A Fast and Simple Partially Oblivious PRF, with Applications

Nirvan Tyagi[1], Sofía Celi[2], Thomas Ristenpart[1], Nick Sullivan[2], Stefano Tessaro[3], and Christopher A. Wood[2]

[1] Cornell Tech
[2] Cloudflare
[3] University of Washington

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.

Keywords: verifiable oblivious pseudorandom functions, Diffie-Hellman inversion, anonymous tokens, blind signatures

1 Introduction

An oblivious pseudorandom function (OPRF) [23, 32] 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 [20], 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 [18], private set intersection (PSI) for checking compromised credentials [36, 42], de-identified authenticated logging [26], and password authenticated key exchange [28,30]. 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. While one can simply use a standard PRF with a master key to derive a key for each t (see, e.g., [29]), there is no known way to make this efficiently verifiable. The only known POPRF supporting efficient verification relies on bilinear pairings [20], 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. [28], that is the de facto standard used in practice, with the Dodis-Yampolskiy (DY) verifiable random function [19]. Our POPRF is also closely related to a signature scheme suggested by Zhang, Safavi-Naini, and Susilo (ZSS) [46, 47]. 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 [24]. 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 [20] and previous OPRF formalizations [28, 32]. 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 [20].

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 [28] and prior notions for partially blind signatures [1]. 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 [22, 28, 33]), 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 associate an output $f_{sk}(t, x)$ to the particular oblivious request transcript used to produce $f_{sk}(t, x)$. 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. This is similar to the property of *verifiability*, which, informally, states that servers prove in zero-knowledge that output $f_{sk}(t, x)$ corresponds to the public key associated with 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. [28], 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 [15] 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 [18, 26, 42] and is on track to become a standard [17].

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 [46]. 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 [17,18] (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 [4,8]. 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 $\mathrm{SDH}(\cdot, i)$. The challenge is that we do *not* restrict the number of queries $\mathrm{SDH}(X, j)$ 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 $\mathrm{SDH}(\cdot, i)$ query). Ultimately, we show in the algebraic group model (AGM) [24] that the assumption reduces to one of the uber assumptions from Bauer, Fuchsbauer, and Loss [3], 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) [37,40].

In terms of concrete security, our analysis 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 [16] 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 [1]. 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 [45]. 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 [45].

As previously mentioned, the 3HashSDHI construction is closely related to the ZSS partially blind signature scheme [46], which uses pairings. As we explain in Section 4, the original unforgeability proof is however incorrect. We rectify this situation and provide the first formal analysis of the security of ZSS using our new techniques in the full version [43]. To the best of our knowledge, this result provides the most efficient partially-blind signature supporting arbitrary public metadata; previous RSA-based constructions [1,2] require the set of public metadata tags to be incorporated during parameter setup, previous Schnorr-based constructions [2, 25] are vulnerable in the concurrent signing setting [7], and other existing constructions are more heavyweight as they are tailored for the anonymous credential setting [10].

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 the full version [43].

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 [12, 18] 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 [13,14].

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 [36, 42] 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 [6] allow a client and server to establish a shared session key authenticated by a short password. Strong asymmetric PAKE (SaPAKE) protocols [30] additionally ensure that the server stores what amounts to private salted hashes of user passwords. Since these salted hashes are private, an attacker cannot perform offline pre-computation that would lead to instantaneous compromise of user passwords upon the event of a server breach. The OPAQUE [30] SaPAKE protocol uses an OPRF as one of its core components; it is currently being considered for standardization by the IETF [34]. 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 [24]. 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 $[X_0, \ldots, X_n] \in \mathbb{G}$ that $\mathcal{A}$ has received so far, then, when producing group element Z, $\mathcal{A}$ must also provide a list $[z]_i = [z_0, \ldots, z_n]$ 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 [27]. 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 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* [27] 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:

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

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

```
RO.Init(1^λ)
    $T \leftarrow [\cdot]$
    Return $T$

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 [25], 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 $\perp$ 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, $\pi \leftarrow\$ \mathsf{NiZK.Prove}^{\mathsf{P}}_{pp}(w, x)$[4]. 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

[4] P is an arbitrary ideal primitive.

```
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 $\mathrm{ZK}^{\mathcal{A},b}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda)$
    $pp_1 \leftarrow\$ \mathsf{NiZK.Setup}(\lambda)$
    $st_{\mathsf{P}} \leftarrow\$ \mathsf{P.Init}(\lambda)$
    $(st_{\mathsf{S}}, pp_0) \leftarrow\$ \mathsf{S.Init}(\lambda)$
    $b' \leftarrow\$ \mathcal{A}^{\textsc{Prim},\textsc{Prove}}(pp_b)$
    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 $y_b$
```

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

with perfect completeness. A proof system has *perfect completeness* if for all $(x, w) \in \mathcal{R}$,

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

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 $(x, w) \in \mathcal{R}$. Knowledge soundness is defined by the security game $\textsc{Sound}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P}}(\lambda)$ (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 $\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P},\mathcal{A}}(\lambda) = \Pr[\textsc{Sound}^{\mathcal{A}}_{\mathsf{NiZK},\mathcal{R},\mathsf{P}}(\lambda) = 1]$ 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 $\mathrm{ZK}^{\mathcal{A},b}_{\mathsf{NiZK},\mathcal{R},\mathsf{S},\mathsf{P}}(\lambda)$ (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

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

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" [9] 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 [15] (shown in Figure 2); it is perfectly complete, computationally sound, and perfectly zero-knowledge in the random oracle model.

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

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

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

Fig. 2: Description of Chaum-Pedersen discrete log equality Sigma protocol [15].

3 Partially Oblivious Pseudorandom Functions

We provide a new formalization for POPRFs, including syntax, semantics, and security. Our formalization builds off that from [20], 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:

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

Security. We introduce three new security definitions for POPRFs. We use code-based games mostly following the framework of Bellare and Rogaway [5].

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 [28], 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 [1].

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 [41] 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

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

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

```
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 $\bot$
```

```
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 $y_b$
```

Fig. 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.

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 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, left 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 $\mathcal{A}$.

```
Game 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 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 $\bot$
$y_b \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(st_{j,b}, rep)$
$y_{1-b} \leftarrow \mathsf{Fn.Finalize}^{\mathsf{P}}(st_{j,1-b}, rep')$
If $y_0 = \bot$ or $y_1 = \bot$ then
    Return $\bot$
Return $(y_0, y_1)$
```

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

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., [19]). 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 the full version [43].

Relation to partially blind signatures. POPRFs are related to two-move partially blind signatures, which were introduced by Abe and Fujisaki [1]. 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 the full version [43].

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. [28] 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 the full version [43] 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., [1,2]. The only unique scheme we are aware of is due to Zhang, Safavi-Naini, and Susilo (ZSS) [46], 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 the full version [43]).

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 $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{\gamma_2}$, and $\mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^{\gamma_3}$ 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

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

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

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 [46]. 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 $\mathsf{H}_{\mathbb{G}_2} : \{0,1\}^* \to \mathbb{G}_2$ hashes onto the group $\mathbb{G}_2$ and H_3 is as defined above for 3HashSDHI. We use ZSS1 to differentiate from our suggested modifications, which we call ZSS2 and discuss in the full version [43]. As can be seen, 3HashSDHI uses essentially the same structure, combined with a final hash but we dispense with 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 [46]: (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 the full version [43] 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 [19] 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 [31]. 3HashSDHI is not compatible with the multiplicative blinding protocol used by 2HashDH, however the full version [43] presents an alternative multiplicative blinding protocol [46] that enjoys the same performance benefits.

Miao et al. construct an oblivious evaluation protocol for the DY PRF [38]. Their approach makes use of the additive-homomorphic Camenisch-Shoup encryption scheme [11] 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 [20] 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.

Jarecki et al. [29] propose construction of a POPRF F from OPRF F_1 and PRF F_2 as $\mathsf{F}.\mathsf{Ev}(sk, t, x) = \mathsf{F}_1.\mathsf{Ev}(\mathsf{F}_2.\mathsf{Ev}(sk, t), x)$. They go on to propose a construction instantiating F_1 with 2HashDH and F_2 with a symmetric PRF (e.g., HMAC); this construction is highly efficient but does not provide verifiability due to the non-algebraic choice of F_2. 3HashSDHI can be thought of as following this approach composing 2HashDH with the algebraic VRF of Dodis-Yampolskiy [19].

We also compare to the recent attribute-based verifiable OPRF (AB-VOPRF) suggested by Huang et al. of Facebook [26] 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.

Again following the blueprint of [29], Facebook's proposed construction [26] of an AB-VOPRF combines the 2HashDH OPRF with the Naor-Reingold PRF [39]. Evaluation is defined by

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

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 [41] 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 [35]. 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 [41].

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 note that use of random oracles here allows us to avoid the more complex proof techniques used by Dodis and Yampolskiy [19], and indeed it is unclear what kind of analysis could work for this step without random oracles.

In the second step, we analyze the security of our new assumption, showing that, in the Algebraic Group Model (AGM) [24] it reduces to one of the uber assumptions from Bauer, Fuchsbauer, and Loss (BFL) [3]. 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. Requiring fixed query budgets is an artifact of existing analysis approaches to Diffie-Hellman inversion-like assumptions; we leave it as an open question whether the per-tag query budget can be handled adaptively.

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 $\mathsf{Adv}^{q\text{-dl}}_{\mathsf{GGen},\mathcal{A}}(\lambda) = \Pr\left[\, q\text{-DL}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda) \Rightarrow \mathsf{true} \,\right]$.

As a convenient middle layer, we rely on BFL's "Uber-assumption" [3], 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

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

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 $Z = Y^{1/(sk+c_i)}$
```

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

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

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

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

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

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 $\mathsf{Adv}^{m\text{-uber}}_{\mathsf{GGen},\mathcal{A}}(\lambda) = \Pr\big[\, m\text{-UBER}^{\mathcal{A}}_{\mathsf{GGen}}(\lambda) \Rightarrow \mathsf{true}\,\big]$.

We prove the following theorem in the full version [43]. Here and subsequently we use '$\approx$' to denote that runtimes are equal up to small constant factors.

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

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

Theorem 1. *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 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 $q = \sum^n_i q_i$ and $q_{\max} = \max\{q_i\}^n_i$. 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 [3], this gives us the following immediate corollary.

Corollary 1. *For any algebraic adversary* $\mathcal{A}_{\mathrm{sdhi}}$ *of* (m,n)-OM-GAP-SDHI*, with query budget* $(\vec{q} = [q_1, \dots, q_n],\ q_{\mathsf{SDDH}})$*, and any* GGen *outputting* $(p, g, \mathbb{G})$*, where* g *is a uniformly chosen element of* $\mathbb{G}$*, we give adversary* $\mathcal{A}_{\mathrm{dl}}$ *such that*

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

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

The main difficulty of the proof of Theorem 1 in the full version [43] 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 $\mathrm{SDH}(\cdot, \gamma)$. 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).[5] Specifically, let us describe exponents of the elements provided to $\mathcal{A}_2$ as formal polynomials X_0 (standing for the secret key) and $X_1, \dots, X_m$ (for the values $y_1, \dots, y_m$). Initially, the adversary has these polynomials available, and now a call to $\mathrm{SDH}(P, i)$ 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 $\mathrm{SDH}(P, i)$ adds a new rational function to this set of available rational functions. In other words, consecutive queries induce a *transcript* τ consisting of the initial functions $X_0, X_1, \dots, X_m$, and the functions returned by SDH. The goal of the adversary is to ensure that, for some γ, the span[6] of τ contains $\ell > q_\gamma$ functions of the form

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

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

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

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

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 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 the full version [43] 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 $\mathcal{A}_{\text{prf}}$ 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 $\mathcal{A}_{\text{prf}}$ to H_1 or H_2;
- n is the maximum number of distinct t values queried by $\mathcal{A}_{\text{prf}}$ to BLINDEV, H_2, or H_3;

- $\vec{q} = [\vec{q}_1, \ldots, \vec{q}_n]$ is a vector where each $\vec{q}_i$ is the maximum number of queries by $\mathcal{A}_{\mathrm{prf}}$ to $\textsc{BlindEv}(t_i, \mathit{req})$ for any req and where $t_1, \ldots, t_n$ 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 $\mathcal{A}_{\mathrm{prf}}$ 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 1.

Theorem 2. *Let $\mathcal{A}_{\mathrm{prf}}$ 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 $\mathcal{A}_{\mathrm{zk}}$, an adversary $\mathcal{A}_{\mathrm{sdhi}}$, and a simulator S such that*

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

Adversary $\mathcal{A}_{\mathrm{zk}}$ makes q_{H_4} queries to its random oracle and $\mathcal{A}_{\mathrm{sdhi}}$ has query budget $(\vec{q}, q_{\mathsf{H}_2})$. Further, $T(\mathcal{A}_{\mathrm{prf}}) \approx T(\mathcal{A}_{\mathrm{zk}}) \approx T(\mathcal{A}_{\mathrm{sdhi}})$.

A detailed proof is given in the full version [43]. It proceeds via a sequence of games, starting with the real world $\mathrm{POPRF}^{\mathcal{A}_{\text{po-prf}},1}_{\mathsf{3H},\mathsf{P},\mathsf{S}}(\lambda)$ 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-$\textsc{Gap}$-SDHI: the only difference between this game and the ideal world $\mathrm{POPRF}^{\mathcal{A}_{\mathrm{prf}},0}_{\mathsf{3H},\mathsf{P},\mathsf{S}[\mathsf{S}_\Sigma]}(\lambda)$ 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 $\mathcal{A}_{\mathrm{prf}}$'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 2. *Let $\mathcal{A}_{\mathrm{prf}}$ 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 $\mathcal{A}_{\mathrm{dl}}$ and simulator S *such that*

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

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

Concrete security and parameter selection. Corollary 2 is interpreted best in the generic-group model (GGM) [37, 40], as this yields an absolute bound in terms of $\mathcal{A}_{\mathrm{prf}}$'s resources. The advantage of a generic algorithm $\mathcal{A}_{\mathrm{dl}}$ 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 $\mathsf{Adv}^{(q+1)\text{-dl}}_{\mathsf{GGen},\mathcal{A}_{\mathrm{dl}}}(\lambda)\le (T+q+2)^2(q+1)/(p-1)$ (see, e.g., [3] 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 [16] extracts[7] the secret key from q BLINDEVAL queries in time $\sqrt{p/q}$, as long as q divides $p-1$ or $p+1$.

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 2 is inherited from Theorem 1 and is due to our inability to *efficiently* find, within the adversary $\mathcal{A}_{\mathrm{uber}}$, 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 $\Omega(q^2_{\max})$. In other words, we could make the proof tight with respect to

[7] Define $x=(sk+\mathsf{H}_3(t))^{-1}$ 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, $sk=x^{-1}-\mathsf{H}_3(t)$.

advantage while losing tightness with respect to time complexity. While in our proof flow in the AGM 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 (enumerating polynomials and testing independence) 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 3. *For any* POPRIV1 *adversary* $\mathcal{A}_{\text{po-priv1}}$ *against* $\mathsf{3H}$ *(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 $\mathsf{3H.Req}$. Observe that in game G the values returned by $\textsc{Req}$ are independent of the challenge bit b. Then we have that

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

■

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 $\mathsf{3H}$, adversary $\mathcal{A}_{\text{po-priv2}}$ can pick $sk \in \mathbb{G}$ arbitrarily, let $pk = g^{sk}$, and then query $\textsc{Req}(pk, t, x_0, x_1)$ 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 $\textsc{Fin}(B^{1/(sk+\mathsf{H}_3(t))}, g)$ 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 4. *Let $\mathcal{A}_{\text{po-priv2}}$ be a* POPRIV2 *adversary in the* P*-model against* $\mathsf{3H}$ *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. $B'_1 \neq B_{j,b}^{1/(sk+\mathsf{H}_3(t_j))}$ 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. $B'_2 \neq B_{j,1-b}^{1/(sk+\mathsf{H}_3(t_j))}$ 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

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

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 "$\text{G0} \Rightarrow b$" 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., $rep \leftarrow B_{j,b}^{sk_j}$ and $rep' \leftarrow B_{j,1-b}^{sk_j}$ 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 [5]. We now bound the probability that $\Pr[\text{G0}_{\mathsf{bad}}$ 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

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

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 $\Pr[\mathrm{G1} \Rightarrow b] = 1/2$. Combining all the above yields the advantage statement in the theorem.

∎

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 [17], Pythia [20], and the recent attribute-based VOPRF (ABVOPRF) from Facebook [26]. 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 [44] 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 [21] 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. We defer the results of our benchmarks to the full version [43].

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. In the full version [43], we discuss further three previously mentioned motivating applications: anonymous one-time-use tokens, password breach alerting, and password-based authenticated key exchange.

Acknowledgments

The authors would like to thank Tjerand Silde, Martin Strand, and Tancrede Lepoint for helpful discussions on early versions of this work. This work was supported in part by NSF grants CNS-1930117 (CAREER), CNS-1926324, CNS-2026774, CNS-2120651, a Sloan Research Fellowship, a JP Morgan Faculty Award, and a Facebook PhD Fellowship.

References

1. Abe, M., Fujisaki, E.: How to date blind signatures. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 1163, pp. 244–251. Springer (1996)
2. Abe, M., Okamoto, T.: Provably secure partially blind signatures. In: CRYPTO. Lecture Notes in Computer Science, vol. 1880, pp. 271–286. Springer (2000)
3. Bauer, B., Fuchsbauer, G., Loss, J.: A classification of computational assumptions in the algebraic group model. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 12171, pp. 121–151. Springer (2020)
4. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. J. Cryptol. **16**(3), 185–215 (2003)
5. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer (2006)
6. Bellovin, S., Merritt, M.: Augmented encrypted key exchange: a password based protocol secure against dictionary attacks and password file compromise. In: CCS. pp. 244–250. ACM (1993)
7. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 12696, pp. 33–53. Springer (2021)
8. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer (2003)
9. Camenisch, J.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (1998)
10. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: CRYPTO. Lecture Notes in Computer Science, vol. 3152, pp. 56–72. Springer (2004)
11. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: CRYPTO. Lecture Notes in Computer Science, vol. 2729, pp. 126–144. Springer (2003)
12. Celi, S., Davidson, A., Faz-Hernández, A.: Privacy Pass Protocol Specification. Internet-Draft draft-ietf-privacypass-protocol-00, Internet Engineering Task Force (Jan 2021), https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-protocol-00, work in Progress
13. Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic macs and keyed-verification anonymous credentials. In: CCS. pp. 1205–1216. ACM (2014)
14. Chase, M., Perrin, T., Zaverucha, G.: The signal private group system and anonymous credentials supporting efficient verifiable encryption. In: CCS. pp. 1445–1459. ACM (2020)
15. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO. Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer (1992)
16. Cheon, J.H.: Security analysis of the strong diffie-hellman problem. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 1–11. Springer (2006)
17. Davidson, A., Faz-Hernández, A., Sullivan, N., Wood, C.A.: Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups. Internet-Draft draft-irtf-cfrg-voprf-05, Internet Engineering Task Force (Nov 2020), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-05, work in Progress
18. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. Proc. Priv. Enhancing Technol. **2018**(3), 164–180 (2018)
19. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 3386, pp. 416–431. Springer (2005)
20. Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 547–562. USENIX Association (2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/everspaugh
21. Faz-Hernández, A., Kwiatkowski, K.: Introducing CIRCL: An Advanced Cryptographic Library. Cloudflare (Jun 2019), https://github.com/cloudflare/circl
22. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 60–77. Springer (2006)
23. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: TCC. Lecture Notes in Computer Science, vol. 3378, pp. 303–324. Springer (2005)
24. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer (2018)
25. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 63–95. Springer (2020)
26. Huang, S., Iyengar, S., Jeyaraman, S., Kushwah, S., Lee, C.K., Luo, Z., Mohassel, P., Raghunathan, A., Shaikh, S., Sung, Y.C., Zhang, A.: PrivateStats: De-Identified Authenticated Logging at Scale (Jan 2021), https://research.fb.com/wp-content/uploads/2021/01/PrivateStats-De-Identified-Authenticated-Logging-at-Scale_final.pdf
27. Jaeger, J., Tyagi, N.: Handling adaptive compromise for practical encryption schemes. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 12170, pp. 3–32. Springer (2020)

28. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8874, pp. 233–253. Springer (2014)
29. Jarecki, S., Krawczyk, H., Resch, J.K.: Threshold partially-oblivious prfs with applications to key management. IACR Cryptol. ePrint Arch. p. 733 (2018)
30. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 10822, pp. 456–486. Springer (2018)
31. Jarecki, S., Krawczyk, H., Xu, J.: On the (in)security of the diffie-hellman oblivious PRF with multiplicative blinding. In: Public Key Cryptography (2). Lecture Notes in Computer Science, vol. 12711, pp. 380–409. Springer (2021)
32. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC. Lecture Notes in Computer Science, vol. 5444, pp. 577–594. Springer (2009)
33. Kiayias, A., Zhou, H.: Equivocal blind signatures and adaptive uc-security. In: TCC. Lecture Notes in Computer Science, vol. 4948, pp. 340–355. Springer (2008)
34. Krawczyk, H., Lewi, K., Wood, C.A.: The OPAQUE Asymmetric PAKE Protocol. Internet-Draft draft-irtf-cfrg-opaque-02, Internet Engineering Task Force (Feb 2021), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-02, work in Progress
35. Kreuter, B., Lepoint, T., Orrù, M., Raykova, M.: Anonymous tokens with private metadata bit. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 12170, pp. 308–336. Springer (2020)
36. Li, L., Pal, B., Ali, J., Sullivan, N., Chatterjee, R., Ristenpart, T.: Protocols for checking compromised credentials. In: CCS. pp. 1387–1403. ACM (2019)
37. Maurer, U.M.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3796, pp. 1–12. Springer (2005). https://doi.org/10.1007/11586821_1, https://doi.org/10.1007/11586821_1
38. Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 12172, pp. 3–33. Springer (2020)
39. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: FOCS. pp. 458–467. IEEE Computer Society (1997)
40. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 256–266. Springer (1997)
41. Silde, T., Strand, M.: Anonymous tokens with public metadata and applications to private contact tracing. IACR Cryptol. ePrint Arch. **2021**, 203 (2021)
42. Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P.G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., Bursztein, E.: Protecting accounts from credential stuffing with password breach alerting. In: USENIX Security Symposium. pp. 1556–1571. USENIX Association (2019)
43. Tyagi, N., Celi, S., Ristenpart, T., Sullivan, N., Tessaro, S., Wood, C.A.: A fast and simple partially oblivious prf, with applications. IACR Cryptol. ePrint Arch. p. 864 (2021)
44. de Valence, H., Grigg, J., Tankersley, G., Valsorda, F., Lovecruft, I., Hamburg, M.: The ristretto255 and decaf448 Groups. Internet-Draft draft-irtf-cfrg-ristretto255-decaf448-00, Internet Engineering Task Force (Oct 2020), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-ristretto255-decaf448-00, work in Progress
45. Wilander, J., Taubeneck, E., Knox, A., Wood, C.: Consider using blinded signatures for fraud prevention - Private Click Measurement (2020), https://github.com/privacycg/private-click-measurement/issues/41
46. Zhang, F., Safavi-Naini, R., Susilo, W.: Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In: INDOCRYPT. Lecture Notes in Computer Science, vol. 2904, pp. 191–204. Springer (2003)
47. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 2947, pp. 277–290. Springer (2004)