.\mathsf{Prove}((g, g^{sk+h}, B', B) : st_\Sigma)$
Return $(B', \pi)$
```

```
Oracle $\mathsf{H}_1(x)$
If $x \notin R_1$ then $R_1[x] \leftarrow\$ \mathbb{G}$
Return $R_1[x]$
```

```
Oracle $\mathsf{H}_2(t, x, Y)$
If $t \notin R_3$ then $R_3[t] \leftarrow\$ \mathbb{Z}_p \setminus R_3$
If $x \notin R_1$ then $R_1[x] \leftarrow\$ \mathbb{G}$
If $(t, x, Y) \notin R_2$ then
    $R_2[t, x, Y] \leftarrow\$ \{0,1\}^\lambda$
    If $Y = R_1[x]^{1/(sk+R_3[t])}$ then
        $\boxed{i_t \leftarrow i_t + 1}$
        $\boxed{\text{If } i_t > j_t \text{ then } \mathsf{bad} \leftarrow 1 \;;\; \text{Return } \perp}$
        If $(t, x) \notin R$ then $R[t, x] \leftarrow\$ \{0,1\}^\lambda$
        $R_2[t, x, Y] \leftarrow R[t, x]$
    Else $R_2[t, x, Y] \leftarrow\$ \{0,1\}^\lambda$
Return $R_2[t, x, Y]$
```

```
Oracle $\mathsf{H}_3(x)$
If $x \notin R_3$ then $R_3[x] \leftarrow\$ \mathbb{Z}_p \setminus R_3$
Return $R_3[x]$
```

```
Oracle $\mathsf{H}_4(x)$
Return $\mathsf{S}_\Sigma.\mathsf{H}(x : st_\Sigma)$
```

Figure 11: The key game transitions used in pseudorandomness security for 3H. Greyed highlighted statements are only included in G_2, blue highlighted statements in G_3 and G_4, and boxed statements only in G_4.

BlindEv on the first value t_1 queried in the course of the game, at most $\vec{q}_2$ times for the second value t_2 queried in the course of the game, and so on.

In RO_2, the form of (t, x, Y) is checked using SDDH to determine if a repair between R and R_2 needs to be performed. However, before the repair is done, if there have been more valid (t, x, Y) tuples queried to Prim$_2$ than queries to BlindEv, $\mathcal{A}_{\mathrm{sdhi}}$ then halts execution of $\mathcal{A}_{\mathrm{prf}}$ and concludes by outputting $(\gamma, \hat{Z}[\gamma])$. In this case, the adversary has found "one more" valid strong Diffie-Hellman tuple (one corresponding to each valid (t, x, Y) tuple since $Y = Y_i^{1/(sk+c_j)}$ for some Y_i, c_j) than calls made to SDH. The adversary therefore wins the (m, n)-OM-Gap-SDHI game.

Claim 5: The final game transition restores the possibility of collisions to RO_3, and a birthday analysis gives the upper bound on this transition.

■

C Security with Restricted Tag Space

Corollary 10 *For any adversary $\mathcal{A}_{\mathrm{prf}}$ against the partially-oblivious pseudorandomness of* 3H *with restricted tag space of size $|\mathsf{T}|$, we give adversaries $\mathcal{A}_{\mathrm{zk}}$ and $\mathcal{A}_{\mathrm{sdhi}}$ such that*

$$\mathsf{Adv}^{\text{po-prf}}_{\mathsf{3H},\mathsf{P},\mathsf{S}[\mathsf{S}_\Sigma],\mathcal{A}_{\mathrm{prf}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{zk}}_{\Sigma_{\mathcal{R}},\mathcal{R},\mathsf{RO}_4,\mathsf{S}_\Sigma,\mathcal{A}_{\mathrm{zk}}}(\lambda) + \mathsf{Adv}^{(q_{\mathsf{H}_1},|\mathsf{T}|)\text{-om-gap-sdhi}}_{\mathsf{GGen},\mathcal{A}_{\mathrm{sdhi}}}(\lambda)\,,$$

where S *is the simulator defined in Figure 10, the ideal primitive $\mathsf{P} = \mathsf{RO}_1 \times \mathsf{RO}_2 \times \mathsf{RO}_3 \times \mathsf{RO}_4$ for random oracles over $\mathsf{RO}_1 : * \to \mathbb{G}$, $\mathsf{RO}_2 : * \times * \times \mathbb{G} \to \{0,1\}^\lambda$, $\mathsf{RO}_3 : * \to \mathbb{Z}_p$, $\mathsf{RO}_4 : \mathbb{G}^6 \to \mathbb{Z}_p$ for $(p, \mathbb{G})$ determined by $\mathsf{GGen}(\lambda)$. The running time $T(\mathcal{A}_{\mathrm{prf}}) \approx T(\mathcal{A}_{\mathrm{zk}}) \approx T(\mathcal{A}_{\mathrm{sdhi}})$ and $\mathcal{A}_{\mathrm{prf}}$ makes at most q_{H_1} queries to RO_1.*

Proof: The proof follows the same as above except only queries to Prim$_3$ that fall within the tag space T need to be programmed with a strong Diffie-Hellman constant c; otherwise, a random value can be sampled.

```
$\mathcal{A}_1(p, \mathbb{G})$
$C \leftarrow \emptyset$
For $i = 1$ to $n$:
    $c_i \leftarrow\!\$\; \mathbb{Z}_p \setminus C$ ; $C \leftarrow C \cup \{c_i\}$
$st_{\mathcal{A}} \leftarrow (p, \mathbb{G}, [c_i]_i^n)$
Return $(st_{\mathcal{A}}, [c_i]_i^n)$

$\mathcal{A}_2^{\mathrm{SDH},\mathrm{SDDH}}(g, pk, [Y_i]_i^m : (p, \mathbb{G}, [c_i]_i^n)$
$i \leftarrow 1$ ; $j_* \leftarrow 0$ ; $\ell \leftarrow 1$
$K \leftarrow [\cdot]$ ; $\hat{Z} \leftarrow [\cdot]$ ; $\gamma \leftarrow \bot$
$R_1 \leftarrow [\cdot]$ ; $R_2 \leftarrow [\cdot]$ ; $R_3 \leftarrow [\cdot]$ ; $R \leftarrow [\cdot]$
$pp \leftarrow (p, g, \mathbb{G})$
$st_\Sigma \leftarrow\!\$\; \mathsf{S}_\Sigma.\mathsf{Init}(pp)$
$b' \leftarrow\!\$\; \mathcal{A}_{\mathrm{prf}}^{\mathsf{P},\textsc{Ev},\textsc{BlindEv}}(pp, pk)$
Return $(\gamma, \hat{Z}[\gamma])$

Oracle $\textsc{Ev}(t, x)$
If $(t, x) \notin R$ then $R[t, x] \leftarrow\!\$\; \{0,1\}^\lambda$
$Z \leftarrow R[t, x]$
Return $Z$

Oracle $\textsc{BlindEv}(t, req)$
If $t \notin R_3$ then $R_3[t] \leftarrow \ell$ ; $\ell \leftarrow \ell + 1$
$j_t \leftarrow j_t + 1$
$B \leftarrow req$
$B' \leftarrow \mathrm{SDH}(B, R_3[t])$
$\pi \leftarrow\!\$\; \mathsf{S}_\Sigma.\mathsf{Prove}((g, X \cdot g^{c_{R_3[t]}}, B', B) : st_\Sigma)$
Return $(B', \pi)$
```

```
Oracle $\mathsf{H}_1(x)$
If $x \notin R_1$ then
    $R_1[x] \leftarrow Y_i$
    $K[x] \leftarrow i$ ; $i \leftarrow i + 1$
Return $R_1[x]$

Oracle $\mathsf{H}_2(t, x, Y)$
If $t \notin R_3$ then
    $R_3[t] \leftarrow \ell$ ; $\ell \leftarrow \ell + 1$
If $x \notin R_1$ then
    $R_1[x] \leftarrow Y_i$
    $K[x] \leftarrow i$ ; $i \leftarrow i + 1$
If $(t, x, Y) \notin R_2$ then
    If $\mathrm{SDDH}(Y, R_1[x], R_3[t])$ then
        $\hat{Z}[R_3[t]] \leftarrow \hat{Z}[R_3[t]] \parallel (Y, K[x])$
        If $|\hat{Z}[R_3[t]]| > j_t$ then
            $\gamma \leftarrow R_3[t]$ ; abort $\mathcal{A}_{\mathrm{prf}}$
        If $x \notin R$ then $R[x] \leftarrow\!\$\; \{0,1\}^\lambda$
        $R_2[t, x, Y] \leftarrow R[x]$
    Else $R_2[t, x, Y] \leftarrow\!\$\; \{0,1\}^\lambda$
Return $R_2[t, x, Y]$

Oracle $\mathsf{H}_3(x)$
If $x \notin R_3$ then
    $R_3[x] \leftarrow \ell$ ; $\ell \leftarrow \ell + 1$
Return $c_{R_3[x]}$

Oracle $\mathsf{H}_4(x)$
Return $\mathsf{S}_\Sigma.\mathsf{H}(x : st_\Sigma)$
```

Figure 12: Adversary $\mathcal{A}_{\mathrm{sdhi}} = (\mathcal{A}_1, \mathcal{A}_2)$ used in POPRF security proof of 3H.

D Security of 2HashDH OPRF

In this section, we demonstrate the extensibility of our security definitions by proving the security of the 2HashDH OPRF [JKK14]. The only previous proof of security for 2HashDH is with respect to the UC definition provided by Jarecki et al. [JKK14]. The construction 2HashDH is given in Figure 13. The security proofs for 2HashDH follow closely to the proofs of 3H. As such, we provide only proof sketches for 2HashDH referring heavily to the detailed proofs in Appendix B and Section 5.

Pseudorandomness. First, we prove pseudorandomness of the 2HashDH OPRF with respect to a variant of POPRF given in Figure 3 for the OPRF setting, which we name OPRF. The security game OPRF is the same as POPRF except the public tag inputs to the oracles and algorithms are removed to fit the OPRF setting and only a single query counter is maintained for tracking blind evaluation and limited evaluation oracle queries, rather than a separate query counter for each public tag. We let the advantage of a OPRF adversary $\mathcal{A}$ be defined by

$$\mathsf{Adv}^{\mathrm{oprf}}_{\mathsf{Fn},\mathsf{S},\mathsf{P},\mathcal{A},}(\lambda) = \left|\Pr\left[\,\mathrm{OPRF}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{S},\mathsf{P}}(\lambda) \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{OPRF}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{S},\mathsf{P}}(\lambda) \Rightarrow 1\,\right]\right| .$$

We will reduce the pseudorandomness security of 2HashDH to the security against the so-called one-more gap computational Diffie-Hellman assumption [JKK14, Bol03] (with pseudocode given in Figure 14). Here, an adversary receives $m+1$ group elements, $g^x = X$, $[g^{y_i} = Y_i]_{i=1}^m$, and is tasked with computing $q+1$ computational Diffie-Hellman values of the form $Z_i = g^{x \cdot y_i}$, where q is the number of queries the adversary makes to a helper oracle CDH$(\cdot)$ that returns the CDH value of the input element with X. The adversary also is given access to a gap oracle to answer the decisional Diffie-Hellman question on arbitrary inputs. The non-gap version of this assumption was recently shown secure in the generic group model [BFP21], however the same approach and bounds hold even when the gap oracle is included [BFL20]. We define the m-OM-Gap-CDH-advantage of an adversary $\mathcal{A}$ by

$$\mathsf{Adv}^{m\text{-om-gap-cdh}}_{\mathsf{GGen},\mathcal{A}}(\lambda) = \Pr\left[\,m\text{-OM-}\textsc{Gap}\text{-CDH}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda) \Rightarrow \mathsf{true}\,\right] .$$

We state below a theorem using the ideal primitive model in which $\mathsf{P} = \mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3$ for random oracles

```
$\underline{\mathsf{2HashDH.Setup}(\lambda)}$
    $(p, g, \mathbb{G}) \leftarrow\!\$\ \mathsf{GGen}(\lambda)$
    $pp \leftarrow (p, g, \mathbb{G})$
    Return $pp$

$\underline{\mathsf{2HashDH.KeyGen}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3}(pp)}$
    $(p, g, \mathbb{G}) \leftarrow pp$ ;  $z \leftarrow\!\$\ \mathbb{Z}_p$
    $sk \leftarrow z$ ;  $pk \leftarrow g^z$
    Return $(sk, pk)$

$\underline{\mathsf{2HashDH.Ev}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3}(pp, sk, x)}$
    $Y \leftarrow \mathsf{H}_1(x)^{sk}$
    $Z \leftarrow \mathsf{H}_2(x, Y)$
    Return $Z$
```

```
$\underline{\mathsf{2HashDH.Req}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3}(pk, x)}$
    $r \leftarrow\!\$\ Z_p$ ;  $B \leftarrow \mathsf{H}_1(x)^r$
    Return $((pk, r, x), B)$
                                        $\xrightarrow{B}$
                                            $\underline{\mathsf{2HashDH.BlindEv}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3}(sk, B)}$
                                                $B' \leftarrow B^{sk}$
                                                $\pi \leftarrow\!\$\ \Sigma_{\mathcal{R}}.\mathsf{Prove}^{\mathsf{H}_3}(sk, (g, g^{sk}, B, B'))$
                                                Return $(B', \pi)$
                                        $\xleftarrow{B', \pi}$
$\underline{\mathsf{2HashDH.Finalize}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3}(B', \pi; (pk, r, x))}$
    $Y \leftarrow (B')^{1/r}$
    Require $\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}_3}((g, pk, B, B'), \pi)$
    $Z \leftarrow \mathsf{H}_2(x, Y)$
    Return $Z$
```

Figure 13: The 2HashDH OPRF construction of [JKK14]. Algorithms have implicit input the parameters $pp = (p, g, \mathbb{G})$ that describe the group used. The NIZK uses relation $\mathcal{R} = \{(g, U, V, W), (\alpha) : U = g^{\alpha} \wedge W = V^{\alpha}\}$.

```
$\underline{\text{Game } m\text{-OM-GAP-CDH}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda)}$
    $(p, g, \mathbb{G}) \leftarrow\!\$\ \mathsf{GGen}(\lambda)$
    $q \leftarrow 0$
    $x \leftarrow\!\$\ \mathbb{Z}_p$ ;  $[y_i]_i^m \leftarrow\!\$\ [\mathbb{Z}_p]_i^m$
    $[Z_i, \alpha_i]_i^{\ell} \leftarrow\!\$\ \mathcal{A}^{\mathrm{CDH,DDH}}(p, \mathbb{G}, g, g^x, [g^{y_i}]_i^m)$
    Require $q < \ell \ \wedge\ \forall^{\ell}_{i \neq j}\ \alpha_i \neq \alpha_j$
    Return $[Z_i]_i^{\ell} = [g^{x \cdot y_i}]_i^{\ell}$

$\underline{\text{Oracle CDH}(Y)}$
    $q \leftarrow q + 1$
    $Z \leftarrow Y^x$
    Return $Z$

$\underline{\text{Oracle DDH}(h, A, B, C)}$
    $a \leftarrow \log_h(A)$
    $b \leftarrow \log_h(B)$
    $c \leftarrow \log_h(C)$
    Return $c \equiv_p ab$
```

Figure 14: The one-more gap computational Diffie-Hellman security game.

over $\mathsf{H}_1 : * \to \mathbb{G}$, $\mathsf{H}_2 : * \times \mathbb{G} \to \{0,1\}^{\lambda}$, $\mathsf{H}_3 : \mathbb{G}^6 \to \mathbb{Z}_p$ for $(p, \mathbb{G})$ determined by $\mathsf{GGen}(\lambda)$.

Theorem 11 *Let $\mathcal{A}_{\mathrm{prf}}$ be a P-model OPRF adversary against* 2HashDH. *Then we give a H_3-model adversary $\mathcal{A}_{\mathrm{zk}}$ and adversary $\mathcal{A}_{\mathrm{cdh}}$ such that*

$$\mathsf{Adv}^{\mathrm{oprf}}_{\mathsf{2HashDH},\mathsf{S}[\mathsf{S}_\Sigma],\mathsf{P},\mathcal{A}_{\mathrm{prf}}}(\lambda) \leq \mathsf{Adv}^{\mathrm{zk}}_{\Sigma_{\mathcal{R}},\mathcal{R},\mathsf{H}_3,\mathsf{S}_\Sigma,\mathcal{A}_{\mathrm{zk}}}(\lambda) + \mathsf{Adv}^{(q_{\mathsf{H}_1}+q_{\mathsf{H}_2})\text{-}om\text{-}gap\text{-}cdh}_{\mathsf{GGen},\mathcal{A}_{\mathrm{cdh}}}(\lambda)\,,$$

where S is the simulator defined in Figure 15 that makes use of NIZK simulator S_Σ, and q_{H_1}, q_{H_2}, q_{H_3} are the number of queries $\mathcal{A}_{\mathrm{prf}}$ makes to its respective ideal primitives. Adversary $\mathcal{A}_{\mathrm{zk}}$ makes q_{H_3} queries to its random oracle and $\mathcal{A}_{\mathrm{cdh}}$ makes q_{B} queries to its CDH oracle. Further, $T(\mathcal{A}_{\mathrm{prf}}) \approx T(\mathcal{A}_{\mathrm{zk}}) \approx T(\mathcal{A}_{\mathrm{cdh}})$.

Proof: The follows closely to the proof of pseudorandomness for 3H in Appendix B. We skip game hops $\mathrm{G}_0 \to \mathrm{G}_1$ and $\mathrm{G}_4 \to \mathrm{G}_5$ that deal with disallowing and restoring collisions in the random oracle used for public tags. The 2HashDH construction does not use that random oracle, and thus the birthday bound terms from the 3H analysis do not appear.

The first game hop is instead $\mathrm{G}_1 \to \mathrm{G}_2$ (of Appendix B) which transitions to a game that replaces the NIZK generated in BLINDEV with one generated by the NIZK simulator S_Σ.

The second game hop follows $\mathrm{G}_2 \to \mathrm{G}_3$ (of Appendix B) in which we modify the handling of H_2 queries to check if the table R from EV has been set on a relevant value and, if so, patch up H_2's response to maintain consistency. This does not change the distribution of responses to the adversary so there is no distinguishing advantage.

Lastly, the final game hop follows $\mathrm{G}_3 \to \mathrm{G}_4$ (of Appendix B) by only repairing H_2's response if there has been more queries to BLINDEV than repairs required. The only difference between this game and the ideal world is when more repairs occur than queries to BLINDEV. We show an adversary against the OM-Gap-CDH assumption with advantage greater than or equal to the distinguishing advantage between these two games. The views of the games are simulated using the initial values to program H_1, the CDH oracle in BLINDEV, and the DDH oracle to check for repairs in H_2.

```
$\underline{\mathsf{S.Init}^{\mathrm{Ev}}(\lambda, pp)}$
$R_1 \leftarrow [\cdot]$ ; $R_2 \leftarrow [\cdot]$
$st_\Sigma \leftarrow\$ \mathsf{S}_\Sigma.\mathsf{Init}(\lambda, pp)$
$st_\mathsf{S} \leftarrow (pp, \bot, R_1, R_2, st_\Sigma)$
Return $st_\mathsf{S}$

$\underline{\mathsf{S.KeyGen}^{\mathrm{Ev}}(: (pp, sk, R_1, R_2, st_\Sigma))}$
$(g, p, \mathbb{G}) \leftarrow pp$ ; $sk \leftarrow\$ \mathbb{Z}_p$
Return $g^{sk}$

$\underline{\mathsf{S.BlindEv}^{\mathrm{Ev}}(req : (pp, sk, R_1, R_2, st_\Sigma))}$
$B \leftarrow req$ ; $B' \leftarrow B^{sk}$
$\pi \leftarrow\$ \mathsf{S}_\Sigma.\mathsf{Prove}((g, g^{sk}, B, B') : st_\Sigma)$
Return $(B', \pi)$

$\underline{\mathsf{S.Eval}^{\mathrm{Ev}}(x : (pp, sk, R_1, R_2, st_\Sigma))}$
$(i, x) \leftarrow x$
$y \leftarrow\$ \mathsf{S.Eval}_i(x : st_\mathsf{S})$
Return $y$

$\underline{\mathsf{S.Eval}_1^{\mathrm{Ev}}(x : (pp, sk, R_1, R_2, st_\Sigma))}$
$(g, p, \mathbb{G}) \leftarrow pp$
If $x \notin R_1$ then $R_1[x] \leftarrow\$ \mathbb{G}$
Return $R_1[x]$

$\underline{\mathsf{S.Eval}_2^{\mathrm{Ev}}(x, Y : (pp, sk, R_1, R_2, st_\Sigma))}$
If $x \notin R_1$ then $R_1[x] \leftarrow\$ \mathbb{G}$
If $(x, Y) \notin R_2$ then
    If $Y = R_1[x]^{sk}$ then $R_2[x, Y] \leftarrow \mathrm{Ev}(x)$
    Else $R_2[x, Y] \leftarrow\$ \{0,1\}^\lambda$
Return $R_2[x, Y]$

$\underline{\mathsf{S.Eval}_3^{\mathrm{Ev}}(x : (pp, sk, R_1, R_2, st_\Sigma))}$
Return $\mathsf{S}_\Sigma.\mathsf{Eval}(x : st_\Sigma)$
```

Figure 15: Simulator $\mathsf{S}[\mathsf{S}_\Sigma]$ for OPRF security of $\mathsf{2HashDH}$ where S_Σ is the zero knowledge simulator for $\Sigma_\mathcal{R}$.

Request Privacy. Next, we provide theorems for the two notions of request privacy (with and without proofs of correct blind evaluation) for $\mathsf{2HashDH}$. We again define analogues of POPRIV1 and POPRIV2 (see Figure 4) for the OPRF setting, which we refer to as OPRIV1 and OPRIV2. The games are identical except the public metadata tag is removed as input to oracles and algorithms. The advantage of a OPRIV1 adversary $\mathcal{A}$ in the P-model is defined by

$$\mathsf{Adv}^{\mathrm{opriv1}}_{\mathsf{Fn},\mathsf{P},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{POPRIV1}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right] - \Pr\left[\mathrm{POPRIV1}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right]\right| ,$$

and the advantage of a OPRIV2 adversary $\mathcal{A}$ defined by

$$\mathsf{Adv}^{\mathrm{opriv2}}_{\mathsf{Fn},\mathsf{P},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{POPRIV2}^{\mathcal{A},1}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right] - \Pr\left[\mathrm{POPRIV2}^{\mathcal{A},0}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\right]\right| .$$

We provide the following theorems without proof as they follow directly from the proofs given in Section 5.

Theorem 12 *For any* OPRIV1 *adversary* $\mathcal{A}_{\mathrm{opriv1}}$ *against* $\mathsf{2HashDH}$ *(without client verification) we have that* $\mathsf{Adv}^{\mathrm{opriv1}}_{\mathsf{2HashDH},\mathcal{A}_{\mathrm{po\text{-}priv1}}}(\lambda) = 0$.

Theorem 13 *Let* $\mathcal{A}_{\mathrm{opriv2}}$ *be a* OPRIV2 *adversary in the* P*-model against* $\mathsf{2HashDH}$ *that makes at most* q *queries to* FIN. *We give an adversary* $\mathcal{B}_{\mathrm{sound}}$ *such that*

$$\mathsf{Adv}^{\mathrm{opriv2}}_{\mathsf{2HashDH},\mathsf{P},\mathcal{A}_{\mathrm{opriv2}}}(\lambda) \leq 4q \cdot \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{H}_3,\mathcal{B}_{\mathrm{sound}}}(\lambda)) .$$

Further, $T(\mathcal{B}_{\mathrm{sound}}) \approx T(\mathcal{A}_{\mathrm{opriv2}})$.

E (Partially) Blind Signatures Preliminaries

In this section, we define the syntax, semantics, and security properties of partially blind digital signatures. As in our definitions for POPRFs, the formalism we present here supports the public metadata input of "partially" blind digital signatures, but can be easily adapted for the simpler blind digital signature setting without public metadata. Also as in our treatment of POPRFs, we simplify our handling of the interactive blind signing protocol by restricting our attention to a single round of interaction. This approach differs from related definitions [HKL19] for "three-move" blind signatures capturing Schnorr-type schemes; the constructions we consider in this work are only "two-move" and hence we benefit from shedding the extra definitional complexity.

E.1 Syntax and Semantics

A partially blind digital signature scheme, DS, is a tuple of algorithms

$$(\mathsf{DS.Setup}, \mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Ver}, \mathsf{DS.Req}, \mathsf{DS.BlindSign}, \mathsf{DS.Finalize})$$

The setup and key generation algorithm generate public parameters pp and a public key, secret key pair (pk, sk), respectively. Blind signing is carried out as an interactive protocol run between client and server:

(1) First, a client runs the algorithm $\mathsf{DS.Req}^{\mathsf{P}}_{pp}(pk, t, m)$, which takes input a public key pk, tag (or public input) t, and message m, and outputs a local state st and a request message req. The message req is sent to a server.

(2) A server runs algorithm $\mathsf{DS.BlindSign}^{\mathsf{P}}_{pp}(sk, t, req)$, using as input a secret key, a tag t, and the request message. It produces a response message rep that should be sent back to the client.

(3) Finally, the client runs the algorithm $\mathsf{DS.Finalize}(rep : st)$ and outputs an unblinded signature σ on the message-tag pair (t, m) or $\perp$ if the response message is rejected, for example, due to the verification check failing.

The unblinded signing algorithm $\mathsf{DS.Sign}$ is randomized, and takes as input a secret key sk, an input pair (t, m), and outputs a signature σ. The verification algorithm $\mathsf{DS.Ver}$ takes as input a public key pk, an input pair (t, m), and a signature σ, and outputs 1 if the signature is valid and 0 otherwise. We also define sets $\mathsf{DS.SK}$, $\mathsf{DS.PK}$, $\mathsf{DS.T}$, $\mathsf{DS.M}$, and $\mathsf{DS.\Sigma}$ representing the secret key, public key, tag, message, and signature space, respectively. DS is called *unique* if there exists only a single valid signature σ for input (t, x).

For correctness, we require that both the unblinded signing algorithm and interactive blind signing protocol produce valid signatures. To formalize the latter: we require that for pp output from Setup, any pk, sk output by KeyGen, and any t, m, it holds that $\Pr[\mathsf{Ver}(pk, t, m, \sigma) = 1] = 1$ where the probability is taken over choice of σ via the following process:

$$(st, req) \leftarrow\!\!\$\ \mathsf{Req}^{\mathsf{P}}(pk, t, m)\ ;\ rep \leftarrow\!\!\$\ \mathsf{BlindSign}^{\mathsf{P}}(sk, t, req)\ ;\ \sigma \leftarrow\!\!\$\ \mathsf{Finalize}^{\mathsf{P}}(rep : st)\ .$$

E.2 Security

We introduce two new definitions of security for partially blind digital signatures, tailored for the case of one round blind signing. Our definitions match closely to those introduced for the POPRF setting. We use code-based games mostly following the framework of Bellare and Rogaway [BR06].

One-more unforgeability. The first definition captures unforgeability, ensuring that it is not possible for malicious clients to forge signatures even when given access to a blinded signing oracle. We give a pseudocode game in Figure 16. The adversary is tasked with producing $q + 1$ distinct message-signature tuples $[(m_i, \sigma_i)]_{i=1}^{q+1}$ that all verify under a given public key pk and adversary-chosen public tag t'. The adversary is given access to a blinded signing oracles for pk but wins only if they query the oracle q times or less on the chosen tag t'. This intuitively enforces that each query to the blind signing oracle only results in one learned signature, and queries for a different public tag do not help in forging a signature for the target tag. We let the advantage of a OM-UNF adversary $\mathcal{A}$ be defined by

$$\mathsf{Adv}^{\text{om-unf}}_{\mathsf{DS},\mathsf{P},\mathcal{A}}(\lambda) = \Pr\left[\,\text{OM-UNF}^{\mathcal{A}}_{\mathsf{DS},\mathsf{P}}(\lambda) \Rightarrow \mathsf{true}\,\right]\ .$$

Blindness. The second definition captures message privacy of a client in the face of a malicious server. We give a pseudocode game in Figure 17. The game is the exact analogue of the request privacy against malicious adversary game POPRIV2 (see Section 3) for POPRFs adapted for blind signature syntax. The adversary is given access to a request oracle and finalize oracle representing client behavior. The request oracle takes an adversary-chosen public key, public tag, and pair of messages, and outputs a randomly ordered pair of client request messages (for the input messages) based on a challenge bit. The finalize oracle takes an adversary-chosen pair of response messages, and outputs the produced signatures (in the original order). To prevent trivial attacks of corrupting one response, the oracle requires that both signatures finalize without error, i.e. do not equal $\perp$.

```
Game OM-UNF$^{\mathcal{A}}_{\mathsf{DS},\mathsf{P}}(\lambda)$
$st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(\lambda)$
$pp \leftarrow\$ \mathsf{DS.Setup}(\lambda)$
$(sk, pk) \leftarrow\$ \mathsf{DS.KeyGen}^{\mathsf{P}}_{pp}()$
$(t', [m_i, \sigma_i]^{\ell}_{i}) \leftarrow\$ \mathcal{A}^{\text{BlindSign},\mathsf{P}}(pp, pk)$
Require $q_{t'} < \ell \wedge \forall^{\ell}_{i \neq j}\ (m_i, \sigma_i) \neq (m_j, \sigma_j)$
Return $\bigwedge^{\ell}_{i} \mathsf{DS.Ver}(pk, t', m_i, \sigma_i)$

Oracle BlindSign$(t, req)$
$q_t \leftarrow q_t + 1$
$rep \leftarrow \mathsf{DS.BlindSign}^{\mathsf{P}}(sk, t, req)$
Return $rep$
```

Figure 16: One-more unforgeability security game for partially blind signatures. The query counters q_t are initialized to 0 for all $t \in \mathsf{DS.T}$.

```
Game Blind$^{\mathcal{A},b}_{\mathsf{Fn},\mathsf{P}}(\lambda)$
$i \leftarrow 0$
$st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(\lambda)$
$pp \leftarrow\$ \mathsf{DS.Setup}(\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\text{Req},\text{Fin},\mathsf{P}}(pp)$
Return $b'$

Oracle Req$(pk, t, m_0, m_1)$
$i \leftarrow i + 1$
$(st_{i,0}, req_0) \leftarrow\$ \mathsf{DS.Req}^{\mathsf{P}}(pk, t, m_0)$
$(st_{i,1}, req_1) \leftarrow\$ \mathsf{DS.Req}^{\mathsf{P}}(pk, t, m_1)$
Return $(req_b, req_{1-b})$

Oracle Fin$(j, rep, rep')$
If $j > i$ then return $\perp$
$\sigma_b \leftarrow\$ \mathsf{DS.Finalize}^{\mathsf{P}}(st_{j,b}, rep)$
$\sigma_{1-b} \leftarrow\$ \mathsf{Fn.Finalize}^{\mathsf{P}}(st_{j,1-b}, rep')$
If $\sigma_0 = \perp$ or $\sigma_1 = \perp$ then
    Return $\perp$
Return $(\sigma_0, \sigma_1)$
```

Figure 17: Security definition for blindness of message in partially blind signatures.

The advantage of a Blind adversary $\mathcal{A}$ is defined by

$$\mathsf{Adv}^{\text{blind}}_{\mathsf{DS},\mathsf{P},\mathcal{A}}(\lambda) = \left|\Pr\left[\text{Blind}^{\mathcal{A},1}_{\mathsf{DS},\mathsf{P}}(\lambda) \Rightarrow 1\right] - \Pr\left[\text{Blind}^{\mathcal{A},0}_{\mathsf{DS},\mathsf{P}}(\lambda) \Rightarrow 1\right]\right| .$$

F (P)OPRFs from Unique (Partially) Blind Signatures

In this section, we provide proof for the folklore transform of a unique partially blind signature (resp. blind signature) to a POPRF (resp. OPRF) in the random oracle model. The transform, observed by Jarecki et al. [JKK14], simply consists of hashing the signature to create the PRF output. We call this transform $\mathsf{HSig}[\mathsf{DS}]$ where if DS uses ideal primitive P, HSig uses ideal primitive $\mathsf{P} \times \mathsf{H}$ where H is a random oracle over $\mathsf{H} : \mathsf{DS.T} \times \mathsf{DS.M} \times \mathsf{DS.\Sigma} \rightarrow \{0,1\}^{\lambda}$. The pseudocode is given in Figure 18.

Pseudorandomness. We prove the following theorem:

Theorem 14 *Let $\mathcal{A}_{\text{prf}}$ be a $(\mathsf{P} \times \mathsf{H})$-model POPRF adversary against $\mathsf{HSig}[\mathsf{DS}]$. Then we give a P-model adversary $\mathcal{A}_{\text{om-unf}}$ such that*

$$\mathsf{Adv}^{\text{po-prf}}_{\mathsf{HSig}[\mathsf{DS}],\mathsf{S}[\mathsf{DS}],\mathsf{P}\times\mathsf{H},\mathcal{A}_{\text{prf}}}(\lambda) \leq \mathsf{Adv}^{\text{om-unf}}_{\mathsf{DS},\mathsf{P},\mathcal{A}_{\text{om-unf}}}(\lambda)\,,$$

where S is the simulator defined in Figure 19. If $q_{\mathsf{B},\mathsf{t}}$ is the number of queries $\mathcal{A}_{\text{prf}}$ makes to its blind evaluation oracle for tag t, adversary $\mathcal{A}_{\text{om-unf}}$ makes $q_{\mathsf{B},\mathsf{t}}$ queries to its blind signing oracle for tag t. Further, $T(\mathcal{A}_{\text{prf}}) \approx T(\mathcal{A}_{\text{om-unf}})$.

Proof: We bound the advantage of $\mathcal{A}_{\text{prf}}$ by bounding the advantage of a series of game hops. We define $\mathrm{G}_0 = \text{POPRF}^{\mathcal{A}_{\text{prf}},1}_{\mathsf{HSig},\mathsf{P}\times\mathsf{H},\mathsf{S}}(\lambda)$. In the first game hop to G_1, we replace the proper evaluation and call to H in Ev with a separate random table. In H, we check if the inputs form a valid signature, and if so ensure that the hash table R is consistent with the random table from Ev. Because of this repair, the view of the adversary remains the same, and there is no distinguishing advantage between G_0 and G_1. This is the case because the partially blind signature is unique – there is only ever one repair for a (t, x) input pair.

In G_2, we track the number of calls to BlindEv for each tag and the number of repairs for each tag that occur in H. If the number of repairs for a tag ever exceeds the number of BlindEv calls, then a bad flag is set. In G_2, the game is aborted if the flag is set, while G_1 repairs and continues. By an identical-until-bad argument via the fundamental lemma of game playing [BR06], the distinguishing advantage between G_1 and

```
$\underline{\mathsf{HSig.Setup}(\lambda)}$
Return $\mathsf{DS.Setup}(\lambda)$

$\underline{\mathsf{HSig.Req}^{\mathsf{P}\times\mathsf{H}}(pk, t, x)}$
$(req, st_{\mathsf{DS}}) \leftarrow\$ \mathsf{DS.Req}^{\mathsf{P}}(pk, t, x)$
$st \leftarrow (t, x, st_{\mathsf{DS}})$
Return $(req, st)$

$\underline{\mathsf{HSig.KeyGen}^{\mathsf{P}\times\mathsf{H}}(pp)}$
Return $\mathsf{DS.KeyGen}^{\mathsf{P}}(pp)$

$\underline{\mathsf{HSig.Finalize}^{\mathsf{P}\times\mathsf{H}}(rep; st)}$
$(t, x, st_{\mathsf{DS}}) \leftarrow st$
$\sigma \leftarrow\$ \mathsf{DS.Finalize}^{\mathsf{P}}(rep; st_{\mathsf{DS}})$
Return $\mathsf{H}(t, x, \sigma)$

$\underline{\mathsf{HSig.BlindEv}^{\mathsf{P}\times\mathsf{H}}(sk, t, req)}$
Return
$\mathsf{DS.BlindSign}^{\mathsf{P}}(sk, t, req)$

$\underline{\mathsf{HSig.Ev}^{\mathsf{P}\times\mathsf{H}}(sk, t, x)}$
$\sigma \leftarrow\$ \mathsf{DS.Sign}^{\mathsf{P}}(sk, t, x)$
Return $\mathsf{H}(t, x, \sigma)$
```

Figure 18: The generic $\mathsf{HSig}[\mathsf{DS}]$ transform to construct a POPRF from any partially blind signature scheme DS.

```
$\underline{\mathsf{S.Init}(\lambda, pp)}$
$R \leftarrow [\cdot]$
$st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(\lambda)$
$(sk, pk) \leftarrow\$ \mathsf{DS.KeyGen}(pp)$
$st_{\mathsf{S}} \leftarrow (pp, pk, sk, R, st_{\mathsf{P}})$
Return $(st_{\mathsf{S}}, pk)$

$\underline{\mathsf{S.BlindEv}^{\textsc{LimEv}}(t, req : (pp, pk, sk, R, st_{\mathsf{P}}))}$
Return $\mathsf{DS.BlindSign}^{\mathsf{P}}(sk, t, req)$

$\underline{\mathsf{S.P}^{\textsc{LimEv}}(x : (pp, pk, sk, R, st_{\mathsf{P}}))}$
Return $\mathsf{P.Eval}(x : st_{\mathsf{P}})$

$\underline{\mathsf{S.H}^{\textsc{LimEv}}(t, x, \sigma : (pp, pk, sk, R, st_{\mathsf{P}}))}$
If $(t, x, \sigma) \notin R$ then
    If $\mathsf{DS.Ver}(pk, t, x, \sigma)$ then $R[t, x, \sigma] \leftarrow \textsc{LimEv}(t, x)$
    Else $R[t, x, Y] \leftarrow\$ \{0,1\}^{\lambda}$
Return $R[t, x, Y]$
```

Figure 19: Simulator $\mathsf{S}[\mathsf{DS}]$ for PRF security of $\mathsf{HSig}[\mathsf{DS}]$.

G_2 is bounded by the probability the bad flag is set. Furthermore, $G_2 = \mathrm{POPRF}^{\mathcal{A}_{\mathrm{prf}},0}_{\mathsf{HSig},\mathsf{P}\times\mathsf{H},\mathsf{S}}(\lambda)$, since its abort in H corresponds to the simulator calling $\textsc{LimEv}$ without any budget. Thus, we conclude the proof by constructing an adversary for the one-more unforgeability game that wins whenever the bad flag is set.

The unforgeability adversary $\mathcal{A}_{\text{om-unf}}$ simulates the $\textsc{BlindEv}$ oracle using its own $\textsc{BlindSign}$ oracle. $\mathcal{A}_{\text{om-unf}}$ stores the sets of all (t, x, σ) valid triples that are passed in to H keyed by the public tag t. If the number of valid triples for a tag exceeds the number of calls to $\textsc{BlindEv}$ for a tag, i.e., the case where the bad flag is set, $\mathcal{A}_{\text{om-unf}}$ returns the valid (x, σ) message-signature pairs along with the given tag to win the game.

Request privacy. We prove the following theorem:

Theorem 15 *Let $\mathcal{A}_{\text{po-priv2}}$ be a $(\mathsf{P}\times\mathsf{H})$-model* POPRIV2 *adversary against $\mathsf{HSig}[\mathsf{DS}]$. Then we give a P-model adversary $\mathcal{A}_{\text{blind}}$ such that*

$$\mathsf{Adv}^{\text{po-priv2}}_{\mathsf{HSig}[\mathsf{DS}],\mathsf{P}\times\mathsf{H},\mathcal{A}_{\text{po-priv2}}}(\lambda) \leq \mathsf{Adv}^{\text{blind}}_{\mathsf{DS},\mathsf{P},\mathcal{A}_{\text{blind}}}(\lambda)\,,$$

where $\mathcal{A}_{\text{blind}}$ makes the same number of queries to its respective request and finalize oracles as $\mathcal{A}_{\text{po-priv2}}$ and $T(\mathcal{A}_{\text{po-priv2}}) \approx T(\mathcal{A}_{\text{blind}})$.

Proof: $\mathcal{A}_{\text{blind}}$ is a wrapper adversary around the blindness security game that simulates $\mathcal{A}_{\text{po-priv2}}$'s environment perfectly. $\mathcal{A}_{\text{blind}}$ simply forwards $\mathcal{A}_{\text{po-priv2}}$'s oracle queries to $\textsc{Req}$ and $\textsc{Fin}$ to its own respective queries, and in the case of $\textsc{Fin}$ performs the final hash, before forwarding the response back to $\mathcal{A}_{\text{po-priv2}}$.

G Security of ZSS Partially Blind Signature

In this section, we provide the first complete security proof for the ZSS partially blind signature scheme [ZSS03]. Our 3HashSDHI POPRF construction follows closely the ZSS partially blind signature scheme, but differs in one significant way. 3HashSDHI uses a NIZK to prove correctness of the blind evaluation, while ZSS uses a pairing verification check (which also gives it its public verifiability property). We show the one-more unforgeability of ZSS is implied by our new one-more gap SDHI assumption, however, we must amend the (m, n)-OM-GAP-SDHI assumption for the bilinear pairing group setting including the extra group elements needed for verification. To be complete, we show that the amended assumption is secure with respect to a similarly generalized uber assumption for the bilinear pairing setting [BFL20].

There are two other small changes we make to original ZSS construction. For clarity, we refer to the original construction by $\mathsf{ZSS1}$ and our modified version as $\mathsf{ZSS2}$. First, we replace the multiplicative blinding

```
$\underline{\mathsf{ZSS2.Setup}(\lambda)}$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi) \leftarrow \mathsf{BGGen}(\lambda)$
$pp \leftarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi)$
Return $pp$

$\underline{\mathsf{ZSS2.KeyGen}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}()}$
$z \leftarrow\$ \mathbb{Z}_p$
$sk \leftarrow z$ ;  $pk \leftarrow g_1^z$
Return $(sk, pk)$

$\underline{\mathsf{ZSS2.Sign}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}(sk, t, m)}$
$\sigma \leftarrow \mathsf{H}_1(m)^{1/(sk+\mathsf{H}_2(t))}$
Return $\sigma$

$\underline{\mathsf{ZSS2.Ver}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}(pk, t, m, \sigma)}$
Return $\left(e\left(g_1^{\mathsf{H}_2(t)} pk, \sigma\right) = e(g_1, \mathsf{H}_1(m))\right)$
```

```
$\underline{\mathsf{ZSS2.Req}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}(pk, t, m)}$
$r \leftarrow\$ Z_p$ ;  $B \leftarrow \mathsf{H}_1(m)^r$
Return $((pk, r, t, m), B)$
                                        $\xrightarrow{B}$
                                                $\underline{\mathsf{ZSS2.BlindSign}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}(sk, t, B)}$
                                                $k \leftarrow sk + \mathsf{H}_2(t)$
                                                $B' \leftarrow B^{1/k}$
                                                Return $B'$
                                        $\xleftarrow{B'}$
$\underline{\mathsf{ZSS2.Finalize}_{pp}^{\mathsf{H}_1 \times \mathsf{H}_2}(B', \pi; (pk, r, t, m))}$
$\sigma \leftarrow (B')^{1/r}$
If not $\mathsf{ZSS2.Ver}^{\mathsf{H}_1 \times \mathsf{H}_2}(pk, t, m, \sigma)$ return $\perp$
Return $\sigma$
```

Figure 20: The ZSS2 partially blind signature construction, a minor modification to the construction from [ZSS03]. Algorithms have implicit input the parameters $pp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ that describe the groups used. Hash function H_1 has range $\mathbb{G}_2$ and H_2 has range $\mathbb{Z}_p$.

```
$\underline{\text{Game } (m,n)\text{-OM-GAP-SDHI}_{\mathsf{BGGen}}^{\mathcal{A}}(\lambda)}$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi) \leftarrow \mathsf{BGGen}(\lambda)$
$sk \leftarrow\$ \mathbb{Z}_p$ ;  $[y_i]_i^m \leftarrow\$ [\mathbb{Z}_p]_i^m$
$(st_{\mathcal{A}}, [c_i]_i^n) \leftarrow\$ \mathcal{A}_1(p, \mathbb{G})$
Require $\forall_{i \neq j}^n c_i \neq c_j$
$(\gamma, [Z_i, \alpha_i, \beta_i]_i^\ell) \leftarrow\$ \mathcal{A}_2^{\mathrm{SDH},\mathrm{SDDH}}\left(g_1, g_2, g_1^{sk}, g_2^{sk}, [g_1^{y_i}, g_2^{y_i}]_i^m : st_{\mathcal{A}}\right)$
Require $q_\gamma < \ell \ \wedge\ \forall_{i \neq j}^\ell \alpha_i \neq \alpha_j \ \wedge\ \forall_i^\ell \beta_i \in [1,2]$
Return $[Z_i]_i^\ell = \left[g_{\beta_i}^{y_{\alpha_i}/(sk+c_\gamma)}\right]_i^\ell$

$\underline{\text{Oracle SDH}(B, i, j)}$
Require $i \in [1, n]$
Require $B \in \mathbb{G}_j$
$q_i \leftarrow q_i + 1$
$Z \leftarrow B^{1/(sk+c_i)}$
Return $Z$

$\underline{\text{Oracle SDDH}(Y, Z, i, j)}$
Require $Y, Z \in \mathbb{G}_j$
Return $Z = Y^{1/(sk+c_i)}$
```

Figure 21: The one-more gap strong Diffie-Hellman inversion security game adapted for bilinear groups.

approach with exponential blinding (consistent with 3HashSDHI) which is both simpler and has security benefits in the non-verification case [JKX21]. ZSS incorrectly claim that the blindness property is not satisfied under the exponential blinding approach, as we explain below where we discuss blindness.

Nevertheless, the multiplicative blinding approach of ZSS is still interesting as it enables a client-side performance improvements of fixed-base exponentiation with precomputation over variable-base exponentiation [JKX21]. We present this alternate blinding approach that can be applied to both 3HashSDHI and ZSS in Appendix I. The second difference is that we change the signature structure to not include the public tag in the first hash input. This change doesn't affect the security of the signature and allows for the blind signing process to be initiated by the client before the public tag is known (allowing for server-chosen public tags). The construction is given in pseudocode as ZSS2 in Figure 20.

Bilinear pairing groups. We follow the notation of [BFL20]. (1) Groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order p. (2) Group element g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$. (3) Pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a computable map with the following properties: *Bilinearity*: $\forall\ u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$, and *Non-degeneracy*: $e(g_1, g_2) \neq 1$. (4) ϕ is an isomorphism $\phi : \mathbb{G}_1 \to \mathbb{G}_2$, and ψ is an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$. We assume an efficient setup algorithm that on input security parameter λ, generates a bilinear group, $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi) \leftarrow \mathsf{BGGen}(1^\lambda)$, where $|p| = \lambda$. If there exist efficiently computable isomorphisms $\phi : \mathbb{G}_1 \to \mathbb{G}_2$ and $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, the bilinear group is called *Type 1*; it is called *Type 2* if there is no efficiently computable isomorphism ϕ, and called *Type 3* if there is neither ϕ nor ψ. In the AGM, group element representations including elements computed from isomorphisms are explicitly indicated along with the representation of the element passed into the isomorphism.

We also extend the (m,n)-OM-GAP-SDHI assumption for bilinear pairing groups in the game given in Figure 21. The game is identical as before except the adversary is given group elements in both $\mathbb{G}_1$

```
Game $m$-UBER$^{\mathcal{A}}_{\mathsf{BGGen}}(\lambda)$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi) \leftarrow \mathsf{BGGen}(\lambda)$
$Q_1 \leftarrow \{\}$ ; $Q_2 \leftarrow \{\}$
$\vec{x} = [x_i]_i^m \leftarrow\$ [\mathbb{Z}_p]_i^m$
$(j, U, \mu(\vec{X})) \leftarrow\$ \mathcal{A}^{\text{Ev},\text{Decide}}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi)$
Return $\left(U = g_j^{\mu(\vec{x})} \wedge (Q_1 \cup Q_2) \perp\!\!\!\perp \{\mu(\vec{X})\}\right)$

Oracle Ev$(j, \rho(\vec{X}))$
Require $i \in [1, 2]$
$Q_j \leftarrow Q_j \cup \{\rho(\vec{X})\}$
Return $g_j^{\rho(\vec{x})}$

Oracle Decide$(j, \rho(\vec{X}), [Y_i]_i^n)$
$\vec{y} = [y_i]_i^n \leftarrow \left[\log_{g_j} Y_i\right]_i^n$
Return $\rho(\vec{y}) \equiv_p 0$
```

Figure 22: The interactive, flexible-output, polynomial uber assumption with decision oracle adapted for bilinear groups.

and $\mathbb{G}_2$ for *sk* and y_i. The SDH and SDDH oracles are also parameterized by an input $j \in [1, 2]$ to determine which group the oracle query takes place over. We provide a corollary that the bilinear-version (m, n)-OM-GAP-SDHI is implied by a bilinear-version m-UBER assumption [BFL20], which in turn has been shown to be implied by a bilinear-version q-DL assumption (both reductions make use of the AGM). The pseudocode games describing these assumptions are given in Figure 22 and Figure 23, respectively. The uber assumption allows the adversary to specify a group in which to receive an evaluation and tracks the polynomial queries made to each group. The winning condition is stated in the most general way (for Type 1 bilinear groups), in which the winning polynomial must be independent from queries made to both groups; relaxed winning conditions can be formulated for Type 2 and 3 bilinear groups in which depending on the group of the winning element, the winning polynomial must be independent from only one query set (see [BFL20] for details). However, the general formulation is sufficient for proving security of (m, n)-OM-GAP-SDHI regardless of the underlying bilinear group type.

We provide the following corollaries for the security of the bilinear version of (m, n)-OM-GAP-SDHI. We reuse the query budget notation from Section 5.

```
Game $q$-DL$^{\mathcal{A}}_{\mathsf{BGGen}}(\lambda)$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi) \leftarrow \mathsf{BGGen}(\lambda)$
$x \leftarrow\$ \mathbb{Z}_p$
$x' \leftarrow\$ \mathcal{A}\left(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, \phi, \psi, \left[g_1^{x^i}, g_2^{x^i}\right]_{i=1}^q\right)$
Return $x = x'$
```

Figure 23: The q-type discrete log security game.

Corollary 16 *For any algebraic adversary $\mathcal{A}_{\text{sdhi}}$ of (m, n)-OM-GAP-SDHI with query budget $(\vec{q} = [q_1, \ldots, q_n]$, $q_{\mathsf{SDDH}})$, and any BGGen outputting $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2)$, where g_1 and g_2 are uniformly chosen elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, we give adversary $\mathcal{A}_{\text{uber}}$ such that*

$$\mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{BGGen},\mathcal{A}_{\text{sdhi}}}(\lambda) \leq (q_{\max} + 1) \cdot \mathsf{Adv}^{(m+1)\text{-uber}}_{\mathsf{BGGen},\mathcal{A}_{\text{uber}}}(\lambda) + \frac{q}{p},$$

where $q = \sum_i^n q_i$ and $q_{\max} = \max\{q_i\}_i^n$. Also, $\mathcal{A}_{\text{uber}}$ makes at most q queries to its polynomial evaluation oracle with maximum degree $q + 1$, and outputs a polynomial of degree at most q. Further, $T(\mathcal{A}_{\text{sdhi}}) \approx T(\mathcal{A}_{\text{uber}})$.

Again, when combined with a basic reduction from [BFL20], this gives us the following immediate corollary.

Corollary 17 *For any algebraic adversary $\mathcal{A}_{\text{sdhi}}$ of (m, n)-OM-GAP-SDHI, with query budget $(\vec{q} = [q_1, \ldots, q_n]$, $q_{\mathsf{SDDH}})$, and any BGGen outputting $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2)$, where g_1 and g_2 are uniformly chosen elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, we give adversary $\mathcal{A}_{\text{dl}}$ such that*

$$\mathsf{Adv}^{(m,n)\text{-om-gap-sdhi}}_{\mathsf{BGGen},\mathcal{A}_{\text{sdhi}}}(\lambda) \leq (q_{\max} + 1) \cdot \mathsf{Adv}^{(q+1)\text{-dl}}_{\mathsf{BGGen},\mathcal{A}_{\text{dl}}}(\lambda) + \frac{q}{p},$$

where $q = \sum_i^n q_i$ and $q_{\max} = \max\{q_i\}_i^n$. Further, $T(\mathcal{A}_{\text{sdhi}}) \approx T(\mathcal{A}_{\text{dl}})$.

These corollaries mirror exactly the results from Section 5. The proof of Corollary 16 follows the same approach as the proof in Appendix A; the polynomial independence argument is not affected by the which group the evaluations are given in (and for Type 1 bilinear groups, it is irrelevant).

One-more unforgeability. We provide the following theorem for the one-more unforgeability of ZSS2. A similar result can be stated for ZSS1, and its proof would be almost identical. Here, the ideal model $\mathsf{P} = \mathsf{H}_1 \times \mathsf{H}_2$ consists of random oracles $\mathsf{H}_1 : * \rightarrow \mathbb{G}_2$ and $\mathsf{H}_2 : * \rightarrow \mathbb{Z}_p$ for $(p, \mathbb{G}_2)$ determined by $\mathsf{BGGen}(\lambda)$.

```
Game G
$i \leftarrow 0$ ; $R'_1 \leftarrow [\cdot]$
$R_1 \leftarrow [\cdot]$ ; $R_2 \leftarrow [\cdot]$
$pp \leftarrow\!\!\$\ \mathsf{ZSS2.Setup}(\lambda)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Req},\text{Fin},\mathsf{P}}(pp)$
Return $b'$

Oracle Req$(pk, t, m_0, m_1)$
$i \leftarrow i + 1$
$t_i \leftarrow t$ ; $pk_i \leftarrow pk$
$m_{i,0} \leftarrow m_0$ ; $m_{i,1} \leftarrow m_1$
$r_{i,0} \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $req_0 \leftarrow g_2^{r_{i,0}}$
$r_{i,1} \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $req_1 \leftarrow g_2^{r_{i,1}}$
Return $(req_b, req_{1-b})$

Oracle Fin$(j, rep, rep')$
If $j > i$ then return $\perp$
If not $e\left(g_1^{\mathsf{H}_2(t_i)} pk_i, (rep)^{1/r_{i,b}}\right) = e(g_1, g_2)$ then
    Return $\perp$
If not $e\left(g_1^{\mathsf{H}_2(t_i)} pk_i, (rep')^{1/r_{i,1-b}}\right) = e(g_1, g_2)$ then
    Return $\perp$
$\sigma_b \leftarrow (rep)^{R'_1[m_{i,b}]/r_{i,b}}$
$\sigma_{1-b} \leftarrow (rep')^{R'_1[m_{i,1-b}]/r_{i,1-b}}$
Return $(\sigma_0, \sigma_1)$

Oracle $\mathsf{H}_1(m)$
If $m \notin R_1$ then
    $h \leftarrow\!\!\$\ \mathbb{Z}_p$
    $R_1[m] \leftarrow g_2^h$
    $R'_1[m] \leftarrow h$
Return $R_1[m]$

Oracle $\mathsf{H}_2(t)$
If $t \notin R_2$ then
    $R_2[t] \leftarrow\!\!\$\ \mathbb{Z}_p$
Return $R_2[t]$
```

Figure 24: Game transition used in blindness security proof of ZSS2 in Theorem 19. Algorithms have implicit input $pp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$.

Theorem 18 *Let $\mathcal{A}_{\text{om-unf}}$ be a* P*-model one-more unforgeability adversary against* ZSS1*. Then we give an adversary $\mathcal{A}_{\text{sdhi}}$ such that*

$$\mathsf{Adv}^{\text{om-unf}}_{\mathsf{ZSS2},\mathsf{P},\mathcal{A}_{\text{om-unf}}}(\lambda) \leq \mathsf{Adv}^{(q_{\mathsf{H}_1}, q_t)\text{-om-gap-sdhi}}_{\mathsf{BGGen},\mathcal{A}_{\text{sdhi}}}(\lambda) + \frac{q_t^2}{p},$$

where q_{H_1} is the maximum distinct queries $\mathcal{A}_{\text{om-unf}}$ makes to H_1 and q_t is the maximum distinct t queries $\mathcal{A}_{\text{om-unf}}$ makes to H_2 and BlindSign*. Further, $T(\mathcal{A}_{\text{om-unf}}) \approx T(\mathcal{A}_{\text{sdhi}})$.*

Proof: We define $\mathrm{G}_0 = \text{OM-Unf}^{\mathcal{A}_{\text{om-unf}}}_{\mathsf{ZSS2},\mathsf{P}}(\lambda)$. We first transition to a game G_1 in which we disallow collisions in the output of H_2, i.e., we replace sampling from $\mathbb{Z}_p$ for new range values with sampling without replacement from $\mathbb{Z}_p$. Recall that at most q_t unique t values are queried by $\mathcal{A}_{\text{om-unf}}$ in the course of the game, and so a standard birthday analysis establishes an upper bound of $q_t(q_t - 1)/2p$.

We build adversary $\mathcal{A}_{\text{sdhi}}$ to simulate exactly $\mathcal{A}_{\text{om-unf}}$'s environment in G_1. Adversary $\mathcal{A}_{\text{sdhi}}$ simulates the ideal primitives by using its input values $g_2^{y_i}$ as random outputs of H_1 and randomly chosen unique c_j values as outputs of H_2. Adversary $\mathcal{A}_{\text{sdhi}}$ simulates BlindSign by looking up (or programming) the appropriate c_j and forwarding a query to its SDH oracle, returning the result. When $\mathcal{A}_{\text{om-unf}}$ returns, $\mathcal{A}_{\text{sdhi}}$ matches the returned messages to its programmed outputs in H_1, and returns the signatures along with appropriate $g_2^{y_i}$ index, specifying group $j = 2$. Whenever $\mathcal{A}_{\text{om-unf}}$ wins G_1, then $\mathcal{A}_{\text{sdhi}}$ also wins (q_{H_1}, q_t)-OM-Gap-SDHI because if the signatures are valid, then so will be the SDHI checks.

Blindness. ZSS describe a possible linking attack against exponential blinding [ZSS03, Footnote 1]. Adapted to our version of the scheme and notation, their claimed attack checks if $e(\sigma, B) = e(B', \mathsf{H}_1(m))$, and if so concludes that B, B' is the partially blinded protocol transcript used to generate σ. (Note, that this computation is only possible for type 1 pairings, or type 2 pairings if one uses the map ψ.) However, this check *always* passes for any B, B' (with $B' = B^{1/(sk + \mathsf{H}_2(t))}$), regardless of whether the transcript B, B' is associated with the signature σ.

Instead, we provide the following theorem for the blindness of ZSS2, which uses exponential blinding.

Theorem 19 *For any* Blind *adversary $\mathcal{A}_{\text{blind}}$ in the* P*-model against* ZSS2*, we have that*

$$\mathsf{Adv}^{\text{blind}}_{\mathsf{ZSS2},\mathsf{P},\mathcal{A}_{\text{blind}}}(\lambda) = 0.$$

Proof:

We transition to a game G shown in Figure 24 in which the requests output from Req are swapped from $req_d = \mathsf{H}_1(m_d)^{r_d}$ to $req_d = g_2^{r_d}$ for $d \in \{0, 1\}$. Observe that the outputs of Req are now independent of challenge bit b. Fin is also changed. In the original game, Fin outputs valid signatures if the responses given by the adversary are equal to $rep = req^{1/(sk + \mathsf{H}_2(t))}$ where $pk = g_1^{sk}$; this is confirmed by the pairing verification equation. Otherwise the output is $\perp$. In G, the same pairing check is made to see if the adversary responded in the correct manner. If pairing check succeeds, then a valid signature is created and returned

```
Game $\mathrm{POUNIQ}^{\mathcal{A}}_{\mathsf{Fn},\mathsf{P}}(\lambda)$
$q \leftarrow 0$ ;   $R \leftarrow [\cdot]$
$pp \leftarrow\!\!\$\ \mathsf{Fn.Setup}(\lambda)$
$st_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}(\lambda)$
$(i, j, rep_i, rep_j) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{REQ},\mathsf{P}}(pp)$
Require $1 \le i \le j \le q$
$(st_i, pk_i, t_i, x_i) \leftarrow R[i]$
$(st_j, pk_j, t_j, x_j) \leftarrow R[j]$
$y_i \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(rep_i : st_i)$
$y_j \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(rep_j : st_j)$
If $y_i = \bot$ or $y_j = \bot$ then
    Return 0
Return $((pk_i, t_i, x_i) = (pk_j, t_j, x_j)) \wedge (y_i \neq y_j)$

Oracle $\mathrm{REQ}(pk, sk, t, x)$
Require $\mathsf{Fn.Wellformed}(pk, sk)$
$(st, req) \leftarrow \mathsf{Fn.Req}^{\mathsf{P}}(pk, t, x)$
$q \leftarrow q + 1$
$R[q] \leftarrow (st, pk, t, x)$
Return $req$
```

Figure 25: Security definition for uniqueness of PO-PRF outputs despite a malicious server. The highlighted code is included for simplicity to address rogue key attacks using the knowledge of secret key model [Bol03].

by simulating H_1 and using the discrete log of the programmed hash outputs. Importantly, the output valid signatures, do not include the blinding factor r_d and are thus, independent of the challenge bit and the outputs from REQ. Therefore, the outputs of G are distributed identically to the blind security game, and since they do not depend on the challenge bit, the adversary can have no advantage.

H Uniqueness from Request Privacy

We formalize uniqueness via game POUNIQ shown in Figure 25. It is parameterized by a security parameter λ, a PO-PRF scheme Fn, an adversary $\mathcal{A}$, and an ideal primitive P. The adversary can query a request oracle REQ to generate honest requests for adversary-chosen public key, public input t and private input x. The adversary finishes by outputting a pair of numbers indicating one or two queries to REQ, as well as two response messages rep_i, rep_j. The adversary wins if the triple pk, t, x are the same for both queries but the client can be tricked into outputting distinct y values. We let the advantage of a POUNIQ adversary $\mathcal{A}$ be defined by

$$\mathsf{Adv}^{\text{po-uniq}}_{\mathsf{Fn},\mathcal{A},P}(\lambda) = \Pr\left[\,\mathrm{POUNIQ}^{\mathcal{A}}_{\mathsf{Fn},\mathsf{P}}(\lambda) \Rightarrow 1\,\right]$$

where the probability space is taken over the random choices made in the game.

Looking ahead, our proof of uniqueness will depend on our correctness property. However, our correctness property is stated to hold for wellformed public key pairs, while uniqueness captures a malicious adversary that may use a rogue public key. To bridge this gap, we require the adversary to prove knowledge of secret keys. For simplicity, we model this by asking the adversary to provide the secret key as input to the request oracle, shown highlighted in Figure 25, following the knowledge of secret key model [Bol03]. This model can be instantiated by including extractable proofs of knowledge of secret keys, from which the secret key can be extracted and the proof may proceed in the KOSK model. In the game pseudocode, we use a wellformed predicate to capture this check, i.e., for a discrete log public key X and secret key x, checks $X = g^x$.

The next theorem captures that POUNIQ security is implied by POPRIV2 security.

Theorem 20 *Let $\mathcal{A}$ be a POUNIQ adversary against a PO-PRF scheme Fn. Then we give a POPRIV2 adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\textit{po-uniq}}_{\mathsf{Fn},\mathcal{A},P}(\lambda) \le \mathsf{Adv}^{\text{po-priv2}}_{\mathsf{Fn},\mathcal{B},P}(\lambda)\,,$$

where $\mathcal{B}$ makes the same number of queries to its respective request oracle as $\mathcal{A}$, makes at most 2 queries to its finalize oracle, and $T(\mathcal{A}) \approx T(\mathcal{B})$.

Proof: The reduction works as follows given a POUNIQ adversary $\mathcal{A}$. Adversary $\mathcal{B}$ (given in pseudocode in Figure 26) on input pp runs $\mathcal{A}$ on input pp. Upon receiving a request query $\mathrm{REQ}(pk, t, x)$, adversary $\mathcal{B}$ makes oracle call $\mathrm{REQ}(pk, t, x, x)$ to its own request oracle. Adversary $\mathcal{B}$ receives back two request messages

```
Adversary $\mathcal{B}^{\text{REQ},\text{FIN},\mathsf{P}}(pp)$
$q \leftarrow 0$ ; $R \leftarrow [\cdot]$
$st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(\lambda)$
$(i, j, rep_{i,0}, rep_{j,0}) \leftarrow\$ \mathcal{A}^{\text{SIMREQ},\mathsf{P}}(pp)$
Require $1 \le i \le j \le q$
$(pk_i, sk_i, t_i, x_i, req_{i,1}) \leftarrow R[i]$
$(pk_j, sk_j, t_j, x_j, req_{j,1}) \leftarrow R[j]$
$rep_{i,1} \leftarrow\$ \mathsf{Fn.BlindEv}^{\mathsf{P}}(sk_i, t_i, req_{i,1})$
$rep_{j,1} \leftarrow\$ \mathsf{Fn.BlindEv}^{\mathsf{P}}(sk_j, t_j, req_{j,1})$
$(y_{i,0}, y_{i,1}) \leftarrow \text{FIN}(i, rep_{i,0}, rep_{i,1})$
$(y_{j,0}, y_{j,1}) \leftarrow \text{FIN}(j, rep_{j,0}, rep_{j,1})$
$y \leftarrow \mathsf{Fn.Ev}^{\mathsf{P}}(sk_i, t_i, x_i)$
If $y_{i,0} \ne y_{i,1}$ then return $y_{i,0} == y$
Return $y_{j,0} == y$

Oracle $\text{SIMREQ}(pk, sk, t, x)$
Require $\mathsf{Fn.Wellformed}(pk, sk)$
$(req_0, req_1) \leftarrow\$ \text{REQ}(pk, t, x, x)$
$q \leftarrow q + 1$
$R[q] \leftarrow (pk, sk, t, x, req_1)$
Return $req_0$
```

Figure 26: Adversary $\mathcal{B}$ against POPRIV2 used in uniqueness security proof.

```
$\mathsf{3H.Req}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3 \times \mathsf{H}_4}(pk, t, x)$
$r \leftarrow\$ Z_p$ ; $B \leftarrow \mathsf{H}_1(x) \cdot \left(g^{\mathsf{H}_3(t)} pk\right)^r$
Return $((pk, r, t, x), B)$

        $\xrightarrow{\quad B \quad}$

$\mathsf{3H.BlindEv}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3 \times \mathsf{H}_4}(sk, t, B)$
    $k \leftarrow sk + \mathsf{H}_3(t)$
    $B' \leftarrow B^{1/k}$
    $\pi \leftarrow\$ \Sigma_{\mathcal{R}}.\mathsf{Prove}^{\mathsf{H}_4}(k, (g, g^k, B', B))$
    Return $(B', \pi)$

        $\xleftarrow{\quad B' \quad}$

$\mathsf{3H.Finalize}^{\mathsf{H}_1 \times \mathsf{H}_2 \times \mathsf{H}_3 \times \mathsf{H}_4}(B', \pi; (pk, r, t, x))$
$Y \leftarrow B'/g^r$
Require $\Sigma_{\mathcal{R}}.\mathsf{Ver}^{\mathsf{H}_4}((g, g^{\mathsf{H}_3(t)} \cdot pk, B', B), \pi)$
$Z \leftarrow \mathsf{H}_2(t, x, Y)$
Return $Z$
```

Figure 27: An alternative multiplicative blinding protocol [ZSS03] that provides client-side performance benefits [JKX21].

req_0, req_1 and returns req_0 to $\mathcal{A}$. $\mathcal{B}$ also stores the pk, sk, t, x, req_1 associated with the query. Recall the sk is known due to the knowledge of secret key model. This perfectly simulates $\mathcal{A}$'s request oracle.

When $\mathcal{A}$ returns $(i, j, rep_{i,0}, rep_{j,0})$, $\mathcal{B}$ retrieves pk_i, sk_i, t_i, x_i, $req_{i,1}$ and pk_j, sk_j, t_j, x_j, $req_{j,1}$. We consider the case where $\mathcal{A}$ wins uniqueness game. In this case, $(pk_i, sk_i, t_i, x_i) = (pk_j, sk_j, t_j, x_j)$ so we drop the subscripts.

$\mathcal{B}$ calls FIN by responding to $req_{i,0}$ with the $\mathcal{A}$-provided $rep_{i,0}$ and responding to $req_{i,1}$ with a $rep_{i,1}$ generated honestly. By the correctness property, we know that the value returned from the honest flow with $rep_{i,1}$ is equal to $y = \mathsf{Fn.Ev}(sk, t, x)$. If FIN returns two different values, then by matching which position y is returned reveals the challenge bit and allows $\mathcal{B}$ to win the game.

If FIN returns two identical values, then we repeat the above for the j query. If $\mathcal{A}$ wins the uniqueness game, then we know that FIN will not return identical values for the j query, so the challenge bit will be revealed, and $\mathcal{B}$ wins the game. Thus, $\mathcal{B}$ outputs the correct challenge bit in the POPRIV2 game whenever $\mathcal{A}$ wins POUNIQ.

I Multiplicative Blinding

In this section we present an alternate client-side blinding approach applying a blinding factor using group multiplication rather than group exponentiation. This approach was originally proposed by ZSS [ZSS03]. We show that it is a secure blinding alternative for both the 3HashSDHI POPRF and for the ZSS partially blind signature. The multiplicative blind evaluation protocol is given for 3HashSDHI in Figure 27. It can be adapted for ZSS by including $pk_2 = g_2^{sk}$ as a second component of the public key to be used for blinding.

The multiplicative blinding proofs for 3H and ZSS1 follow symmetrically as those for Theorem 7 and Theorem 19. To create a request for multiplicative blinding that is independent of challenge bit b, we construct the request as $B \leftarrow g \cdot \left(g^{\mathsf{H}_3(t)} pk\right)^r$. This gives the following corollaries:

Corollary 21 *Let $\mathcal{A}_{\text{po-priv2}}$ be a* POPRIV2 *adversary in the* P*-model against* $\mathsf{3H}$ *with multiplicative blinding that makes at most q queries to* FIN. *We give in the proof below a* $\text{SOUND}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{H}_4}$ *adversary* $\mathcal{B}_{\text{sound}}$ *such that*

$$\mathsf{Adv}^{\text{po-priv2}}_{\mathsf{3H},\mathsf{P},\mathcal{A}_{\text{po-priv2}}}(\lambda) \leq 4q \cdot \mathsf{Adv}^{\text{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{H}_4,\mathcal{B}_{\text{sound}}}(\lambda)) \ .$$

Further, $T(\mathcal{B}_{\text{sound}}) \approx T(\mathcal{A}_{\text{po-priv2}})$.

Corollary 22 *For any* BLIND *adversary* $\mathcal{A}_{\text{blind}}$ *in the* P*-model against* $\mathsf{ZSS1}$ *with multiplicative blinding, we have that* $\mathsf{Adv}^{\text{blind}}_{\mathsf{ZSS1},\mathsf{P},\mathcal{A}_{\text{blind}}}(\lambda) = 0$.