On the (In)Security of the Diffie-Hellman Oblivious PRF with Multiplicative Blinding

Stanisław Jarecki[1], Hugo Krawczyk[2], and Jiayu Xu[3]

[1] University of California, Irvine, stasio@ics.uci.edu
[2] Algorand Foundation, hugokraw@gmail.com
[3] George Mason University, jiayux@uci.edu

Abstract. Oblivious Pseudorandom Function (OPRF) is a protocol between a client holding input x and a server holding key k for a PRF F. At the end, the client learns $F_k(x)$ and nothing else while the server learns nothing. OPRF's have found diverse applications as components of larger protocols, and the currently most efficient instantiation, with security proven in the UC model, is $F_k(x) = H_2(x, (H_1(x))^k)$ computed using so-called *exponential blinding*, i.e. the client sends $a = (H_1(x))^r$ for random r, the server responds $b = a^k$, which the client unblinds as $v = b^{1/r}$ to compute $F_k(x) = H_2(x, v)$.
However, this protocol requires two variable-base exponentiations on the client, while a more efficient *multiplicative blinding* scheme replaces one or both client exponentiations with fixed-base exponentiation, leading to the decrease of the client's computational cost by a factor between two to six, depending on pre-computation.
We analyze the security of the above OPRF with multiplicative blinding, showing surprising weaknesses that offer attack avenues which are not present using exponential blinding. We characterize the security of this OPRF implementation as a "Correlated OPRF" functionality, a relaxation of UC OPRF functionality used in prior work.
On the positive side, we show that the Correlated OPRF suffices for the security of OPAQUE, the asymmetric PAKE protocol, hence allowing OPAQUE the computational advantages of multiplicative blinding. Unfortunately, we also show examples of other OPRF applications which become insecure when using such blinding. The conclusion is that usage of multiplicative blinding for $F_k(x)$ defined as above, in settings where correct value g^k (needed for multiplicative blinding) is not authenticated, and OPRF inputs are of low entropy, must be carefully analyzed, or avoided all together. We complete the picture by showing a simple and safe alternative definition of function $F_k(x)$ which offers (full) UC OPRF security using either form of blinding.

1 Introduction

An *Oblivious Pseudorandom Function* (OPRF) scheme consists of a Pseudorandom Function (PRF) F for which there exists a two-party protocol between a server S holding a PRF key k and a client C holding an input x

through which C learns $F_k(x)$ and S learns nothing (in particular, nothing about the input x or the output $F_k(x)$). More generally, the security properties of the PRF, namely indistinguishability from a random function under polynomially many queries, must be preserved by the protocol. The OPRF notion was introduced explicitly in [8] but constructions, particularly those based on *blinded DH*, were studied earlier (e.g., [5, 23, 7]). OPRF has been formally defined under different models [8, 18, 10, 11] with the last two works framing them in the Universally Composable (UC) framework [4]. The OPRF notion has found many applications, and recently such applications have been proposed for actual deployment in practice, including the Privacy Pass protocol [6] and the OPAQUE password-authenticated key exchange protocol [17]. This gave rise to standardization proposals for OPRFs [25] and the OPAQUE protocol [22, 21, 26], which further motivates understanding the costs and benefits of possible OPRF implementations.

Exponential vs. multiplicative blinding in Hashed Diffie-Hellman PRF.[4] In several of the above mentioned applications, the underlying PRF is instantiated with a *(Double) Hashed Diffie-Hellman* construction (2HashDH) [11], namely:

$$F_k(x) = H_2(x, (H_1(x))^k) \tag{1}$$

where hash functions H_1, H_2 are defined respectively as $H_1 : \{0,1\}^* \rightarrow \mathbb{G}\backslash\{1\}$ and $H_2 : \{0,1\}^* \times \mathbb{G} \rightarrow \{0,1\}^\tau$ for a multiplicative group $\mathbb{G}$ of prime order q, and the PRF key k is a random element in $\mathbb{Z}_q$, while τ is a security parameter. The protocol for the oblivious computation of 2HashDH used e.g. in [7, 2, 10, 11] employs the so-called *exponential blinding* method, i.e. protocol Exp-2HashDH shown in Fig. 1: Client C sends to server S its input x blinded as $a = (H_1(x))^r$, for $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, and then unblinds the server's response $b = a^k$ as $v = b^{1/r}$ [$= (a^k)^{1/r} = (((H_1(x))^r)^k)^{1/r} = (H_1(x))^k$] and outputs $H_2(x, v)$. It is easy to see that the client's input is perfectly hidden from the server because if $H_1(x) \neq 1$ then a is a random element in $\mathbb{G}$ independent from x.

An alternative *multiplicative blinding* technique, denoted Mult-2HashDH, is shown in Fig. 2. The protocol is an equivalent of Chaum's technique for blinding RSA signatures: Given generator g of group $\mathbb{G}$, the client blinds its input as $a = H_1(x) \cdot g^r$, and using the server's *public key* $z = g^k$ corresponding to the PRF key k, the client unblinds the server's response $b = a^k$ as $v = b \cdot z^{-r}$ [$= a^k \cdot (g^k)^{-r} = (H_1(x) \cdot g^r)^k \cdot g^{-kr} = (H_1(x))^k$]. It is easy to see that this blinding hides x with perfect security, as in the case of Exp-2HashDH.

Comparing the computational cost of the two techniques, we see that both require a single variable-base exponentiation for the server. However, for the client, Exp-2HashDH requires two variable-base exponentiations (for blinding and unblinding) while Mult-2HashDH involves a single *fixed-base* exponentiation for blinding and a variable-base exponentiation (to the base z) for unblinding.

[4] In the context of additive groups, "multiplicative" would be replaced with "additive" and "exponential" with "scalar-multiplicative". A less confusing terminology could refer to these as fixed-base and var-base blindings, respectively.

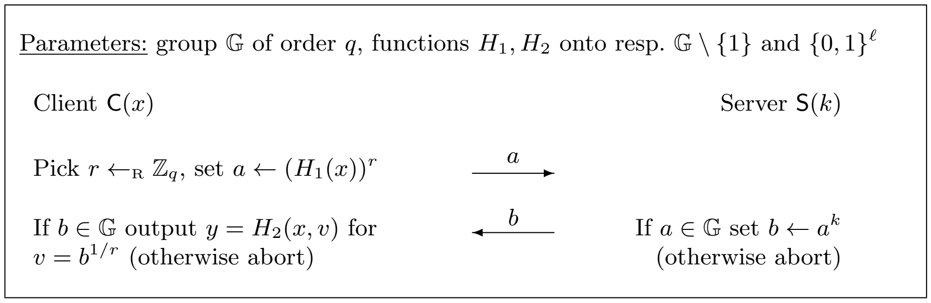

Parameters: group $\mathbb{G}$ of order q, functions H_1, H_2 onto resp. $\mathbb{G} \setminus \{1\}$ and $\{0,1\}^\ell$
Client $\mathsf{C}(x)$
Server $\mathsf{S}(k)$
Pick $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $a \leftarrow (H_1(x))^r$
a
If $b \in \mathbb{G}$ output $y = H_2(x, v)$ for $v = b^{1/r}$ (otherwise abort)
b
If $a \in \mathbb{G}$ set $b \leftarrow a^k$ (otherwise abort)

Fig. 1. Exp-2HashDH: Oblivious PRF using *Exponential* Blinding [11]

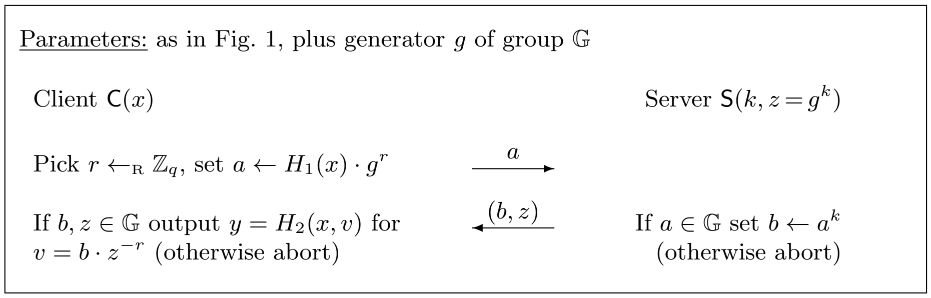

Parameters: as in Fig. 1, plus generator g of group $\mathbb{G}$
Client $\mathsf{C}(x)$
Server $\mathsf{S}(k, z = g^k)$
Pick $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $a \leftarrow H_1(x) \cdot g^r$
a
If $b, z \in \mathbb{G}$ output $y = H_2(x, v)$ for $v = b \cdot z^{-r}$ (otherwise abort)
(b, z)
If $a \in \mathbb{G}$ set $b \leftarrow a^k$ (otherwise abort)

Fig. 2. Mult-2HashDH: Oblivious PRF using *Multiplicative* Blinding

In applications where the client stores z,[5] the latter exponentiation can use fixed-base optimization, reducing the client's total computation to two-fixed base exponentiations. Given that exponentiation with a fixed base is about 6-7 times faster than with a variable base (cf. [3, 13]), Mult-2HashDH becomes at least 1.7 faster than Exp-2HashDH and 6x faster if z is stored at the client and treated as a fixed base. On the other hand, in cases where the client does not hold z, Mult-2HashDH requires the server to store z and send it with each execution of the OPRF protocol. This cost may not be significant in some cases but in constrained environments where bandwidth and/or storage is a costly resource (e.g., mobile and IoT scenarios) [9], Exp-2HashDH may be preferred. Fortunately, 2HashDH allows an application to choose the blinding mechanism that best fits its needs, possibly choosing one technique or the other depending on the network setting and client configuration.

These are good news for performance and implementation flexibility, but regarding security, things are not as straightforward, as we explain next.

Is multiplicative blinding secure? On the face of it, it would seem that exponential and multiplicative blindings are equivalent, functionally and security-wise, thus allowing for performance optimization and flexibility as discussed above. However, determining the security of Mult-2HashDH turns

[5] For example, in a password protocol such as OPAQUE [17], a user can cache values z corresponding to servers it accesses frequently, e.g., Google, Facebook, etc.

out to be non-trivial, showing unexpected attack avenues which are not present in Exp-2HashDH. In particular, while Exp-2HashDH has been proven to satisfy the UC OPRF notion from [11], protocol Mult-2HashDH is *not secure* under this same definition. The problem is, broadly speaking, that the dependency of the protocol on z implies that multiplicative blinding does not ensure full independence between OPRF instances indexed by different public keys.[6]

Let us elaborate. In protocol Exp-2HashDH, server's response b to the client's message $a \neq 1$ defines a unique key $k = \mathsf{DL}(a, b)$ for which C computes $y = F_k(x)$. (Since client's output is $y = H_2(x, v)$ for $v = b^{1/r}$ and $a = (H_1(x))^r$, it follows that $v = a^{k/r} = (H_1(x))^k$ and therefore $y = F_k(x)$ for $k = \mathsf{DL}(a, b)$.) In other words, server's response b commits the server to a single value k, hence to a unique function $F_k(x)$. This commitment to a unique function is central to the OPRF UC modeling from [11]. The same, however, does *not* hold for Mult-2HashDH where the server's response (b, z) to the client's message a gives the attacker an additional degree of freedom in manipulating C's output $y = H(x, b \cdot z^{-r})$. Specifically, response (b, z) given a determines pair (δ, z) where $\delta = b/a^k$ for $k = \mathsf{DL}(g, z)$, thus leading to the following function:

$$F_{(\delta,z)}(x) \triangleq H_2(x, \delta \cdot (h_x)^k) \quad \text{for} \quad z = g^k \text{ and } h_x = H_1(x) \tag{2}$$

which an honest C computes on its input x given S's response (b, z) in the Mult-2HashDH protocol. Indeed, if $a = h_x \cdot g^r$, $z = g^k$ and $\delta = b/a^k$ then

$$v = b \cdot z^{-r} = b \cdot (g^k)^{-r} = b \cdot (g^r)^{-k} = b \cdot (a/h_x)^{-k} = (b/a^k) \cdot (h_x)^k = \delta \cdot (h_x)^k$$

The important point is that value $\delta = b/a^k$ for $k = \mathsf{DL}(g, z)$ introduces a multiplicative shift in the value v computed by C. Moreover, an adversarial S can exploit this shift to create correlated responses that leak information on the client's input. In particular, for any choice of client input $\bar{x}$, an attacker S can find values $\delta_1, \delta_2, k_1, k_2$ such that

$$\delta_1 \cdot (h_{\bar{x}})^{k_1} = \delta_2 \cdot (h_{\bar{x}})^{k_2} \quad \text{for} \quad z_1 = g^{k_1}, z_2 = g^{k_2} \text{ and } h_{\bar{x}} = H_1(\bar{x}) \tag{3}$$

Using these values the attacker can respond to the first client's query a_1 with $(b_1, z_1) = (\delta_1 {a_1}^{k_1}, g^{k_1})$, and to a second query a_2 with $(\delta_2 {a_2}^{k_2}, g^{k_2})$, leading C to compute values v_1, v_2 that coincide if C's input is $x = \bar{x}$ and do not coincide if $x \neq \bar{x}$. In other words, $F_{(\delta_1,z_1)}(\bar{x}) = F_{(\delta_2,z_2)}(\bar{x})$, showing that in contrast to the family $\{F_k\}$ defined by equation (1), the function family $\{F_{(\delta,z)}\}$ defined by equation (2) is *not* a family of *independent* random functions in ROM.[7]

Potential vulnerabilities. The core advantage a corrupt server may gain by exploiting the above correlations is the ability to test whether a given value of x has been input by the client in a previous interaction with the server. Our

[6] The potential insecurity of multiplicative blinding as UC OPRF was pointed out in [17], which left its security analysis as an open question.

[7] Note that an honest server's response $(b, z) = (a^k, g^k)$ corresponds to $\delta = 1$ and the evaluated function $F_{(1,z)}$ is identical to the intended function F_k.

analysis of Mult-2HashDH shows that the server can test at most one such input per interaction. For OPAQUE, this property suffices to *prove the security of the protocol with Mult-2HashDH.* The intuitive reason is that in OPAQUE, a malicious server already has the ability to test guesses for the client's inputs (a password in the case of OPAQUE) with each interaction with the client, thus the above attack based on correlation does not add to the attacker's power. In contrast, in Section 7 we show examples of applications where the correlated nature of Mult-2HashDH opens attack avenues not available with exponential blinding. This demonstrates that the two OPRF implementations, Exp-2HashDH and Mult-2HashDH, are not equivalent vis-à-vis security, and replacing one with another within some application needs to be analyzed on a per-case basis, as we do here for OPAQUE.

Modeling Mult-2HashDH as Correlated OPRF. To analyze the security of applications that use Mult-2HashDH, we show that there are limits on the correlations which an adversary can create among the functions effectively evaluated in the Mult-2HashDH protocol. Specifically, each pair of functions can be correlated only as in equation (3) and *only on one* argument x. We prove this formally by introducing a relaxation of the UC OPRF functionality of [11] which we call *Correlated OPRF.* The purpose of this relaxation is to model the exact nature of function correlations which multiplicative blinding gives to a malicious server. We show that Mult-2HashDH realizes the Correlated OPRF functionality under the Gap$^+$-OMDH assumption in ROM, a mild strengthening of the Gap-OMDH assumption which sufficed for Exp-2HashDH to satisfy the UC OPRF functionality [11].

Security of OPAQUE under both blindings. Based on the UC modeling of Mult-2HashDH as a Correlated OPRF, we prove the OPAQUE strong asymmetric PAKE protocol [17] secure using 2HashDH with multiplicative blinding. (*Strong* asymmetric PAKE is secure against pre-computation of password hashes before server compromise.) Specifically, we show that OPAQUE remains secure if the OPRF building block it uses is relaxed from the UC OPRF notion of [11] to the Correlated OPRF defined here. This means that the asymmetric PAKE standard being defined by the IETF on the basis of OPAQUE [22, 21, 26] can use the 2HashDH function and leave the choice of exponential or multiplicative blinding to individual implementations.

We believe that the same holds for another construction from [17], which shows that a composition of UC OPRF and any asymmetric PAKE results in a strong asymmetric PAKE. This transformation was proven secure using UC OPRF, implemented by Exp-2HashDH and we believe that this result can also be "upgraded" to the case of UC Correlated OPRF, i.e. using Mult-2HashDH, but we leave the formal verification of that claim to future work.

When is it safe to use Mult-2HashDH? In cases where the client has access to the value g^k in some authenticated/certified form, such as in applications requiring a Verifiable OPRF [10], e.g., Privacy Pass [6], one can use (1) with either blinding. For multiplicative blinding, one just uses the authenticated z in the unblinding. However, when z is received from the server in unauthenticated way, much care is needed, and security under multiplicative blinding needs to be proven on a per-application basis. Even then, small changes in applications and implementations may turn this mechanism insecure as evidenced by the case of using OPAQUE with a threshold OPRF which we show in Section 7 to be insecure if used with Mult-2HashDH. As a rule of thumb, *it seems prudent to advise not to use Mult-2HashDH in setting with unauthenticated* g^k *and where the input to the OPRF is taken from a low-entropy space.*

An alternative OPRF specification. Another fix is to replace function 2HashDH defined in eq. (1) with the following simple modification, where $z = g^k$ is included under the hash, which is secure using *either* blinding:

$$F'_k(x) = H_2(x, z, H_1(x)^k) \quad \text{where} \quad z = g^k \tag{4}$$

It can be shown that this scheme avoids the correlation attacks[8], and therefore can be proven secure with either blinding method as a realization of the UC OPRF functionality from [11]. The security holds even when the value z input into the hash by the client is the (unauthenticated) z received from the server.

However, while this scheme allows an implementation to choose (even at execution time) the blinding mechanism it prefers, it forces the transmission of z from server to client even in the case of exponential blinding, a drawback in constrained settings discussed above, e.g. [9]. In the case of OPAQUE, one can still use the simpler 2HashDH without transmitting z but with the subtleties and warnings surrounding security as demonstrated in this paper.[9]

2 Preliminaries

The Gap One-More Diffie-Hellman assumptions. The security of protocol Mult-2HashDH as UC Correlated OPRF relies on the interactive *Gap*$^+$ *One-More Diffie-Hellman* (Gap$^+$-OMDH) assumption, a mild strengthening of the Gap-OMDH assumption used to realize UC OPRF [11] or *verifiable* UC OPRF [10]. Let $\mathbb{G}$ be a group of prime order q, and let g be an arbitrary generator of $\mathbb{G}$. Let $(\cdot)^k$ for $k \in \mathbb{Z}_q$ denote an oracle which returns $y = x^k$ on input $x \in \mathbb{G}$. Let CDH_g denote a CDH oracle which returns g^{xy} on input (g^x, g^y). Let DDH_g denote a DDH oracle which returns 1 on input (A, B, C) s.t. $C = \mathsf{CDH}_g(A, B)$, and 0 otherwise. Let DDH_g^+ denote an oracle which returns 1 on input (A, B, A', B', C) s.t. $C = \mathsf{CDH}_g(A, B) \cdot \mathsf{CDH}_g(A', B')$, and 0 otherwise. The (N, Q)-Gap$^+$-OMDH assumption on group $\mathbb{G}$ states that for any polynomial-time algorithm $\mathcal{A}$,

$$\Pr_{k \leftarrow_{\mathrm{R}} \mathbb{Z}_q,\ h_1,\ldots,h_N \leftarrow_{\mathrm{R}} \mathbb{G}} \left[\mathcal{A}^{(\cdot)^k, \mathsf{DDH}_g^+}(g, g^k, h_1, \ldots, h_N) = (J, S) \right]$$

[8] The correlation between functions $F_{(\delta_1, z_1)}$ and $F_{(\delta_2, z_2)}$ would now require that $z_1{=}z_2$, hence $k_1{=}k_2$, in which case eq. (3) holds only if $\delta_1{=}\delta_2$, hence $(\delta_1, z_1) = (\delta_2, z_2)$.

[9] Another way for 2HashDH to realize UC OPRF with multiplicative blinding, is to add to Mult-2HashDH a zero-knowledge proof that (g, z, a, b) is a DDH tuple, but this would void the performance benefit of Mult-2HashDH.

is negligible, where $J = (j_1, \ldots, j_{Q+1})$, $S = ((h_{j_1})^k, \ldots, (h_{j_{Q+1}})^k)$, Q is the number of $\mathcal{A}$'s $(\cdot)^k$ queries, and $j_1, \ldots, j_{Q+1}$ are *distinct* elements in $\{1, \ldots, N\}$.

In other words, Gap$^+$-OMDH models the following experiment: Let $\mathcal{A}$ have access to a DDH^+ oracle and an "exponentiation to k-th power" oracle for random k in $\mathbb{Z}_q$, and the number of queries to the latter is limited by Q. $\mathcal{A}$ is given N random elements in $\mathbb{G}$ as the challenge values, and since $\mathcal{A}$ is allowed to query the exponentiation oracle Q times, it is able to compute the k-th power of any Q of the N elements, but the assumption postulates that it is infeasible that $\mathcal{A}$ computes the k-th power of any $Q+1$ of the N group elements, i.e. that it computes the k-th power of "one more" element.

The Gap-OMDH assumption is defined in the exact same way as Gap$^+$-OMDH, except $\mathcal{A}$ has access to oracle DDH_g instead of DDH_g^+. We believe that Gap$^+$-OMDH is a mild strengthening of Gap-OMDH because assuming OMDH in a group with a bilinear map implies both assumptions: Given an efficiently computable map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ s.t. $e(g^a, g^b) = e(g, g)^{ab}$, one can implement DDH_g oracle, by checking if $e(A, B) = e(g, C)$, as well as DDH_g^+ oracle, by checking if $e(A, B) \cdot e(A', B') = e(g, C)$. In the full version [15] we show that the Gap$^+$-OMDH assumption holds in the generic group model, which extends similar argument given for Gap-OMDH in [12].

3 The Correlated OPRF Functionality $\mathcal{F}_{\mathsf{corOPRF}}$

As we explain in Section 1, we will model the type of PRF-correlations which protocol Mult-2HashDH allows with a correlated OPRF functionality, and here we define it as functionality $\mathcal{F}_{\mathsf{corOPRF}}$ shown in Fig. 3. In Section 4 we will argue that protocol Mult-2HashDH, i.e. the multiplicative blinding protocol together with the PRF defined in equation (1), realizes functionality $\mathcal{F}_{\mathsf{corOPRF}}$ under Gap-OMDH assumption in ROM.

Functionality $\mathcal{F}_{\mathsf{corOPRF}}$ is a relaxation of the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ of [17], which is an adaptive extension of the UC OPRF defined in [11]. To make this relation easier to see we mark in Fig. 3 all the code fragments which are novel with respect to functionality $\mathcal{F}_{\mathsf{OPRF}}$ of [17]. Below we will first explain the basic properties which $\mathcal{F}_{\mathsf{corOPRF}}$ shares with $\mathcal{F}_{\mathsf{OPRF}}$, and then we explain the crucial differences which make $\mathcal{F}_{\mathsf{corOPRF}}$ a relaxation of $\mathcal{F}_{\mathsf{OPRF}}$.

Correlated OPRF model: basic logic. Functionality $\mathcal{F}_{\mathsf{corOPRF}}$ models OPRF in a similar way as $\mathcal{F}_{\mathsf{OPRF}}$ of [11, 17]. First, when an honest server S initializes a PRF by picking a random key, this is modeled in the ideal world via call $\textsc{Init}$ from S, which initializes a random function $F_{\mathsf{S}} : \{0,1\}^* \to \{0,1\}^\ell$. Second, the real-world S can evaluate F_{S} off-line on any argument, which is modeled in the ideal world by call $(\textsc{OfflineEval}, \mathsf{sid}, i, x, L)$ from S with $i = \mathsf{S}$ and $L = \perp$, which gives $F_{\mathsf{S}}(x)$ to S. (The role of list L, which a malicious server can make non-empty, is discussed further below.) Third, in addition to the off-line evaluation, any client C can start an on-line OPRF protocol instance with S on local input x, which is modeled by call $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}', x)$ from $\mathsf{P} = \mathsf{C}$ with $\mathsf{S}' = \mathsf{S}$, where ssid stands for *sub-session ID*, a fresh identifier of this

Public Parameters: PRF output-length ℓ, polynomial in security parameter τ.
Conventions: $\forall\, i, x$ value $F_i(x)$ is initially undefined, and if undefined $F_i(x)$ is referenced then $\mathcal{F}_{\mathsf{corOPRF}}$ sets $F_i(x) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$. Variable P ranges over all *honest* network entities and $\mathcal{A}^*$, and we assume *all corrupt entities are operated by* $\mathcal{A}^*$.

Initialization

- On $(\textsc{Init}, \mathsf{sid})$ from S, set $\mathsf{tx} \leftarrow 0$, $\mathcal{N} \leftarrow [\mathsf{S}]$, $\mathcal{E} \leftarrow \{\}$, $\mathcal{G} \leftarrow (\mathcal{N}, \mathcal{E})$.
  Ignore all subsequent $\textsc{Init}$ messages.
  Below "S" stands for the entity which sent the $\textsc{Init}$ message.

Server Compromise

- On $(\textsc{Compromise}, \mathsf{sid})$ from $\mathcal{A}^*$, declare server S as COMPROMISED.
  (If S is corrupted then it is declared COMPROMISED as well.)

Offline Evaluation

- On $(\textsc{OfflineEval}, \mathsf{sid}, i, x, \mathrm{L})$ from P do:
  (1) If $\mathsf{P} = \mathcal{A}^*$ and $i \notin \mathcal{N}$ then append i to $\mathcal{N}$ and run $\textsc{Correlate}(i, L)$;
  (2) Ignore this message if $\mathsf{P} = \mathcal{A}^*$, S is not COMPROMISED, and $(i, \mathsf{S}, x) \in \mathcal{E}$;
  (3) Send $(\textsc{OfflineEval}, \mathsf{sid}, F_i(x))$ to P if (i) $\mathsf{P} = \mathsf{S}$ and $i = \mathsf{S}$ or (ii) $\mathsf{P} = \mathcal{A}^*$ and either $i \neq \mathsf{S}$ or S is COMPROMISED.

Online Evaluation

- On $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}', x)$ from P, send $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, \mathsf{S}')$ to $\mathcal{A}^*$. On prfx from $\mathcal{A}^*$, reject it if prfx was used before. Else record $\langle \mathsf{ssid}, \mathsf{P}, x, \mathsf{prfx}, 0 \rangle$ and send $(\mathsf{Prefix}, \mathsf{ssid}, \mathsf{prfx})$ to P.
- On $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{ssid}')$ from S, send $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{ssid}', \mathsf{S})$ to $\mathcal{A}^*$. On prfx' from $\mathcal{A}^*$ send $(\mathsf{Prefix}, \mathsf{ssid}', \mathsf{prfx}')$ to S. If there is a record $\langle \mathsf{ssid}, \mathsf{P}, x, \mathsf{prfx}, 0 \rangle$ s.t. $\mathsf{prfx} = \mathsf{prfx}'$, change it to $\langle \mathsf{ssid}, \mathsf{P}, x, \mathsf{prfx}, 1 \rangle$, else set $\mathsf{tx}{+}{+}$.
- On $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P}, i, \mathrm{L})$ from $\mathcal{A}^*$, retrieve $\langle \mathsf{ssid}, \mathsf{P}, x, \mathsf{prfx}, \mathsf{ok?} \rangle$ (ignore the message if there is no such record) and do:

```
  (1) If $i \notin \mathcal{N}$ then append $i$ to $\mathcal{N}$ and run $\textsc{Correlate}(i, L)$;
  (2) If $\mathsf{S}$ is not COMPROMISED and $\mathsf{ok?} = 0$ do:
        If $i = \mathsf{S}$ or $[(i, \mathsf{S}, x) \in \mathcal{E}$ and $\mathsf{P} = \mathcal{A}^*]$ do:
            If $\mathsf{tx} = 0$ then ignore this message, else set $\mathsf{tx}{-}{-}$;
  (3) Send $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, F_i(x))$ to $\mathsf{P}$.
```

$\textsc{Correlate}(i, L)$:

- Reject if list L contains elements $(j, x), (j', x')$ s.t. $j = j'$ and $x \neq x'$.
  Else, for all $(j, x) \in L$ s.t. $j \in \mathcal{N}$, add (i, j, x) to $\mathcal{E}$ and set $F_i(x) \leftarrow F_j(x)$.

Fig. 3. The Correlated OPRF functionality $\mathcal{F}_{\mathsf{corOPRF}}$. The (adaptive) OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ of [16] is formed by omitting all text in gray boxes.

OPRF instance. If S honestly engages in this protocol, which is modeled by call $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{ssid})$ from S, functionality $\mathcal{F}_{\mathsf{corOPRF}}$ increments the server-specific ticket-counter tx, initially set to 0. If the real-world adversary allows an uninterrupted interaction between C and S, which is modeled by a call $(\textsc{RcvComplete}, \mathsf{sid}, \mathsf{ssid}, \mathsf{C}, i, L)$ with $i = \mathsf{S}$ and $L = \perp$ from the ideal-world adversary $\mathcal{A}^*$, then $\mathcal{F}_{\mathsf{corOPRF}}$ decrements counter tx and sends $F_{\mathsf{S}}(x)$ to C.[10]

The man-in-the-middle adversary (our OPRF model does not rely on authenticated links) who interacts with client C, can make C output $F_i(x)$ for a different function $F_i \neq F_{\mathsf{S}}$, using a call $(\textsc{RcvComplete}, \mathsf{ssid}, \mathsf{C}, i, L)$ for $i \neq \mathsf{S}$, which models a real-world adversary acting like the server but on a wrong key $k_i \neq k$ in this interaction. To model a real-world adversary choosing different PRF keys in either offline or online evaluations, functionality $\mathcal{F}_{\mathsf{corOPRF}}$ keeps a list of indexes $\mathcal{N}$ of independent random functions, and effectively associates each real-world key with a distinct index in $\mathcal{N}$, whereas the key of the honest server S is associated with a special symbol S.

Practical implications. Note that $\textsc{RcvComplete}$ computes function F_{S} on P's input x only if $\mathsf{tx} > 0$, i.e. if the number of instances completed by S is greater than the number of instances completed by any client. This implies that if S engages in n OPRF instances this allows function F_{S} to be computed, by all other parties combined, on at most n arguments. However, the functionality does not establish strict binding between these server and client instances. Indeed, this *ticket-based* enforcement allows an OPRF functionality to be realized using homomorphic blinding without zero-knowledge proofs. Note that in protocol Exp-2HashDH of Fig. 1 the interaction between C and S can be "double blinded" by the network adversary, who can modify P's original message a as $a' = a^s$, and then modify S's response $b = a^k$ as $b' = b^{1/s}$. Such interaction produces the correct output on the client, but a' which S sees is a random group element, independent of a sent by C, which makes it impossible to identify the pair of C and S instances which the network adversary effectively pairs up.

Another feature which enables efficient $\mathcal{F}_{\mathsf{corOPRF}}$ realization is that the argument x of client C engaging in an OPRF instance can be defined only *after* server S completes this instance. Note that in the ideal world C outputs $F_{\mathsf{S}}(x)$ even if S completes an OPRF instance first, by sending message $(\textsc{SvrComplete}, \mathsf{sid}, \mathsf{ssid})$, and C only afterwards sends $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, x)$, followed by $\textsc{RcvComplete}$ from $\mathcal{A}^*$. Indeed, this "delayed input extraction" feature of $\mathcal{F}_{\mathsf{corOPRF}}$ enables protocol Exp-2HashDH to realize it in ROM, where the ideal-world adversary can extract argument x from the local computation of the real-world client, namely from H_2 query (x, v) for $v = (H_1(x))^k$, but that computation (and input-extraction) happens *after* S completes the protocol.

In some applications, notably OPAQUE [17], see Section 5, it is useful for OPRF to output a transcript, or its prefix, as a handle on OPRF instance in

[10] As in the adaptive version of UC OPRF $\mathcal{F}_{\mathsf{OPRF}}$ [17], we allow server S to be adaptively compromised, via call $\textsc{Compromise}$ from $\mathcal{A}^*$, which models a leakage of the private state of S, including its PRF key and all its authentication tokens. One consequence of server compromise is that $\textsc{RcvComplete}$ will no longer check that $\mathsf{tx} > 0$.

a higher-level protocol. Functionality $\mathcal{F}_{\mathsf{corOPRF}}$ allows each party to output a transcript prefix prfx, and if prfx output by S and C match then $\mathcal{F}_{\mathsf{corOPRF}}$ allows C session to compute the PRF output without using the tx counter. This does not affect the logic of tx-checking: Each run of SndrComplete either increments tx or ok's some particular client OPRF instance, so either way the number of on-line OPRF evaluations is limited by the number of SndrComplete instances.

Relaxation of the UC OPRF model. The crucial difference between the Correlated OPRF functionality $\mathcal{F}_{\mathsf{corOPRF}}$ and the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ of [11] is that when any party evaluates function F_i for a *new* index $i \notin \mathcal{N}$, which corresponds to a real-world adversary evaluating the (O)PRF either offline or online on a new key, the adversary can supply a list L of *correlations* which the new function F_i will have with previously initialized functions F_j, $j \in \mathcal{N}$, potentially including the honest server function F_{S}. Such correlations were not allowed in $\mathcal{F}_{\mathsf{OPRF}}$, and indeed $\mathcal{F}_{\mathsf{corOPRF}}$ reduces to $\mathcal{F}_{\mathsf{OPRF}}$ if $\mathcal{A}^*$ sets L as an empty list in OfflineEval and RcvComplete messages. Argument L can specify a sequence of pairs (j, x) where $j \in \mathcal{N}$ is an index of a previously initialized function F_j, and the correlation consists of setting the value of the new function F_i on x as $F_j(x)$. After setting $F_i(x) \leftarrow F_j(x)$ for all $(j, x) \in L$, the values of F_i on *all other arguments* are set at random by $\mathcal{F}_{\mathsf{corOPRF}}$. Functionality $\mathcal{F}_{\mathsf{corOPRF}}$ keeps track of these correlations in a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $(i, j, x) \in \mathcal{E}$ if $F_i(x)$ is set to $F_j(x)$ in the above manner, i.e., an edge between i and j, labeled x, represents a correlation between functions F_i and F_j on argument x.

A crucial constraint on the correlation list L is that for each $j \in \mathcal{N}$ list L can contain only one entry of the form $(j, \cdot)$, i.e. two functions F_i, F_j can be correlated on *at most one* argument.Note that if the adversary correlates F_i with the honest server function F_{S} on argument x, and then evaluates $F_i(x)$ via the online OPRF instance, i.e. Eval and RcvComplete where $\mathsf{P} = \mathcal{A}^*$, functionality $\mathcal{F}_{\mathsf{corOPRF}}$ treats this as an evaluation of F_{S} and decrements the ticket-counter tx. This restriction is necessary because otherwise the adversary could effectively compute F_{S} on more than n arguments even if an honest server S engages in only n OPRF instances: It could first correlate $n' > n$ adversarial functions $F_1, ..., F_{n'}$ with F_{S}, each function F_i on a different argument x_i, and each evaluation of $F_i(x_i)$ would reveal the value of F_{S} on all these arguments as well. However, our $\mathcal{F}_{\mathsf{corOPRF}}$ model allows $\mathcal{A}^*$ to let any *honest* party P compute $F_i(x)$ for F_i correlated with F_{S} without decrementing the ticket-counter tx. This is a weakness, e.g. if the higher-level application reveals these OPRF outputs to the attacker. A stronger version of $\mathcal{F}_{\mathsf{corOPRF}}$ would decrement tx even if $F_i(x) = F_{\mathsf{S}}(x)$ is computed by honest parties, but we used a weaker version for two reasons: First, it suffices for OPAQUE security. Second, we can show that Mult-2HashDH realizes this weaker version under Gap$^+$-OMDH, and it is an open problem whether the same can shown for the stronger version of the functionality.

Necessity of the relaxation. As noted in Section 1, Exp-2HashDH satisfies the UC OPRF notion of [11] because S's response b to C's message a defines key $k = \mathsf{DL}(a, b)$ s.t. C outputs $y = F_k(x)$ for function F_k defined in eq. (1). However, in Mult-2HashDH, S's response (b, z) defines the function which C effectively computes as $F_{(\delta,z)}$ defined in eq. (2). Moreover, different choices of (δ, z) do *not* define independent random functions. Indeed, an efficient attacker can easily pick (δ_1, z_1) and (δ_2, z_2) which satisfy equation (3) for any x, which implies that the two functions will be correlated by constraint $F_{(\delta_1,z_1)}(x) = F_{(\delta_2,z_2)}(x)$.

The consequences of such correlations can be illustrated by the following example. Assume that the higher-level application allows a malicious server to detect whether in two OPRF instances the client outputs the same two values or not. Let x_1 and x_2 be two client input candidates. If the server picks two indexes (δ_1, z_1) and (δ_2, z_2) s.t. $F_{(\delta_1,z_1)}(x_1) = F_{(\delta_2,z_2)}(x_1)$ and $F_{(\delta_1,z_1)}(x_2) \neq F_{(\delta_2,z_2)}(x_2)$ and inputs (δ_1, z_1) into the first OPRF instance and (δ_2, z_2) into the second one, then the client's outputs in these two executions will be the same if its input is x_1 and different if its input is x_2, and by the assumption on the application context the server will learn which one is the case. (In Section 7 we show examples of applications where this knowledge creates an attack avenue.)

The UC OPRF notion of [11] does not allow for this attack avenue because in that model each choice of a function index which server S can input into an OPRF instance defines an *independent* (pseudo)random function. However, no choice of two functions F_i, F_j for these two instances allows S to distinguish between C's input x_1 and x_2: If $F_i = F_j$ then C's output in the two instances will be the same for any x, and if $F_i \neq F_j$ then C's output in the two instances will be different, also for any x, except for a negligible probability that S finds two functions F_i, F_j among the polynomially-many random functions it can query offline s.t. $F_i(x) = F_j(x)$ for $x \in \{x_1, x_2\}$.

4 Security Analysis of Multiplicative DH-OPRF

Fig. 2 in Section 1 shows the OPRF protocol Mult-2HashDH, which uses multiplicative blinding for oblivious evaluation of the (Double) Hashed Diffie-Hellman function defined in eq. (1), i.e. $F_k(x) = H_2(x, (H_1(x))^k)$. Here, in Fig. 4, we render the same protocol as a realization of the Correlated OPRF functionality $\mathcal{F}_{\mathsf{corOPRF}}$ defined in Fig. 3. As we explain in Section 3, functionality $\mathcal{F}_{\mathsf{corOPRF}}$ reflects the correlations which a real-world adversary can introduce in the PRF functions the honest users compute in this protocol. Indeed, as we show in Theorem 1 below, under the *Gap One-More Diffie-Hellman* assumption protocol Mult-2HashDH securely realizes this functionality in ROM.

Theorem 1. *Protocol Mult-2HashDH realizes correlated OPRF functionality* $\mathcal{F}_{\mathsf{corOPRF}}$ *in the* $\mathcal{F}_{\mathsf{RO}}$*-hybrid world under the Gap-OMDH assumption.*

Proof: We show that for any efficient environment $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$ (more precisely, for $\mathcal{A}$ in the $\mathcal{F}_{\mathsf{RO}}$-hybrid world, i.e. the real world amended by random oracle hash functions), there exists an efficient simulator SIM, a.k.a. an "ideal-world adversary", s.t. the environment's view in the real world, where the honest parties implement the Mult-2HashDH protocol

Setting: – Group $\mathbb{G}$ of prime order q with generator g.
– Hash functions H_2, H_1 with ranges $\{0,1\}^\ell$ and $\mathbb{G}$, respectively.

Functions H_2, H_1 are specific to the OPRF instance initialized for a *unique* session id sid, and in practice they should be implemented by folding sid into their inputs.

Initialization: On input $(\textsc{Init}, \mathsf{sid})$, S picks $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and records $(\mathsf{sid}, k, z = g^k)$.

Server Compromise: On $(\textsc{Compromise}, \mathsf{sid}, \mathsf{S})$ from $\mathcal{A}$, reveal k to $\mathcal{A}$.

Offline Evaluation:
On $(\textsc{OfflineEval}, \mathsf{sid}, \mathsf{S}, x, \cdot)$, S outputs $(\textsc{OfflineEval}, \mathsf{sid}, F(k,x))$ where

$$F(k,x) \triangleq H_2(x, (H_1(x))^k))$$

Evaluation:

- On input $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, x)$, C picks $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, records $(\mathsf{sid}, \mathsf{ssid}, r)$, sends (ssid, a) to S for $a = H_1(x) \cdot g^r$, and locally outputs $(\mathsf{Prefix}, \mathsf{ssid}, a)$.
- On input $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{ssid}')$ and message (ssid, a) from C s.t. $a \in \mathbb{G}$, server S retrieves (sid, k, z), sends (ssid, b, z) to C for $b = a^k$, and locally outputs $(\mathsf{Prefix}, \mathsf{ssid}', a)$. (Note that ssid and ssid' can be different.)
- On S's message (ssid, b, z) from S s.t. $b, z \in \mathbb{G}$ and C holds tuple $(\mathsf{sid}, \mathsf{ssid}, r)$ for some r, party C outputs $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, y)$ for $y = H_2(x, b \cdot z^{-r})$.

Fig. 4. Protocol Mult-2HashDH of Fig. 2 as a realization of $\mathcal{F}_{\mathsf{corOPRF}}$.

interacting with adversary $\mathcal{A}$, is indistinguishable from its view in the ideal world, where the honest parties are "dummy" entities which pass their inputs to (and outputs from) the ideal functionality $\mathcal{F}_{\mathsf{corOPRF}}$, and where the real-world adversary $\mathcal{A}$ is replaced by the simulator SIM (who locally interacts with $\mathcal{A}$). The construction of SIM is shown in Fig. 5. While the real-world adversary $\mathcal{A}$ works in a hybrid world with the random oracle modeled by functionality $\mathcal{F}_{\mathsf{RO}}$, for notation simplicity in Fig. 5 we short-circuit the $\mathcal{F}_{\mathsf{RO}}$ syntax and we assume that SIM implements oracles H_1, H_2. Without loss of generality, we assume that $\mathcal{A}$ is a "dummy" adversary who merely passes all messages between $\mathcal{Z}$ and SIM, hence we will treat $\mathcal{A}$ as just an interface of $\mathcal{Z}$. For brevity we also denote $\mathcal{F}_{\mathsf{corOPRF}}$ as $\mathcal{F}$, and we omit the (fixed) session identifier sid from all messages. Also, the simulator assumes that a unique party S for which this $\mathcal{F}$ instance is initialized is honest, and that its identity "S" encoded as a bitstring is different from any pair $(\delta, z) \in \mathbb{G}^2$.

For a fixed environment $\mathcal{Z}$, let $q_{\mathrm{H}_1}, q_{\mathrm{H}_2}$ be the number of $\mathcal{A}$'s queries to resp. H_1 and H_2 hash functions, and let $q_{\mathsf{C}}, q_{\mathsf{S}}$ be the number of $\mathcal{Z}$'s invocations of resp. client and server OPRF instances, via resp. queries $\textsc{Eval}$ sent to some C and query $\textsc{SndrComplete}$ sent to S.

Initialization: Pick $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$ //SIM picks S's key//, set T_{H_1} as an empty table, set functions H_1, H_2 as undefined on all arguments, and set $\mathcal{N}_{\mathsf{SIM}} \leftarrow [\mathsf{S}]$ //$\mathcal{N}_{\mathsf{SIM}}$ is the list of identified function indices//.

Server Compromise: On $(\textsc{Compromise}, \mathsf{S})$ from $\mathcal{A}$, send $(\textsc{Compromise}, \mathsf{S})$ to $\mathcal{F}$ and reveal k to $\mathcal{A}$.

Hash query to H_1: On $\mathcal{A}$'s fresh query x to H_1, pick $u \leftarrow_{\mathrm{R}} \mathbb{Z}_q \setminus \{0\}$, define $h_x \triangleq g^u$, set $H_1(x) \leftarrow h_x$, and add (x, u, h_x) to table T_{H_1}.
//T_{H_1} records $h_x = H_1(x)$ and the discrete-logarithm trapdoor $u = \mathsf{DL}(g, h_x)$//

Online Evaluation:

1. On $(\textsc{Eval}, \mathsf{ssid}, \mathsf{C}, \mathsf{S}')$ from $\mathcal{F}$, pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, record $(\mathsf{C}, \mathsf{ssid}, w)$, send (ssid, a) for $a \leftarrow g^w$ to $\mathcal{A}$, and send $\mathsf{prfx} = a$ to $\mathcal{F}_{\mathsf{corOPRF}}$. (Abort if $\mathcal{F}_{\mathsf{corOPRF}}$ rejects it.)
2. On $(\textsc{SndrComplete}, \mathsf{ssid}', \mathsf{S})$ from $\mathcal{F}$ and message (ssid, a') from $\mathcal{A}$ s.t. $a' \in \mathbb{G}$, send ssid and $(b', z^*) = ((a')^k, g^k)$ to $\mathcal{A}$ and $\mathsf{prfx}' = a'$ to $\mathcal{F}_{\mathsf{corOPRF}}$.
3. On message (ssid, b, z) to C from $\mathcal{A}$ s.t. $b, z \in \mathbb{G}$, retrieve record $(\mathsf{C}, \mathsf{ssid}, w)$ (ignore the message if there is no such record) and do:
   //C should output $F_{(\delta,z)}(x)$ for $\delta = b/a^{\mathsf{DL}(g,z)} = b/z^w$//
   (1) Set $\delta \leftarrow b/z^w$, $i \leftarrow (\delta, z)$, $L \leftarrow []$;
   (2) If $i = (1, g^k)$ //$\mathcal{A}$ lets C evaluate on F_{S}// then (re)set $i \leftarrow \mathsf{S}$;
   (3) If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then for each $(x', u, h_{x'}) \in T_{H_1}$ and $(\delta', z') \in \mathcal{N}_{\mathsf{SIM}}$ do:
   - If $\delta' \cdot (z')^u = \delta \cdot z^u$ then add (j, x') for $j = (\delta', z')$ to L;
     //correlation on x' between F_i and F_j for $j = (\delta', z')$//
   - If $(h_{x'})^k = \delta \cdot z^u$ then add (S, x') to L; //correlation on x' with F_{S}//

   (4) Send $(\textsc{RcvComplete}, \mathsf{ssid}, \mathsf{C}, i, L)$ to $\mathcal{F}$, and append i to $\mathcal{N}_{\mathsf{SIM}}$ if $i \notin \mathcal{N}_{\mathsf{SIM}}$.

Hash query to H_2: On $\mathcal{A}$'s fresh query (x, v) to H_2, do:

1. If $(x, u, h_x) \in T_{H_1}$ and $v = (h_x)^k$ //$\mathcal{A}$ evaluates $F_{\mathsf{S}}(x)$// then do:
   - If S is compromised, send $(\textsc{OfflineEval}, \mathsf{S}, x, \perp)$ to $\mathcal{F}$; on $\mathcal{F}$'s response $(\textsc{OfflineEval}, y)$, set $H_2(x, v) \leftarrow y$;
   - Otherwise send $(\textsc{Eval}, \mathsf{ssid}, \mathsf{S}, x)$ and then $(\textsc{RcvComplete}, \mathsf{ssid}, \mathsf{SIM}, \mathsf{S}, \perp)$ to $\mathcal{F}$ for a fresh ssid; if $\mathcal{F}$ replies $(\textsc{Eval}, \mathsf{ssid}, y)$ then set $H_2(x, v) \leftarrow y$, otherwise output $\textsc{halt}$ and abort.
2. If $(x, u, h_x) \in T_{H_1}$ and $v \neq (h_x)^k$ then for the first $(\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$ s.t. $v = \delta \cdot z^u$ //$\mathcal{A}$ evaluates $F_{(\delta,z)}(x)$// send $(\textsc{OfflineEval}, i = (\delta, z), x, \perp)$ to $\mathcal{F}$; on $\mathcal{F}$'s response $(\textsc{OfflineEval}, y)$, set $H_2(x, v) \leftarrow y$.
3. If $H_2(x, v)$ remains undefined set $i = (v, 1)$ and: //$\mathcal{A}$ evaluates $F_{(v,1)}(x)$//
   (1) If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then for each $(x', u, h_{x'}) \in T_{H_1}$ and $(\delta', z') \in \mathcal{N}_{\mathsf{SIM}}$ do:
   - If $\delta' \cdot (z')^u = v$ then add (j, x') for $j = (\delta', z')$ to L;
   - If $(h_{x'})^k = v$ then add (S, x') to L;

   (2) Send $(\textsc{OfflineEval}, i, x, L)$ to $\mathcal{F}$; on $\mathcal{F}$'s response $(\textsc{OfflineEval}, y)$, set $H_2(x, v) \leftarrow y$, and append i to $\mathcal{N}_{\mathsf{SIM}}$ if $i \notin \mathcal{N}_{\mathsf{SIM}}$.

Fig. 5. Simulator SIM for Protocol Mult-2HashDH //with comments inline//

The simulator. The simulator SIM, shown in Fig. 5, follows a similar simulation strategy to the one used to show that exponential blinding protocol realizes UC OPRF notions of [10, 11, 17]. At initialization, the simulator picks a random key k on behalf of server S. If SIM receives $\textsc{SndrComplete}$ from $\mathcal{F}$, i.e. server S wants to complete an OPRF instance, and SIM receives message a with matching ssid from adversary $\mathcal{A}$ playing a client, SIM replies as the real-world S would, i.e. with $(b, z) = (a^k, g^k)$. Responding to $\mathcal{A}$ playing a server is more complex. The simulator prepares for this by embeding discrete-logarithm trapdoors in H_1 outputs and in messages a formed on behalf of honest clients. Namely, for each x, SIM defines $H_1(x)$ as $h_x = g^u$ for random u, and it forms each message a on behalf of some honest client as $a = g^w$ for random w. The discrete-logarithm trapdoor $u = \mathsf{DL}(g, a)$ enables SIM to compute, given response (b, z) sent by $\mathcal{A}$ on behalf of some server, the function index $i = (\delta, z)$ for which a real-life honest client would effectively compute its output as $y = F_{(\delta,z)}(x)$ for $F_{(\delta,z)}$ defined as in eq. (2). This is done by setting $\delta = b/z^w$ because then $\delta = b/a^k$ for $k = \mathsf{DL}(g, z)$. (See *Is multiplicative blinding secure?* in Section 1 for why the client effectively evaluates $F_{(\delta,z)}$ for $\delta = b/a^k$.) If $\mathcal{A}$ responds as the honest server S (or forwards S's response), SIM detects it because then $\delta = 1$, in which case SIM sets the function index to the "honest S function", $i \leftarrow \mathsf{S}$.

Finally, SIM checks if $i = (\delta, z)$ is in $\mathcal{N}_{\mathsf{SIM}}$, a sequence of function indices which SIM has previously identified, and if $i \notin \mathcal{N}_{\mathsf{SIM}}$, i.e. if it is a new function, SIM uses the trapdoors it embedded in H_1 outputs to detect if $F_i(x) = F_j(x)$ for any x queried to H_1 (without such query $\mathcal{A}$ cannot establish a correlation on x except for negligible probability) and any previously seen function index $j \in \mathcal{N}_{\mathsf{SIM}}$ or $j = \mathsf{S}$. The first condition holds if $\delta' \cdot (h_x)^{\mathsf{DL}(g,z')} = \delta \cdot (h_x)^{\mathsf{DL}(g,z)}$ for $i = (\delta, z)$ and $j = (\delta', z')$ while the second one holds if $(h_x)^k = \delta \cdot (h_x)^{\mathsf{DL}(g,z)}$. The simulator cannot compute $\mathsf{DL}(g, z)$ for an adversarial public key z, but the trapdoor in the hash function output $H_1(x) = h_x = g^u$ allows for computing $(h_x)^{\mathsf{DL}(g,z)}$ as z^u.

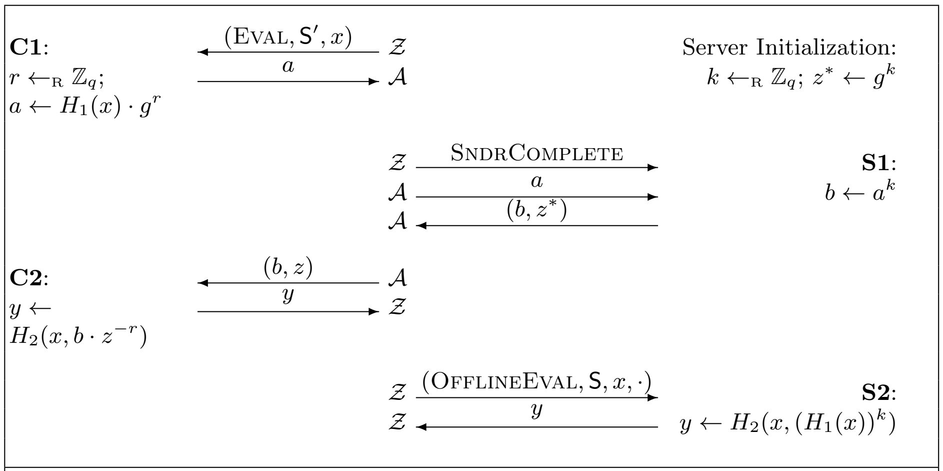
There is a further complication in the simulator's code, in responding to $\mathcal{A}$'s local H_2 queries (x, v). Such calls can represent either (I) an offline PRF evaluation on argument x of function $F_{(\delta,z)}$ s.t. $v = \delta \cdot (h_x)^{\mathsf{DL}(g,z)}$, where $(\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$, or, if S is compromised (or corrupted), for $(\delta, z) = (1, g^k)$; or (II) in case $v = (h_x)^k$ and S is not compromised, they can represent a finalization of the computation of $F_{\mathsf{S}}(x)$ by a malicious client in the online OPRF instance. Case (I) is treated similarly as the detection of the correlations explained above: SIM searches for index $i = (\delta, z)$ in $\mathcal{N}_{\mathsf{SIM}}$ s.t. $v = \delta \cdot (h_x)^{\mathsf{DL}(g,z)} = \delta \cdot z^u$ where $H_1(x) = h_x = g^u$, in which case this is interpreted as evaluation of F_i and SIM sets $H_2(x, v)$ to the value of $F_i(x)$ which the functionality defines in response to the offline evaluation call $(\textsc{OfflineEval}, i, x, \cdot)$. If S is compromised then the simulator does this also for $i = \mathsf{S}$ if $v = (h_x)^k$. However, in Case (II), i.e. if $v = (h_x)^k$ but S is not compromised, such query could come from $\mathcal{A}$'s post-processing of an online OPRF evaluation, hence SIM in this case sends $(\textsc{Eval}, \mathsf{ssid}, \mathsf{S}, x)$ and $(\textsc{RcvComplete}, \mathsf{ssid}, \mathsf{SIM}, \mathsf{S}, \perp)$ to $\mathcal{F}$. If $\mathcal{F}$ allows this call to evaluate successfully, i.e. if $\mathsf{tx} > 0$, then $\mathcal{F}$ return $y = F_{\mathsf{S}}(x)$ and SIM

defines $H_2(x,v) \leftarrow y$. Otherwise $\mathcal{F}$ will ignore this RcvComplete call, in which case SIM outputs halt and aborts, which the environment will detect as a simulation failure. Indeed, this case corresponds to $\mathcal{A}$ evaluating function F_{S} on more arguments than the number of OPRF instances performed by S, i.e. the number of SndrComplete calls from an ideal-world S to $\mathcal{F}$.

Finally, SIM must carefully handle $H_2(x,v)$ queries which are *not* recognized as evaluations of $F_i(x)$ for any $i \in \mathcal{N}_{\mathsf{SIM}} \cup \{\mathsf{S}\}$, because they can correspond to evaluating $F_{(\delta,z)}(x)$ for index (δ,z) which $\mathcal{A}$ will reveal in the future. SIM picks the simplest pair (δ,z) s.t. $\delta \cdot (h_x)^{\mathsf{DL}(g,z)} = v$, namely $(\delta,z) = (v,1)$. If any future index $(\delta,z) \neq (v,1)$ defined in a subsequent OPRF evaluation satisfies $\delta \cdot (h_x)^{\mathsf{DL}(g,z)} = v$, this will be detected by SIM as a correlation between $F_{(\delta,z)}$ and $F_{(v,1)}$. Note that SIM must process $H_2(x,v)$ query as evaluation of $F_{(v,1)}(x)$ even if $H_1(x)$ is undefined, because regardless of the value of $h_x = H_1(x)$ it will hold that $F_{(v,1)}(x) = H_2(x,v)$, because $v \cdot (h_x)^{\mathsf{DL}(g,1)} = v \cdot (h_x)^0 = v$. Indeed, an adversary can first query $H_2(x,v)$ for some (x,v), then compute $h_x = H_1(x)$, and then input (δ,z) into an OPRF instance for $\delta = v/(h_x)^{\mathsf{DL}(g,z)}$, which corresponds to oblivious evaluation of $F_{(\delta,z)}$, which is correlated with $F_{(v,1)}$ on argument x.

Sequence of games. Our proof uses the standard sequence of games method, starting from the interaction of $\mathcal{Z}$ (and "dummy" adversary $\mathcal{A}$) with the real-world protocol, and ending with the ideal world, in which $\mathcal{Z}$ instead interacts with the simulator SIM and functionality $\mathcal{F}$. We fix an arbitrary efficient environment $\mathcal{Z}$ which without loss of generality outputs a single bit, we use $\mathbf{G_i}$ to denote the event that $\mathcal{Z}$ outputs 1 when interacting with Game i, and for each two adjacent games, Game i and Game $i+1$, we argue that these games are indistinguishable to $\mathcal{Z}$, i.e. that there is a negligible difference between the probabilities of events $\mathbf{G_i}$ and $\mathbf{G_{i+1}}$, which implies that $\mathcal{Z}$'s advantage in distinguishing between the real world and the ideal world is also negligible. Let $q_{\mathrm{H}_1}, q_{\mathrm{H}_2}$ be the total number of resp. H_1, H_2 queries made in the security game with $\mathcal{A}$ and $\mathcal{Z}$. Let q_{C} and q_{S} and q'_{S} be the number of resp. C and S sessions and S offline PRF evaluations started by $\mathcal{Z}$ via resp. the Eval, SndrComplete, and (OfflineEval, $\mathsf{S}, \cdot, \cdot$) commands. Let $\epsilon_{\mathrm{OMDH}}(\mathbb{G}, N, Q)$ be the maximum advantage of any algorithm with computational resources comparable to $\mathcal{Z}$ against the (N,Q)-Gap$^+$-OMDH problem in $\mathbb{G}$.

Game 1: (*Real world, except for discrete-logarithm trapdoors in H_1 outputs*) This is the real-world interaction, shown in Fig. 6, i.e. the interaction of environment $\mathcal{Z}$ and its subroutine $\mathcal{A}$ with honest entities C and S executing protocol Mult-2HashDH of Fig. 4. We assume that the interaction starts with server initialization, triggered by Init command from $\mathcal{Z}$ to S. We denote the public key of server S as $z^* = g^k$. For visual clarity we omit the fixed sid tag and the variable ssid tags from all messages in Fig. 6. We assume that when functions H_1, H_2 are executed by C1, C2, and S2, these hash function calls are serviced as described in the lower-half of Fig. 6. Queries $H_2(x,v)$ are implemented as in the real world except that the game records tuples $(x, v, H_2(x,v))$ in table T_{H_2}. However, queries $H_1(x)$ are implemented with trapdoors embedded in values $h_x = H_1(x)$ by setting $h_x = g^{u_x}$ for random $u_x \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and recording (x, u_x, h_x) in table T_{H_1}.


C1:
$r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$;
$a \leftarrow H_1(x) \cdot g^r$
$(\textsc{Eval}, \mathsf{S}', x)$
a
$\mathcal{Z}$
$\mathcal{A}$
Server Initialization:
$k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$; $z^* \leftarrow g^k$
SndrComplete
a
(b, z^*)
S1:
$b \leftarrow a^k$
C2:
$y \leftarrow$
$H_2(x, b \cdot z^{-r})$
(b, z)
y
$(\textsc{OfflineEval}, \mathsf{S}, x, \cdot)$
y
S2:
$y \leftarrow H_2(x, (H_1(x))^k)$

H₁: On query x, pick $u_x \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $h_x \leftarrow g^{u_x}$ and $H_1(x) \leftarrow h_x$, add (x, u_x, h_x) to T_{H_1};
H₂: On query (x, v), pick $y \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$, set $H_2(x, v) \leftarrow y$, add (x, v, y) to T_{H_2};
S-compromise: On $\mathcal{A}$'s message Compromise, send k to $\mathcal{A}$.

Fig. 6. Game 1: Interaction of $\mathcal{Z}/\mathcal{A}$ with Mult-2HashDH protocol.

Game 2: (*Abort on hash H_1 collisions*) Abort if the security game ever encounters a collision in H_1, i.e. if for some argument x queried either by $\mathcal{A}$ or by the security game in oracles C1 and S2 (see Fig. 6), oracle H_1 picks u s.t. tuple (x', u, g^u) for some $x' \neq x$ is already in T_{H_1}. Clearly

$$|\Pr[\mathbf{G_2}] - \Pr[\mathbf{G_1}]| \leq \frac{(q_{\mathrm{H}_1})^2}{q}$$

Game 3: (*Making* C*'s message input-oblivious*) We change how oracle C1 generates message a so that it is generated obliviously of input x. Namely, instead of computing $a = H_1(x) \cdot g^r = g^{u_x + r}$ for $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, oracle C1 will now generate $a = g^w$ for $w \leftarrow_{\mathrm{R}} \mathbb{Z}_q$. The input x for this session ssid will be then passed to oracle C2, which (1) queries H_1 on x to retrieve (or create) tuple (x, u_x, g^{u_x}) from T_{H_1}, and (2) outputs $y = H_2(x, v)$ for $v = b \cdot z^{u_x - w}$. Note that for every x, and hence every u_x, value $w = (u_x + r) \bmod q$ is random in $\mathbb{Z}_q$ if r random in $\mathbb{Z}_q$, hence this modification does not change the distribution of values a output by C1. Moreover, if $w = (u_x + r) \bmod q$ then $z^{-r} = z^{u_x - w}$, thus C2's output is the same as in Game 2, hence Game 3 and Game 2 are externally identical.

Game 4: (*Defining adversarial functions*) We make a notational change in oracle C2, so that it outputs $y = H_2(x, v)$ for $v = \delta \cdot z^{u_x}$ where $\delta = b/z^w$. Since this is a merely notational difference, Game 4 and Game 3 are identical.

Note that this change makes oracles C1/C2 implement the following process: C1's message $a = g^w$ together with $\mathcal{A}$'s response (b, z) define (δ, z) s.t. $\delta = b/z^w$, which defines a function which C2 evaluates on $\mathcal{Z}$'s input x as $F_{(\delta,z)}$ for

$$F_{(\delta,z)}(x) \triangleq H_2(x, \delta \cdot z^{u_x}) \quad \text{where} \quad u_x \triangleq \mathsf{DL}(g, H_1(x)) \tag{5}$$

Note that equation (5) is equivalent to equation (2) where $F_{(\delta,z)}(x) = H_2(x, \delta \cdot (H_1(x))^k)$ for k s.t. $z = g^k$. For notational convenience we define also a "helper" function family $f_i : \{0,1\}^* \to \mathbb{G}$ for $i \in \mathbb{G}^2$ s.t.

$$f_{(\delta,z)}(x) = \delta \cdot z^{u_x} \quad \text{where} \quad u_x \triangleq \mathsf{DL}(g, H_1(x)) \tag{6}$$

Note that $F_{(\delta,z)}(x) = H_2(x, f_{(\delta,z)}(x))$.

We will argue that pairs (δ, z) encountered in the security game can be thought of as indexes of random functions, including pair $(\delta, z) = (1, z^*)$ for $z^* = g^k$ which defines the "honest" random function of S, except that the adversary can "program" a limited number of correlations in these functions, by setting $i = (\delta, g^k)$ and $j = (\delta', g^{k'})$ s.t. $\delta'/\delta = (h_x)^{k-k'}$, which implies that $F_i(x) = F_j(x)$. In the next few game changes we will show that these correlations are constrained as prescribed by functionality $\mathcal{F}_{\mathsf{corOPRF}}$, i.e. that (1) each two functions can be "programmed" to have equal output only for a single argument, (2) that if an adversarial function F_i is correlated on some x with function F_{S} of the honest server S then evaluating $F_i(x)$ is treated the same as $F_{\mathsf{S}}(x)$, and in particular requires that $\mathsf{tx} > 0$, and (3) that otherwise all adversarial functions are indistinguishable from independent random functions.

Game 5: (*Building correlation graph*) The security game will build a graph of correlations between functions $F_{(\delta,z)}$ occurring in the game. In particular the game will maintain sequence $\mathcal{N}_{\mathsf{SIM}}$ and sets $X_{H_1}, \mathcal{E}$, all initially empty:

1. Set X_{H_1} contains all inputs x queried to H_1, by either $\mathcal{A}$, C2, or S2.
2. Set $\mathcal{N}_{\mathsf{SIM}}$ contains all (δ, z) function indexes, including (1) the honest server function index $(1, z^*)$, (2) each (δ, z) defined by $\mathcal{A}$'s interaction with oracles C1/C2, as described in Game 4, and (3) $(\delta, z) = (v, 1)$ for every direct query (x, v) of $\mathcal{A}$ to H_2.
3. Set $\mathcal{E}$ contains *labeled edges* between indexes in $\mathcal{N}_{\mathsf{SIM}}$, maintained as follows:

   (1) When function index $i = (\beta, z) \notin \mathcal{N}_{\mathsf{SIM}}$ is specified in C1/C2 then for each $j = (\delta', z')$ in $\mathcal{N}_{\mathsf{SIM}}$ and $x' \in X_{H_1}$, test if $f_j(x') = f_i(x')$, and if so then add (i, j, x'), i.e. an edge (i, j) with label x', to $\mathcal{E}$.

   (2) If H_2 is queried on new (x, v) by $\mathcal{A}$ or by oracles C2 or S2 for $(v, 1) \notin \mathcal{N}_{\mathsf{SIM}}$ then do step (1) above for $i = (v, 1)$. (Note that $f_{(v,1)}(x') = v$ for all x'.)

Since these are only notational changes Game 5 and Game 4 are identical.

Game 6: (*Discarding double links*) We add an abort if there are two distinct values x, x' in X_{H_1} and two distinct function indexes $i = (\delta, z)$ and $j = (\delta', z')$ in

$\mathcal{N}_{\mathsf{SIM}}$ s.t. $f_i(x) = f_j(x)$ and $f_i(x') = f_j(x')$. These conditions imply respectively that $\delta'/\delta = (z/z')^{u_x}$ and $\delta'/\delta = (z/z')^{u_{x'}}$. Since H_1 collisions are discarded beginning in Game 2, it follows that $u_{x'} \neq u_x$, which implies that $(\delta, z) = (\delta', z')$, i.e. this abort cannot happen. Consequently, Game 6 and Game 5 are identical.

Game 7: (*Discarding future correlations*) We add an abort in H_1 processing if new query $x \notin X_{H_1}$ samples $h_x = H_1(x)$ s.t. there exists two distinct function indexes $i, j \in \mathcal{N}_{\mathsf{SIM}}$ s.t. $f_i(x) = f_j(x)$. Note that in this case there would be no edge (i, j, x) in $\mathcal{E}$, and that this is *the only* case in which $f_i(x) = f_j(x)$ but $(i, j, x) \notin \mathcal{E}$. However if query x to H_1 is made after defining i, j then $h_x = H_1(x)$ is independent of i, j, in which case $\Pr[f_i(x) = f_j(x)] = 1/q$, because this equation holds only for a single value h_x s.t. $u_x = \mathsf{DL}(g, h_x) = \mathsf{DL}((z_i/z_j), (\delta_j/\delta_i))$. If there are q_{C} instances of C2 and q_{H_2} queries to H_2 then there can be at most q_{C} indexes (δ, z) in $\mathcal{N}_{\mathsf{SIM}}$ s.t. $z \neq 1$ and at most q_{H_2} indexes (δ, z) s.t. $z = 1$. Since condition $f_i(x) = f_j(x)$ cannot be met if $i = (v, 1)$ and $j = (v', 1)$ for $v \neq v'$, each new query x to H_1 causes an abort only if u_x falls in the solution set of at most $q_{\mathsf{C}} \cdot (q_{\mathrm{H}_2} + q_{\mathsf{C}})$ equations, which implies that

$$|\Pr[\mathbf{G_7}] - \Pr[\mathbf{G_6}]| \leq \frac{q_{\mathrm{H}_1} \cdot q_{\mathsf{C}} \cdot (q_{\mathrm{H}_2} + q_{\mathsf{C}})}{q}$$

Game 8: (*Implementing H_2 using correlated random functions*) We replace hash function H_2 using an oracle $\mathcal{R}$ that maintains a random function family, in which the adversary can "program" correlations as follows:

– When $\mathcal{R}$ starts it initializes a random function $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^\ell$ and an index sequence $\mathcal{I} \leftarrow [(1, z^*)]$;
– On query Correlate(i, L), $\mathcal{R}$ rejects if $i \notin \mathcal{I}$ or list L contains (j, x) and (j', x') s.t. $j = j'$ and $x \neq x'$. Otherwise it appends i to $\mathcal{I}$, and for each $(j, x) \in L$ it re-defines $R(i, x) \leftarrow R(j, x)$;
– On query Eval(i, x), $\mathcal{R}$ replies $R(i, x)$ if $i \in \mathcal{I}$, else ignores this query.

We use oracle $\mathcal{R}$ to change the implementation of H_2 function called by oracles S2, C2, or the direct calls to H_2:

1. When $\mathcal{A}$ calls S2 on x: Assign $H_2(x, f_i(x)) \leftarrow \mathcal{R}.$Eval$(i, x)$ for $i = (1, z^*)$.
2. When oracle C2 calls H_2 on $(x, f_i(x))$ for some $i = (\delta, z)$:
   (a) if $i \notin \mathcal{N}_{\mathsf{SIM}}$ then send Correlate(i, L) to $\mathcal{R}$ where L consists of all tuples (j, x') s.t. $f_i(x') = f_j(x')$ for some $j \in \mathcal{N}_{\mathsf{SIM}}$ and $x' \in X_{H_1}$;
   (b) set $H_2(x, f_i(x)) \leftarrow \mathcal{R}.$Eval$(i, x)$.
3. When $\mathcal{A}$ calls H_2 on (x, v): Service it as in Step 2 but use $i = (v, 1)$.

To see the correspondence between Game 8 and Game 7, observe that starting from Game 5 function H_2 is evaluated only on pairs of the form $(x, f_i(x))$ for some $i \in \mathcal{N}_{\mathsf{SIM}}$. Define $\mathcal{R}(i, x)$ as $H_2(x, f_i(x))$. Function $\mathcal{R}$ is not random even if H_2 is, because we have that $\mathcal{R}(i, x) = \mathcal{R}(j, x)$ for any i, j, x s.t. $f_i(x) = f_j(x)$. However, from Game 7 this equation can hold, for any i, x s.t. H_2 is queried on

$(x, f_i(x))$, only if i is a new index, $i = (\delta, z)$ or $i = (v, 1)$, appended to $\mathcal{N}_{\mathsf{SIM}}$ in a query to oracles resp. C1/C2 and H_2, for values j, x s.t. $j \in \mathcal{N}_{\mathsf{SIM}}$ and $x \in X_{H_1}$ at the time this query is made. Note that list L sent for a new function f_i to $\mathcal{R}$ in GAME 8 by oracles C1/C2 and H_2 consists exactly of all such pairs (j, x), hence it follows that GAME 8 and GAME 7 are identical.

GAME 9: (*Walking back aborts in* H_1) We remove the aborts in H_1 introduced in GAME 2 and GAME 7, i.e. we no longer abort if (1) the same u_X was chosen before on some previous query to H_1, or (2) if there are two function indices $i = (z, \delta)$ and $j = (z', \delta')$ in $\mathcal{N}_{\mathsf{SIM}}$ s.t. $f_i(x) = f_j(x)$, i.e. $\delta \cdot z^{u_x} = \delta' \cdot (z')^{u_x}$. By the same arguments used above where these games are introduced, these two changes can be observed with probability at most $(q_{\mathrm{H}_1}^2)/q$ and $(q_{\mathrm{H}_1} \cdot q_{\mathsf{C}} \cdot (q_{\mathrm{H}_2} + q_{\mathsf{C}}))/q$, respectively, which implies that

$$|\Pr[\mathbf{G_9}] - \Pr[\mathbf{G_8}]| \leq \frac{q_{\mathrm{H}_1}^2 + q_{\mathrm{H}_1} \cdot q_{\mathsf{C}} \cdot (q_{\mathrm{H}_2} + q_{\mathsf{C}})}{q}$$

Security game review. In Fig. 7 we put together all the changes made so far and review how the game oracles operate in GAME 9.

- **Init:** Initialize RF R, pick $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $\mathcal{N}_{\mathsf{SIM}} \leftarrow [(1, g^k)]$ and $X_{H_1} \leftarrow \{\}$.
- **H₁**: On input $x \notin X_{H_1}$, pick $u_x \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, add x to X_{H_1}, output g^{u_x} to $\mathcal{A}$.
- **S1:** On input $a \in \mathbb{G}$, send $(b, z) = (a^k, g^k)$ to $\mathcal{A}$.
- **S2:** On input x, set $i \leftarrow (1, g^k)$ and send $R(i, x)$ to $\mathcal{A}$.
- **C1:** On input x, pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, store (x, w), send $a = g^w$ to $\mathcal{A}$.
- **C2:** On input $(b, z) \in \mathbb{G}^2$, recover (x, w) stored by C1 and set $\delta \leftarrow b/z^w$. Assign $i \leftarrow (\delta, z)$. If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then run PROCESS(i). Send $R(i, x)$ to $\mathcal{A}$.
- **H₂**: On new input (x, v) from $\mathcal{A}$ for $v \in \mathbb{G}$, set $i \leftarrow (v, 1)$. If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then run PROCESS(i). Send $R(i, x)$ to $\mathcal{A}$.
- **S-Compromise:** On message (COMPROMISE, S) from $\mathcal{A}$, send k to $\mathcal{A}$.
- Subprocedure PROCESS(i): Parse i as $(\delta, z) \leftarrow i$. Define list L s.t.

$$L = \{\, (j, x) \in \mathcal{N}_{\mathsf{SIM}} \times X_{H_1} \;\text{ s.t. }\; j = (\delta', z') \text{ and } \delta \cdot z^{u_x} = \delta' \cdot (z')^{u_x} \,\}$$

  Abort it L contains $(j, x), (j, x')$ s.t. $x \neq x'$.
  Otherwise append i to $\mathcal{N}_{\mathsf{SIM}}$, and for each (j, x) in L, reset $R(i, x) \leftarrow R(j, x)$.

Fig. 7. Interaction defined by GAME 9.

GAME 10: (*Identifying existing functions in* H_2 *processing*) In GAME 9 a fresh query (x, v) to H_2 is answered as $R(i, x)$ for $i = (v, 1)$, and if $(v, 1) \notin \mathcal{N}_{\mathsf{SIM}}$ then function $R((v, 1), \cdot)$ is created and correlated with all previous functions $\{R(i, \cdot)\}_{i \in \mathcal{N}_{\mathsf{SIM}}}$ by the rule that $R((v, 1), x') \leftarrow R(i, x')$ for each $x' \in X_{H_1}$ and $i \in \mathcal{N}_{\mathsf{SIM}}$ s.t. $f_i(x') = v$. In GAME 10 we modify the code of oracle H_2 so that when it gets a fresh query (x, v) s.t. $x \in X_{H_1}$ it first checks if

$$v = f_i(x) \quad \text{for any index} \quad i \in \mathcal{N}_{\mathsf{SIM}} \tag{7}$$

(Note that if $x \in X_{H_1}$ the game can evaluate $f_{(\delta,z)}(x) = \delta \cdot z^{u_x}$ for any δ, z.) If $v = f_i(x)$ for some $i \in \mathcal{N}_{\mathsf{SIM}}$ then GAME 10 takes *the first* index i in $\mathcal{N}_{\mathsf{SIM}}$ s.t. $v = f_i(x)$ holds, replies $R(i, x)$, and does not create a new function $R((v, 1), \cdot)$ even if $(v, 1) \notin \mathcal{N}_{\mathsf{SIM}}$. (Note that this condition can hold for several indexes i in $\mathcal{N}_{\mathsf{SIM}}$, and indeed it will hold for all indexes of functions which are correlated on argument x. Note also that the index $i = (1, z^*)$ of the "honest server function" occurs as the first in $\mathcal{N}_{\mathsf{SIM}}$.) If $x \notin X_{H_1}$ or for all $i \in \mathcal{N}_{\mathsf{SIM}}$ $v \neq f_i(x)$ then the processing is as before, i.e. the game processes this query as a call to $R((v, 1), x)$. We show the modification done by GAME 10 in Figure 8.

```
– H₂: On new input $(x, v)$ from $\mathcal{A}$ for $v \in \mathbb{G}$:
    1. If $x \in X_{H_1}$ and $v = (g^k)^{u_x}$: Set $i \leftarrow (1, g^k)$, send $R(i, x)$ to $\mathcal{A}$
    2. If $x \in X_{H_1}$ and $v \neq (g^k)^{u_x}$, but $\exists\ (\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$ s.t. $v = \delta \cdot z^{u_x}$ then set $i$
       to the first $(\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$ for which it holds and send $R(i, x)$ to $\mathcal{A}$
    3. Else set $i \leftarrow (v, 1)$. If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then run PROCESS$(i)$. Send $R(i, x)$ to $\mathcal{A}$.
```

Fig. 8. GAME 10: modification in Fig. 7

Note that this modification doesn't change the value returned by $H_2(x, v)$: If condition (7) holds then either way $H_2(x, v) = R(i, x)$. The only other change this modification causes is that if (7) holds then function $R((v, 1), \cdot)$ is not created. However, this does not affect any future interactions with the random function R. Let X_{H_1} and $\mathcal{N}_{\mathsf{SIM}}$ are the values of these variables at the time $R((v, 1), \cdot)$ is created in GAME 9. Consider that at some subsequent step an evaluation call, either C2 or H_2, creates a new function $R(i, \cdot)$ s.t. $f_i(x) = f_{(v,1)}(x)$ for some $x \in X'_{H_1}$ where X'_{H_1} and $\mathcal{N}'_{\mathsf{SIM}}$ denote the new values of these variables. Assume also that until this point there was no other opportunity to create $\mathcal{R}((v, 1), \cdot)$ in GAME 10, i.e. $i = (v, 1)$ was not used in oracle C2, and $H_2(x', v)$ was not queried on any x' s.t. $f_i(x') \neq v$ for some $i \in \mathcal{N}'_{\mathsf{SIM}}$. (This is the case when the modification of GAME 10 can affect the security experiment.) There are two cases to consider: (1) If $x \in X_{H_1}$ and $f_{(v,1)}(x) = f_j(x)$ for some $j \in \mathcal{N}_{\mathsf{SIM}}$, then whether or not $R((v, 1), \cdot)$ is created in both games it holds that $R(i, x) = R(j, c)$; (2) If $x \notin X_{H_1}$, or $x \in X_{H_1}$ but $f_{(v,1)}(x) \neq f_j(x)$ for any $j \in \mathcal{N}_{\mathsf{SIM}}$, then $R((v, 1), x)$ is uncorrelated with previous functions, but since $R((v, 1), x)$ is not used before, it does not matter if $R(i, x)$ is chosen at random or assigned as $R(i, x) \leftarrow R((v, 1), x)$. It follows that GAME 10 and GAME 9 are identical.

GAME 11: (*Ideal-world interaction*) In Figure 9 we show the ideal-world game, denoted GAME 11, defined by the interaction of simulator SIM of Figure 5 and functionality $\mathcal{F}_{\mathsf{corOPRF}}$ of Figure 3. We use the same notation used for

GAME 9 for the correlated random functions, i.e. we define $F_{\mathsf{S}}(x) = R((1, z^*), x)$ and for all $i \neq \mathsf{S}$ we define $F_i(x) = R(i, x)$. Also, we rename oracles which the game implements as in GAME 9: S1 implements $\mathcal{Z}$'s query SNDRCOMPLETE to S, S2 implements $\mathcal{Z}$'s query OFFLINEEVAL to S, C1 implements $\mathcal{Z}$'s query EVAL to C, and C2 responds to $\mathcal{A}$'s message (b, z) to C.

```
– Init: Initialize RF $R$, $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$, $\mathcal{N}_{\mathsf{SIM}} \leftarrow [(1, g^k)]$, $X_{H_1} \leftarrow \{\}$, $\mathsf{tx} \leftarrow 0$.
– H_1: On input $x \notin X_{H_1}$, pick $u_x \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, add $x$ to $X_{H_1}$, output $g^{u_x}$ to $\mathcal{A}$.
– S1: On input $a \in \mathbb{G}$, send $(b, z) = (a^k, g^k)$ to $\mathcal{A}$.
  If $\exists$ record $(x, w, a, 0)$ change it to $(x, w, a, 1)$, else $\mathsf{tx}$++.
– S2: On input $x$, set $i \leftarrow (1, g^k)$ and send $R(i, x)$ to $\mathcal{A}$.
– C1: On input $x$, pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, store $(x, w, a, 0)$, send $a = g^w$ to $\mathcal{A}$.
– C2: On input $(b, z) \in \mathbb{G}^2$, recover $(x, w, a, \mathsf{ok?})$ stored by C1 and set $\delta \leftarrow b/z^w$.
  Assign $i \leftarrow (\delta, z)$. If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then run PROCESS($i$). Send $R(i, x)$ to $\mathcal{A}$.
  If $\mathsf{S}$ not compromised, $\mathsf{ok?} = 0$, and $i = (1, g^k)$ then do:
     If $\mathsf{tx} = 0$ then *abort the game*, else set $\mathsf{tx}$−−
– H_2: On new input $(x, v)$ from $\mathcal{A}$ for $v \in \mathbb{G}$:
  1. If $x \in X_{H_1}$ and $v = (g^k)^{u_x}$: Set $i \leftarrow (1, g^k)$, send $R(i, x)$ to $\mathcal{A}$, and do:
        If $\mathsf{S}$ not compromised and $\mathsf{tx} = 0$ then *abort the game*
        If $\mathsf{S}$ not compromised and $\mathsf{tx} > 0$ then set $\mathsf{tx}$−−
  2. If $x \in X_{H_1}$ and $v \neq (g^k)^{u_x}$, but $\exists$ $(\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$ s.t. $v = \delta \cdot z^{u_x}$ then set $i$
     to the *first* $(\delta, z) \in \mathcal{N}_{\mathsf{SIM}}$ for which it holds and send $R(i, x)$ to $\mathcal{A}$
  3. Else set $i \leftarrow (v, 1)$. If $i \notin \mathcal{N}_{\mathsf{SIM}}$ then run PROCESS($i$). Send $R(i, x)$ to $\mathcal{A}$.
– S-Compromise: On message (COMPROMISE, $\mathsf{S}$) from $\mathcal{A}$, send $k$ to $\mathcal{A}$.

– Subprocedure PROCESS($i$): Parse $i$ as $(\delta, z) \leftarrow i$. Define list $L$ s.t.

     $L = \{ (j, x) \in \mathcal{N}_{\mathsf{SIM}} \times X_{H_1}$ s.t. $j = (\delta', z')$ and $\delta \cdot z^{u_x} = \delta' \cdot (z')^{u_x} \}$

  Abort it $L$ contains $(j, x), (j, x')$ s.t. $x \neq x'$.
  Otherwise append $i$ to $\mathcal{N}_{\mathsf{SIM}}$, and for each $(j, x)$ in $L$, reset $R(i, x) \leftarrow R(j, x)$.
```

Fig. 9. GAME 11: Interaction of $\mathcal{Z}/\mathcal{A}$ with the ideal-world execution

Figure 9 simplifies the ideal-world game by not accounting for function correlations using edge set $\mathcal{E}$, as done by $\mathcal{F}_{\mathsf{corOPRF}}$, and ignoring some of the conditional clauses in the code of simulator SIM. However, we argue that these overlooked clauses are never triggered. Assume that whenever sub-procedure PROCESS(i) programs a correlation $R(i, x) \leftarrow R(j, x)$ the game adds set (i, j, x) to $\mathcal{E}$. The conditional clauses missing from GAME 11 figure are in clauses (2) and (3) in H_2 processing. In clause (2), SIM ignores this call, and the game does not send $R(i, x)$ to $\mathcal{A}$, if S was not compromised and either $i = (1, g^k)$ or $(i, (1, g^k), x) \in \mathcal{E}$. However, condition $i = (1, g^k)$ implies that $v = (g^k)^{u_x}$, which is excluded by case (2). Likewise, condition $(i, (1, g^k), x) \in \mathcal{E}$ implies that $f_i(x) = f_{(1,g^k)}(x) = (g^k)^{u_x}$, which would trigger case (1) and is excluded in case (2). In clause (3) SIM would ignore this call and not send $R(i, x)$ to $\mathcal{A}$

under the same conditions, i.e. if S was not compromised and either $i = (1, g^k)$ or $(i, (1, g^k), x) \in \mathcal{E}$. Case $i = (v, 1) = (1, g^k)$ implies $k = 0$, which is excluded by sampling k in $\mathbb{Z}_q^* = \mathbb{Z}_q \setminus \{0\}$, and case $(i, (1, g^k), x) \in \mathcal{E}$ implies that $x \in X_{H_1}$ and $f_i(x) = f_{(1,g^k)}(x)$, which would trigger clause (1).

Finally, in Figure 9 in two clauses when $\mathsf{tx} = 0$, in C2 and H_2 case (1), we wrote that the game *aborts*. In the actual ideal-world game, the first case corresponds to functionality $\mathcal{F}_{\mathsf{corOPRF}}$ dropping the $(\textsc{RcvComplete}, ..., \mathsf{C}, ...)$ call from SIM, and not sending $R(i, x)$ to C, and thus to $\mathcal{Z}$. The second case corresponds to $\mathcal{F}_{\mathsf{corOPRF}}$ not responding with $R(i, x)$ to SIM's call $(\textsc{RcvComplete}, ..., \mathsf{SIM}, ...)$, in which case SIM aborts. The difference is in the first case, but it is a syntactical difference because we can equate $\mathcal{Z}$'s not receiving any output from C in response to $(\textsc{RcvComplete}, ..., \mathsf{C}, ...)$, or any output from H_2 call, with the game returning an abort symbol.

The differences between Game 10 and Game 11, apart of the trivial difference of constraining key k s.t. $k \neq 0$ in Game 11, consist of the following:

1. S1 either increments tx or changes $\mathsf{ok?}$ in some C1-record from 0 to 1.
2. C2 decrements tx if S not compromised, $\mathsf{ok?} = 0$, $i = (1, g^k)$, and $\mathsf{tx} > 0$.
3. C2 aborts the game if S not compromised, $\mathsf{ok?} = 0$, $i = (1, g^k)$, and $\mathsf{tx} = 0$.
4. H_2, clause 1, decrements tx if S not compromised, $i = (1, g^k)$, and $\mathsf{tx} > 0$
5. H_2, clause 1, aborts the game if S not compromised, $i = (1, g^k)$, and $\mathsf{tx} = 0$.

Let E be the event that game aborts either in C2 or H_2, denoted resp. E_{C2} and E_{H_2}. Note that unless event E happens Game 10 and Game 11 are identical (except for $1/q$ probability that $k = 0$ in Game 10), and that event E can happen only if S is not compromised, thus the two games diverge only before S compromise. Note that E_{C2} requires that $i = (1, g^k)$, i.e. that $\mathcal{A}$ sends (b, z) to C2 s.t. $z = g^k$ and $b = z^w = g^{kw} = a^k$. Call such C2 query *k-computed.* Note that E_{H_2} requires that $i = (1, g^k)$, i.e. that $\mathcal{A}$ queries H_2 on (x, v) for $v = (h_x)^k$. Call such H_2 query *k-computed* as well. Since counter tx is decremented, or C-record $(x, w, a, 1)$ is "processed" only on such k-computed C2 and H_2 queries, and tx is incremented or record $(x, w, a, 1)$ is created with each query to S1, hence E happens only if $\mathcal{A}$ triggers more k-computed C2/H_2 queries than S1 queries.

Correlations monitored only at evaluation. Before we show that event E can happen with at most negligible probability, we need to change the way Game 10 and Game 11 build correlations in function R. Instead of setting them at the time a new function is added, in the modified games the correlations are checked only when a function is evaluated, i.e. the game keeps track of each referenced value of function R, i.e. each triple (δ', z', x') s.t. $R((\delta', z'), x')$ was queried eiter in S2, C2, or H_2. When the game queries a new point, $R(i, x)$ for $i = (\delta, z)$, the game looks for the first record (δ, z', x') on the list of queries s.t. $x' = x$ and $f_{(\delta', z')}(x) = f_{(\delta, z)}(x)$, i.e. $\delta'(z')^{u_x} = \delta(z)^{u_x}$. If so, the game first assigns $R(i, x) \leftarrow R(i', x)$ for $i = (\delta, z)$ and $i' = (\delta', z')$ and only then replies $R(i, x)$. It is easy to see that this is an equivalent process of keeping correlations because indeed the only information about these functions $R(i, \cdot)$ which the game reveals is through evaluated points, so it makes no difference if we postpone correlating values of $R(i, x)$ with $R(i', x)$ until $R(i, x)$ is actually queried.

We show a reduction to the Gap$^+$-OMDH assumption in the case E happens in Game 10. Reduction $\mathcal{R}$ takes the Gap$^+$-OMDH challenge $(g, z^*, h_1, \ldots, h_N)$ where $N = (q_{\mathrm{H}_1} + q_{\mathsf{C}})$, and responds to $\mathcal{A}$'s queries as follows:

1. Initialize $\mathcal{N}_{\mathsf{SIM}} \leftarrow [(1, z^*)]$ and $\mathcal{S} \leftarrow []$.
2. Embed OMDH challenges into H_1 and C1 outputs, i.e. assign each $H_1(x)$ output, and each value a sent by C1, to a unique OMDH challenge h_i.
3. On message a to S1, use oracle $(\cdot)^k$ to send back $b = a^k$ and $z = z^*$.
4. On query x to S2, set $(a, b, z) \leftarrow (1, 1, z^*)$, run Correlate$((a, b, z), x)$, and output $R((a, b, z), x)$
5. On message (b, z) to C2, recovers C1 input x and output a, run Correlate$((a, b, z), x)$, and output $R((a, b, z), x)$.
6. On query (x, v) to H_2, set $(a, b, z) \leftarrow (1, v, 1)$, run Correlate$((a, b, z), x)$, and output $R((a, b, z), x)$.
7. If $\mathcal{A}$ queries S-Compromise, $\mathcal{R}$ aborts.
8. Correlate$((a, b, z), x)$: Return if $(a, b, z, x) \in \mathcal{S}$. Otherwise, set $h_x \leftarrow H_1(x)$, and if $\exists\ (a', b', z', x)$ in $\mathcal{S}$ s.t.
$$b \cdot \mathsf{CDH}_g(z, h_x/a) = b' \cdot \mathsf{CDH}_g(z', h_x/a') \qquad (8)$$
then set $R((a, b, z), x) \leftarrow R((a', b', z'), x)$. Otherwise add (a, b, z, x) to $\mathcal{S}$.

Observe that $\mathcal{R}$ can verify equation (8) using oracle DDH_g^+. Secondly, observe that $b \cdot \mathsf{CDH}_g(z, h_x/a)$ correctly evaluates $f_i(x)$ for the corresponding index i: In S2 we set $(a, b, z) = (1, 1, z^*)$, so $b \cdot \mathsf{CDH}_g(z, h_x/a) = \mathsf{CDH}_g(z^*, h_x) = (h_x)^k$ where $z^* = g^k$, as in Game 10; In C2, in Game 10 we compute $f_i(x) = f_{(\delta,z)}(x) = \delta \cdot (z)^{u_x} = \delta \cdot \mathsf{CDH}(z, h_x)$, but since $\delta = b/z^w = b \cdot \mathsf{CDH}(z, a^{-1})$ this implies that $f_i(x) = \delta \cdot \mathsf{CDH}(z, h_x/a)$; In H_2 we set $(a, b, z) = (1, v, 1)$, so $b \cdot \mathsf{CDH}_g(z, h_x/a) = v \cdot \mathsf{CDH}_g(1, h_x) = v$, also as in Game 10.

Therefore $\mathcal{R}$ presents a view which is ideantical to Game 10 as long as S-Compromise is not queried. Therefore event E occurs in the interaction with $\mathcal{R}$ with the same probability as in Game 10. Let $Q = q_{\mathsf{S}}$ be the number of S1 queries, hence the number of $(\cdot)^k$ oracle accesses by $\mathcal{R}$. Event E implies that the number of k-computed C2 queries and k-computed H_2 queries is larger than Q, i.e. at least $Q+1$. Note that a k-computed H_2 query is a pair (x, v) s.t. $v = (h_x)^k$, so each such query computes $(h_i)^k = \mathsf{CDH}(h_i, z^*)$ on a unique OMDH challenge h_i. Likewise, a k-computed C2 query is a response $(b, z) = (a^k, g^k)$ to C1's message a, and since $\mathcal{R}$ embeds a unique OMDH challenge h_i into each a, such query also computes $a^k = \mathsf{CDH}(h_i, z^*)$ on a unique OMDH challenge h_i. Since $\mathcal{R}$ can use DDH_g^+ oracle to implement DDH, and test whether any H_2 or C2 query is k-computed, $\mathcal{R}$ will solve $Q + 1$ OMDH challenges if event E happens, which implies
$$|\Pr[\mathbf{G_{11}}] - \Pr[\mathbf{G_{10}}]| \leq \epsilon_{\mathrm{OMDH}}(\mathbb{G}, q_{\mathrm{H}_1}, q_{\mathsf{S}})$$

Summing up we conclude that the real-world and the ideal-world interactions are indistinguishable under the Gap-OMDH assumption.

5 Strong aPAKE Protocol Based on $\mathcal{F}_{\mathsf{corOPRF}}$

We show that the OPAQUE protocol of [17] remains secure as UC Strong aPAKE even if it is instantiated with the UC Correlated OPRF of Section 3 instead of UC OPRF of [11]. This implies that one can safely modify the OPAQUE protocol by replacing the *exponential* blinding in the Hashed Diffie-Hellman OPRF with the *multiplicative* blinding (as done in [22]), thus shaving off either 1 variable-base exponentiation from the client, or 2 such exponentiations if the protocol is routinely performed with the same server.

Technically, we show that the OPAQUE compiler construction of [17], which shows that OPRF + AKE $\rightarrow$ saPAKE, can be used to construct UC saPAKE from any UC Correlated OPRF and any UC AKE which is adaptively secure and resilient to Key-Compromise Impersonation attack (AKE-KCI). We call this compiler OPAQUE+ and show it in Fig. 10. It is exactly the same as the OPAQUE compiler except that the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ used in [16] is replaced with the Correlated OPRF functionality $\mathcal{F}_{\mathsf{corOPRF}}$. We show that protocol OPAQUE+ realizes the UC saPAKE functionality.

The saPAKE and AKE-KCI functionalities. Protocol OPAQUE+ and its analysis build on two functionalities from of [16]: The (strong) aPAKE functionality $\mathcal{F}_{\mathsf{saPAKE}}$ and the adaptively-secure UC AKE-KCI functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$. We refer to that paper for their detailed description and rationale. We note that AKE-KCI protocol can be instantiated, for example, by the 3-message version of the HMQV protocol, called HMQV-C in [20], or the 3-message SIGMA protocol [19] underlying the design of TLS 1.3.

Security of OPAQUE+. We now state the security of OPAQUE+ in Theorem 2. As in [17], we assume that the adversary $\mathcal{A}$ always sends $(\textsc{Compromise}, \mathsf{sid})$ aimed at $\mathcal{F}_{\mathsf{corOPRF}}$ and $(\textsc{StealPwdFile}, \mathsf{sid})$ aimed at S simultaneously, since in the real world when the attacker compromises the server, the corresponding OPRF session is always compromised simultaneously.

Theorem 2. *If protocol Π realizes functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$, then protocol OPAQUE+ in Fig. 10 realizes the strong aPAKE functionality $\mathcal{F}_{\mathsf{saPAKE}}$ in the $(\mathcal{F}_{\mathsf{corOPRF}}, \mathcal{F}_{\mathsf{RO}})$-hybrid model.*

The security argument is very similar to that of OPAQUE in [17]; we briefly explain the differences. First of all, note that when the adversary acts as the client in Correlated OPRF, its power is exactly the same as the client in OPRF, hence for that case the security argument is the same in OPAQUE+ as in OPAQUE.

Secondly, an additional power which Correlated OPRF gives to the adversary is to make correlations between OPRF functions while acting as the server. Yet, this does not change the fact that for every function index i (no matter if $i = \mathsf{S}$ or i is an index created by the adversary) and every value $y \in \{0,1\}^{\ell}$, with overwhelming probability there is at most one argument x s.t. $y = F_i(x)$. In Correlated OPRF the adversary can find F_i with two arguments that form a collision in F_i if it finds (i_1, x_1) and (i_2, x_2) s.t. $F_{i_1}(x_1) = F_{i_2}(x_2)$ and then sets

Public Components:

- KCI-secure AKE protocol Π with private/public keys denoted p_s, P_s, p_u, P_u;
- Random-key robust and equivocable authenticated encryption $(\mathsf{AuthEnc}, \mathsf{AuthDec})$ (see [17] for definitions of these properties);
- Functionality $\mathcal{F}_{\mathsf{corOPRF}}$ with output length parameter τ;

Password Registration

1. On input $(\textsc{StorePwdFile}, \mathsf{sid}, \mathsf{C}, \mathsf{pw})$, S generates keys (p_s, P_s) and (p_c, P_c) and sends $(\textsc{Init}, \mathsf{sid})$ and $(\textsc{OfflineEval}, \mathsf{sid}, \mathsf{S}, \mathsf{pw}, \perp)$ to $\mathcal{F}_{\mathsf{corOPRF}}$. On $\mathcal{F}_{\mathsf{corOPRF}}$'s response $(\textsc{OfflineEval}, \mathsf{sid}, \mathsf{rw})$, S computes $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_c, P_s)$ and records $\mathsf{file}[\mathsf{sid}] \leftarrow (p_s, P_s, P_c, c)$.

Server Compromise

1. On $(\textsc{StealPwdFile}, \mathsf{sid})$ from $\mathcal{A}$, S retrieves $\mathsf{file}[\mathsf{sid}]$ and sends it to $\mathcal{A}$.

Login

1. On $(\textsc{UsrSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw}')$, C sends $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{corOPRF}}$ and records $\mathcal{F}_{\mathsf{corOPRF}}$'s response $(\mathsf{Prefix}, \mathsf{ssid}, \mathsf{prfx})$.
2. On $(\textsc{SvrSession}, \mathsf{sid}, \mathsf{ssid})$, S retrieves $\mathsf{file}[\mathsf{sid}] = (p_s, P_s, P_c, c)$, sends c to C, sends $(\textsc{SndrComplete}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{corOPRF}}$, and given $\mathcal{F}_{\mathsf{corOPRF}}$'s response $(\mathsf{Prefix}, \mathsf{ssid}, \mathsf{prfx}')$ it runs Π on input (p_s, P_s, P_c) and $\mathsf{ssid}_\Pi = [\mathsf{ssid}||\mathsf{prfx}']$.
3. On $(\textsc{Eval}, \mathsf{sid}, \mathsf{ssid}, \mathsf{rw}')$ from $\mathcal{F}_{\mathsf{corOPRF}}$ and c from S, C computes $m \leftarrow \mathsf{AuthDec}_{\mathsf{rw}'}(c)$. If $m = (p_c', P_c', P_s')$ then C retrieves $(\mathsf{Prefix}, \mathsf{ssid}, \mathsf{prfx})$ and runs Π on input (p_c', P_c', P_s') and $\mathsf{ssid}_\Pi = [\mathsf{ssid}||\mathsf{prfx}]$; else C outputs $(\textsc{abort}, \mathsf{sid}, \mathsf{ssid})$ and halts.
4. Given Π's local output SK, each party outputs $(\mathsf{sid}, \mathsf{ssid}, SK)$.

Fig. 10. OPAQUE+: Strong aPAKE in the $(\mathcal{F}_{\mathsf{corOPRF}}, \mathcal{F}_{\mathsf{RO}})$-Hybrid World

F_i to be correlated with F_{i_1} on x_1 and with F_{i_2} on x_2. In OPRF the adversary must look for such collisions within each function separately, but in either case the probability of a collision is upper-bounded by $q^2/2^\ell$ where q is the number of F evaluations on all indices. Hence the ciphertext c^* sent from the adversary to an honest client together with index i^* of the random function F_{i^*} which the adversary makes that honest client compute on its password, together commit to a unique password guess pw^* such that $\mathsf{AuthDec}_{\mathsf{rw}^*}(c^*) \neq \perp$ for $\mathsf{rw}^* = F_{i^*}(\mathsf{pw}^*)$.

Lastly, in the Correlated OPRF an adversarial function F_{i^*} is not guaranteed to be completely independent from the honest server's function F_k for every $i^* \neq \mathsf{S}$. Instead, the adversary can correlate F_{i^*} with F_k, although on only a single point x. This allows the adversary a potentially damaging behavior in which it forwards ciphertext $c^* = c$ from the honest server to the honest client and lets the honest client evaluate F_{i^*} on its password. In case both parties' passwords are equal to x the client will compute $F_{i^*}(x) = F_k(x)$, and thus the two parties will establish a key if their shared passwords are equal to x, and fail to establish a key otherwise. This "conditional password test" could not be done in protocol OPAQUE, and yet it is not an attack on saPAKE, because it requires the adversary to guess the password; therefore, the simulator can (1) use a TestAbort command to check if the client and server's passwords match, and if so, it can then (2) use a TestPwd command to check if the adversary's password guess is correct. If both checks pass, the simulator can compromise both client's and server's sessions, and make these two sessions connect with the same session key; if either check fails, the simulator can force the client to abort.

We present the full proof of Theorem 2 in the full version of this paper [15].

6 Concrete OPAQUE+ Instantiation Using HMQV

Figure 11 shows a concrete instantiation of protocol OPAQUE+ of Figure 10, where the UC Correlated OPRF is instantiated with protocol Mult-2HashDH, and UC AKE is instantiated with HMQV [20]. Note that the protocol takes 3 flows (τ_s can be piggybacked with S's earlier message), and 2 fixed-base (fb) and 2 variable-base (vb) (multi-base) exp's for C and resp. 1fb and 2vb exp's for S.

7 Insecure Applications of Multiplicative Blinding

As we noted in the introduction, the correlations allowed by Mult-2HashDH can be exploited in some applications for the benefit of a corrupt server. We illustrate this ability with several examples.

Consider a setting where a client C with input x interacts using Mult-2HashDH with a server S with key k to compute $y = F_k(x) = H_2(x, (h_x)^k)$ where h_x denotes $H_1(x)$. C then uses y for some task; for concreteness, think of x as a password and y as a key that allows C to authenticate to some application. At some point S becomes corrupted and wants to check whether a given value x' equals the user's input x. Using correlations as described in the introduction, e.g., equation (3), S mounts the following attack: When C sends its blinded value $a = h_x g^r$, S chooses random k', sets $z = g^{k'}$ and $b = (h_{x'})^{k-k'} a^{k'}$, and sends (b, z) to C, who computes the unblinded value $v = b(z)^{-r}$ and outputs $y' = H_2(x, v)$. It can be checked that $v = (h_x)^k$ if and only if $x' = x$.[11] If S can observe whether C recovered the correct value $y' = y$, e.g. whether it successfully authenticated using the recoverd y', then S learns whether C's secret x equals S's guess x'.

The Correlated OPRF functionality, which Mult-2HashDH realizes, assures that server S cannot test more than one guess x' per interaction, and while in some applications, like the PAKE protocol OPAQUE, this ability doesn't affect the application, e.g. because the application itself allows the attacker such on-line guess-and-test avenue, in other cases this suffices to break the

[11] Observe that $v = bz^{-r} = (h_{x'})^{k-k'}(h_x g^r)^{k'}(g^{k'})^{-r} = h_{x'}^k (h_{x'}/h_x)^{k'}$, hence $v = (h_x)^k$ iff $h_x = h_{x'}$. Using the terminology of equation (2), C computes $y' = F_{(\delta,z)}(x)$ for $F_{(\delta,z)}$ which is *correlated* with F_k on x', hence $y' = F_k(x)$ iff $x = x'$.

application. Below we show a few application examples which are all secure with Exp-2HashDH, but not with Mult-2HashDH. In all examples the application doesn't expose the client to on-line attacks, and using Exp-2HashDH ensures that the implementation does not either, but using Mult-2HashDH adds this exposure and breaks the application.

Public Parameters and Components

Group $\mathbb{G}$ of prime order q with generator g;
Random-key robust and equivocable authenticated encryption $(\mathsf{AuthEnc}, \mathsf{AuthDec})$ (see [17] for definitions of these properties);
Hash functions $H_1, H_3, H_5 : \{0,1\}^* \to \{0,1\}^\tau$, $H_2 : \{0,1\}^* \to \mathbb{G}$, $H_4 : \{0,1\}^* \to \mathbb{Z}_q$
Pseudorandom function (PRF) $f : \{0,1\}^* \to \{0,1\}^\tau$

S on $(\textsc{StorePwdFile}, \mathsf{sid}, \mathsf{pw})$

Pick $k, p_s, p_c \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $(z, P_s, P_c) \leftarrow (g^k, g^{p_s}, g^{p_c})$, $\mathsf{rw} \leftarrow F_k(\mathsf{pw})$, $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_c, P_s)$; record $\mathsf{file}[\mathsf{sid}] \leftarrow \langle k, z, c, p_s, P_c \rangle$ and erase everything else

C on $(\textsc{UsrSession}, \mathsf{sid}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw})$ S on $(\textsc{SvrSession}, \mathsf{sid}, \mathsf{ssid})$

retrieve $\langle k, z, c, p_s, P_c \rangle \leftarrow \mathsf{file}[\mathsf{sid}]$

$r, x_c \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ $x_s \leftarrow_{\mathrm{R}} \mathbb{Z}_q$

$a \leftarrow H_2(\mathsf{pw}) \cdot g^r, X_c \leftarrow g^{x_c}$ $\xrightarrow{a,\ X_c}$

$\mathsf{rw} \leftarrow H_1(\mathsf{pw}, b \cdot z^{-r})$ $\xleftarrow{b,\ z,\ X_s,\ c}$ $b \leftarrow a^k, X_s \leftarrow g^{x_s}$

parse $(p_c, P_s) \leftarrow \mathsf{AuthDec}_{\mathsf{rw}}(c)$
(if this parsing fails C outputs $\textsc{abort}$)

C and S set $\mathsf{ssid}' \leftarrow H_5(\mathsf{sid}, \mathsf{ssid}, a)$, $e_c \leftarrow H_4(X_c, \mathsf{S}, \mathsf{ssid}')$, $e_s \leftarrow H_4(X_s, \mathsf{C}, \mathsf{ssid}')$

$K \leftarrow H_3\left((X_s P_s^{e_s})^{x_c + e_c p_c}\right)$ $K \leftarrow H_3\left((X_c P_c^{e_c})^{x_s + e_s p_s}\right)$

$\xleftarrow{\tau_s}$ $\tau_s \leftarrow f_K(1, \mathsf{ssid}')$

if $\tau_s \neq f_K(1, \mathsf{ssid}')$: $(SK, \tau_c) \leftarrow (\perp, \perp)$
else: $SK \leftarrow f_K(0, \mathsf{ssid}')$ and $\tau_c \leftarrow f_K(2, \mathsf{ssid}')$

$\xrightarrow{\tau_c}$ if $\tau_c \neq f_K(2, \mathsf{ssid}')$: $SK \leftarrow \perp$
else: $SK \leftarrow f_K(0, \mathsf{ssid}')$

output $(\mathsf{sid}, \mathsf{ssid}, SK)$ output $(\mathsf{sid}, \mathsf{ssid}, SK)$

Fig. 11. Protocol OPAQUE+ (Fig. 10) with Mult-2HashDH and HMQV

OPAQUE with outsourced envelope. Recall that OPAQUE [17] combines an OPRF with an authenticated key-exchange (AKE) protocol as follows: At registration, the server and the user choose private-public AKE key pairs. The user then runs an OPRF with the server where the user's input is a password pw and the server's input is an OPRF key k. The output of the OPRF, learned only by the user, is a random key $\mathsf{rw} = F_k(\mathsf{pw})$, and the user uses rw to authenticate-encrypt her AKE private key and the server's public key. The ciphertext c that results from this encryption is stored by the server, together with the OPRF key k, the user's public AKE key, and the server's AKE key pair. At login, the user runs the OPRF with the server on input pw, learns rw, uses rw to decrypt its own private key and the server's public key encrypted in c, and uses these keys to run the AKE with the server. Only a user in possession of the registered password can successfully run the AKE.

However, consider a modification where the user stores ciphertext c at some other location than server S, e.g. a laptop or another server. In this case a malicious S, who holds only OPRF key k and the AKE keys, cannot stage either online or offline attacks on the user's password: Without ciphertext c, S cannot test candidate values $\mathsf{rw} = F_k(\mathsf{pw})$. However, this property is *not* ensured if OPRF is implemented with Mult-2HashDH. Indeed, using the strategy described above, a malicious S can test whether the user's password is equal to a chosen pw^*, by running login using function F_{k^*} which is correlated on argument pw^* with function F_k used in registration. If the user recovers its credentials and authenticates in that login, S learns that $\mathsf{pw} = \mathsf{pw}^*$. Crucially, this online attack opportunity for server S is not available using Exp-2HashDH.

Device-enhanced PAKE. [14, 24] presents a password protocol that uses an auxiliary device (typically a smartphone but can also be an online server) in the role of a password manager. When the user wishes to password-authenticate to a server S, it communicates with the device who holds key k for 2HashDH OPRF. The user's input to the OPRF is her password, and the OPRF result $\mathsf{rw} = F_k(\mathsf{pw})$ is used as the "randomized" password with service S. Using Exp-2HashDH, a corrupt device learns nothing about the user's password, but it can test a guess for the user's password at the cost of one online interaction with S per guess. However, using Mult-2HashDH, the corrupt device can validate a guess without interacting with S, by watching if the user's interaction with S succeeded, thus resulting in weaker security guarantees.

Threshold OPRF (including Threshold OPAQUE). A multi-server threshold implementation of Exp-2HashDH is presented in [12]. It ensures the security of the OPRF as long as no more than a threshold of servers are compromised. Such threshold OPRF can be used e.g. to construct Password-Protected Secret Sharing (PPSS) [1, 11], which in turn can implement Threshold PAKE. It is straightforward to see that the above correlation attacks apply to these constructions if Exp-2HashDH is replaced with Mult-2HashDH. They allow a single corrupted server to choose correlated values with which it can verify guesses for the client's inputs. As an illustration, consider a 2-out-of-2 Threshold OPRF that computes h_x^k as $h_x^{k_1+k_2}$ using two servers S_1, S_2 with respective keys k_1, k_2. Such a scheme should ensure that nothing can be learned about the input x without compromising both servers. However, a corrupted S_2 can check whether C's input x equals any guess x' by mounting the above attack using ony key k_2. If C reconstructs the correct y, then $x = x'$. This attack also applies to OPAQUE with a multi-server threshold implementation of Mult-2HashDH.

All these examples show that in order to use Mult-2HashDH in an application where an authenticated g^k is not available to the client, a dedicated proof of security (as the one we develop here for OPAQUE) is essential. Even in that case, one can consider this as "fragile evidence", as eventual changes to the application may void the security proof. Thus a safer alternative is to use the scheme (4) presented in the introduction, which implements UC OPRF using both forms of blinding, and would be secure in all the above applications.

References

1. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *ACM Conference on Computer and Communications Security — CCS 2011*. ACM, 2011.
2. X. Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In *Cryptology and Network Security – CANS 2009*, pages 279–298. Springer, 2009.
3. E. F. Brickell, D. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation. In *Advances in Cryptology – EUROCRYPT 1992*, pages 200–207. Springer, 1992.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science – FOCS 2001*, pages 136–145. IEEE, 2001.
5. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology – CRYPTO 1992*, pages 89–105. Springer, 1992.
6. A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. Privacy pass: Bypassing internet challenges anonymously. In *Privacy Enhancing Technologies Symposium – PETS 2018*, pages 164–180. Sciendo, 2019.
7. W. Ford and B. S. Kaliski. Server-assisted generation of a strong secret from a password. In *IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises – WET ICE 2000*, pages 176–180. IEEE, 2000.
8. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography – TCC 2005*, pages 303–324. Springer, 2005.
9. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. In *CHES*, 2019.
10. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology – ASIACRYPT 2014*, pages 233–253. Springer, 2014.
11. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-efficient and composable password-protected secret sharing (Or: how to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy – EuroS&P 2016*, pages 276–291. IEEE, 2016.
12. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In *Applied Cryptography and Network Security – ACNS 2017*, pages 39–58. Springer, 2017.
13. S. Jarecki, H. Krawczyk, and J. Resch. Updatable oblivious key management for storage systems. In *ACM Conference on Computer and Communications Security — CCS 2019*. ACM, 2019.

14. S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-enhanced password protocols with optimal online-offline protection. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 177–188. ACM, 2016.
15. S. Jarecki, H. Krawczyk, and J. Xu. On the (In)Security of the Diffie-Hellman Oblivious PRF with Multiplicative Blinding. *IACR Cryptology ePrint Archive*, 2021:273.
16. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. *IACR Cryptology ePrint Archive*, 2018:163.
17. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *Advance in Cryptology – EUROCRYPT 2018*, pages 456–486. Springer, 2018.
18. S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks – SCN 2010*, pages 418–435. Springer, 2010.
19. H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Advances in Cryptology – CRYPTO 2003*, pages 400–425. Springer, 2003.
20. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol (extended abstract). In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.
21. H. Krawczyk. The OPAQUE asymmetric PAKE protocol, `https://tools.ietf.org/html/draft-krawczyk-cfrg-opaque`, May 2020.
22. H. Krawczyk, K. Lewi, and C. A. Wood. The OPAQUE asymmetric PAKE protocol, `https://tools.ietf.org/html/draft-irtf-cfrg-opaque`, November 2020.
23. M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Advances in Cryptology – EUROCRYPT 1999*, pages 327–346. Springer, 1999.
24. M. Shirvanian, N. Saxena, S. Jarecki, and H. Krawczyk. Building and studying a password store that perfectly hides passwords from itself. *IEEE Transactions on Dependable and Secure Computing*, 16:5, 2019.
25. N. Sullivan. Exported authenticators in TLS, `https://tools.ietf.org/html/draft-ietf-tls-exported-authenticator`, May 2020.
26. N. Sullivan, H. Krawczyk, O. Friel, and R. Barnes. Usage of OPAQUE with TLS 1.3, `https://tools.ietf.org/html/draft-sullivan-tls-opaque`, Mar 2019.