The 2Hash OPRF Framework and Efficient Post-Quantum Instantiations

Ward Beullens[1], Lucas Dodgson[2], Sebastian Faller[1,2][0009-0005-4126-3098], and Julia Hesse[1]

[1] IBM Research Europe – Zurich, Switzerland
[2] ETH Zurich, Switzerland

Abstract. An Oblivious Pseudo-Random Function (OPRF) is a two-party protocol for jointly evaluating a Pseudo-Random Function (PRF). OPRFs are a prime tool for building secure authentication and key exchange from passwords, private set intersection, private information retrieval, and many other privacy-preserving systems. While classical OPRFs run as fast as a TLS Handshake, current *quantum-safe* OPRF candidates with malicious security are still practically inefficient.

In this paper, we propose a framework for constructing OPRFs from secure two-party computation. The framework captures a family of so-called *2Hash PRFs*, which sandwich a function evaluation in between two hashes. The core of our framework is a compiler that yields an OPRF from a secure evaluation of any function that is key-collision resistant and one-more unpredictable. We instantiate this compiler by providing such functions built from Legendre symbols or from a block cipher. We then give a case-tailored protocol for securely evaluating our Legendre-based function, built from Oblivious Transfer (OT) and Zero-Knowledge Proofs (ZKP). Instantiated with lattice-based OT and proofs based on Vector Oblivious Linear Evaluation (VOLE), we obtain the first somewhat practically efficient quantum-safe OPRF with malicious and composable security guarantees. Instantiated for 128 bits of security, we estimate that our OPRF would finish in approximately 0.6 seconds on a WAN network with 30 ms latency and 25 Mbps bandwidth with less than 1 MB of total communication.

Keywords: Oblivious Pseudo-Random Function · Secure Function Evaluation · Universal Composability.

1 Introduction

A pseudo-random function (PRF) is a function that is indistinguishable from a truly random function. In 1997, Naor and Reingold [41] presented the first *oblivious* evaluation protocols for pseudo-random functions (OPRF). An OPRF is a protocol between a user and a server, where the user contributes an input x and the server contributes a PRF key K. The result of the protocol is that the user learns $\mathrm{PRF}_K(x)$, while the server learns nothing. These strong secrecy guarantees about user inputs and server keys render OPRFs an extremely useful tool for building all kinds of privacy preserving systems [25,33,5,24,28,37]. In particular, OPRFs can be used to amplify the entropy of user passwords pw by computing $\mathrm{PRF}_K(pw)$ with the help of a server, making them a core component of modern password-based protocols [31,9,34,19,18,40]. Billions of users rely on OPRF-based protocols today, e.g., through WhatsApp's password-protected chat history backups [45,19] or PrivacyPass [44], a Chrome/Firefox extension to replace Captchas. All these real-world systems deploy a discrete-log-based OPRF called *2-Hash Diffie Hellman* (2HashDH) [27]. Since Shor's algorithm computes discrete logarithms in polynomial time on a quantum computer, current OPRF-protected infrastructures are not yet quantum-safe.

1.1 Related work

The 2-Hash Diffie Hellman OPRF. 2HashDH [27] is arguably the most successful OPRF. An evaluation of this OPRF is $F_k(x) := H_2(x, H_1(x)^k)$, where H_1 is a random oracle that hashes into a prime order group G. To jointly evaluate the OPRF

- the user first sends $a := H(x)^r$ to the server, where r is a random value.
- the server answers with $b := a^k$, and
- finally the user computes $c := b^{1/r} = H_1(x)^k$ and outputs $H_2(x, c)$.

This OPRF protocol is very simple and efficient, e.g., it has an optimal round complexity. It also achieves the currently strongest security notion for OPRFs [29] in the Universal Composability (UC) framework [14], under the one-more Diffie Hellman assumption in the group G, in the random oracle model [29].

Replicating the success of 2HashDH to design quantum-safe OPRFs has proven to be difficult. The blind-exponentiate-unblind protocol underlying the 2HashDH OPRF relies on strong algebraic structure which is hard to come by in the world of post-quantum cryptography. Since exponentiation in groups is broken by Shor's algorithm, the next best thing seems to be group actions. Unfortunately, most of the cryptographic group actions in the post-quantum cryptography literature are non-abelian, which breaks the blind-act-unblind protocol. A notable exception is the commutative CSIDH group action [16] on a set of supersingular elliptic curves, for which the blind-act-unblind protocol works without problems, but alas, the problem is that it is notoriously difficult to hash into the set of supersingular elliptic curves in a way that does not reveal the endomorphism ring of $H_1(x)$. Revealing the endomorphism ring of $H_1(x)$ would allow one to compute $F_k(x)$ for all x after seeing only one evaluation $F_k(x')$, so we sadly cannot instantiate the Supersingular Isogeny variant of the 2HashDH protocol.

Some post-quantum OPRFs, such as variants of the isogeny-based OPRF of Boneh et al. [13,6] and the lattice-based OPRF of Albrecht et al. [3], mimic the blind-exponentiate-unblind protocol, but unlike 2HashDH they require very expensive zero-knowledge proofs to enforce honest behavior of the server, which makes these protocols too inefficient to be used in practice.

There have been attempts to build OPRF protocols from approaches not inspired by the blind-exponentiate-unblind protocol (e.g., the Naor-Reingold OPRF, see [15] for more), but to the best of our knowledge, none of these attempts (classical or post-quantum) have been able to realize the strongest security definitions with malicious security.

A very brief survey of quantum-safe OPRFs. Post-quantum OPRFs have been an active research area recently. The first post-quantum OPRFs were presented by Boneh et al. [13], who presented two isogeny-based OPRFs. The first relies on a hardness assumption which has since been shown to be insecure (SIDH) [7,38]. An updated version of this OPRF was introduced by Basso [6] with countermeasures against the attacks of [7,38] and the recent attacks on SIDH. It is round-optimal, requiring only two rounds of communication for an evaluation. The required bandwidth for it is 3 MB when the verifiability property of the OPRF is not needed and 8.7 MB for the verifiable variant that offers security in case of a malicious server. No implementation for this OPRF is available, and we expect the OPRF to be quite slow, since it uses lots of arithmetic modulo very large primes, e.g., 8868-bit primes for 128 bits of security.

The second OPRF of Boneh et al. is based on the CSIDH assumption but no security proof in the Universal Composability framework for it was included and it requires the server to be semi-honest. Boneh et al. estimate that using a 5280-bit prime, their CSIDH-based construction has a communication cost of around 424KB in 3 rounds of communication. However, running the protocol requires knowledge of the relation lattice of the CSIDH ideal class group, which takes a subexponential amount of work to compute. We currently cannot instantiate this OPRF with 5280-bit primes, since the largest CSIDH prime for which the relation lattice has been computed is only 512 bits long [11]. Another CSIDH-based OPRF, called OPUS, was introduced in a recent work [26]. OPUS does not require the relation lattice, so it can be run with primes with more than 512 bits. However, it has a round complexity of $O(\lambda)$, and like the original CSIDH-based OPRF of [13] it is still only secure in the semi-honest server setting.

A lattice-based OPRFs with malicious security was introduced by Albrecht et al. [3]. It requires more than 128 GB of communication, limiting its potential usability. It is proven secure using a non-standard security definition, so the OPRF might not be suitable for arbitrary use cases.

Multiple works have investigated using the Dark Matter weak PRF [12] in an MPC setting as an OPRF. Dinur et al. proposed doing so using secret-sharing [20], resulting in an OPRF secure in the semi-honest model which requires preprocessing. Albrecht et al. instead used torus fully homomorphic encryption [2], resulting in a scheme that requires communication of around 70 MB for an evaluation, whereby only 3 MB is required for the online phase. Again, the OPRF was only shown to be secure in the case of a semi-honest server, although a potential extension based on heuristics to a verifiable OPRF with malicious security was also discussed.

Grassi et al. [23] investigate which pseudo-random functions can be evaluated efficiently with general-purpose secret-sharing-based MPC protocols, identifying the Legendre PRF and MiMC as particularly suitable candidates. This naturally gives rise to OPRF protocols. Seres et al. [43] observe that the Legendre PRF has a limited form of programmability and that the Legendre OPRF of Grassi et al. can be made verifiable using zero-knowledge proofs. None of these works establish any composable security guarantees of the Legendre OPRF, and their use of generic MPC tools comes with some overhead that makes the OPRFs only somewhat efficient. E.g., while the cost of a single evaluation of the LegendrePRF is not reported, Grassi et al. report a throughput of 2.1 seconds per evaluation. However, note that this is the throughput when the LegendrePRF is instantiated with a 127-bit prime, which we now know to be insufficient because of attacks on the Legendre PRF that were discovered more recently [36,10,35]. Faller et al. [22] proposed an OPRF scheme based on the secure evaluation of AES using garbled circuits which is shown to achieve the same security level in the UC framework as 2HashDH *if* the server is assumed to be semi-honest, but is not secure against a malicious server.

In summary, while there exist somewhat practical post-quantum OPRFs, there is still no practical post-quantum OPRF with malicious security and composable security guarantees.

1.2 Contributions/Technical summary

Studying 2Hash OPRF protocols. The blind-exponentiate-unblind secure evaluation protocol underlying the 2HashDH protocol has proven to be difficult to replicate in the post-quantum setting. However, secure function evaluation is possible generically from oblivious transfer, and this has become increasingly practical in the last decade. Therefore, to construct an OPRF protocol, it is natural to use a secure two-party computation protocol for a keyed function $f : \mathcal{I} \times \mathcal{K} \to \mathcal{Y}$, where the user contributes an input $h \in \mathcal{I}$, the server contributes a key $K \in \mathcal{K}$, and at the end the user learns $f(h, K) \in \mathcal{Y}$, while the server does not learn anything about the users input h.

To handle arbitrarily long inputs efficiently, it makes sense to hash the user's input into the input space $\mathcal{I}$ of f. Moreover, if one wants to prove the OPRF protocol secure in the UC framework, it is necessary to run the output $f(\mathsf{H}_1(x), K)$ through a second random oracle $\mathsf{H}_2 : \mathcal{Y} \to \{0, 1\}^\lambda$.[3] We then arrive at the following generic blueprint for building OPRFs, which is clearly inspired by 2HashDH:

[3] This is because if the user was to output $f(\mathsf{H}_1(x), K)$ directly, without running it through an idealized primitive such as a random oracle, then the server would be able to offline evaluate the OPRF without revealing any information about his key K to the UC simulator. This makes it impossible to simulate online executions of the protocol whose output needs to be consistent with offline evaluations.

– First, the Server samples a long-term key K from $\mathcal{K}$.

– To evaluate the OPRF on input $x \in \{0,1\}^*$, the user computes $\mathsf{H}_1(x)$,

– Then the secure evaluation protocol is executed with user input $H_1(x)$, and server input K to let the user learn $f(H_1(x), K)$.

– Finally, the user outputs $H_2(x, f(H_1(x), K))$.

We call OPRF protocols that follow this blueprint "2Hash OPRFs". It is then a natural question to ask, given a generic function evaluation protocol for f (in the UC framework, such generic protocols can be captured through the availability of an ideal functionality $\mathcal{F}^f_{\text{SFE}}$, also called the $\mathcal{F}^f_{SFE}$*-hybrid model*), what properties f should satisfy for a 2Hash OPRF to be as secure as 2HashDH. The first major contribution of this paper is to answer this question. We identify (n,q)-one-more unpredictability and (n)-weak key collision resistance as necessary properties for the 2Hash OPRF protocol to be UC-secure.[4] We then show that these properties are *almost* sufficient to guarantee that the 2Hash OPRF protocol is UC-secure. More precisely, we introduce a new ideal functionality $\mathcal{F}^f_{\text{2H-OPRF}}$, tailor-made to study 2Hash OPRF protocols and we show that if f is one-more unpredictable and weak-key collision resistant, then the induced 2Hash OPRF protocol is a UC-secure realization of the $\mathcal{F}^f_{\text{2H-OPRF}}$ functionality in the $\mathcal{F}^f_{\text{SFE}}$-hybrid model. We dare say that one-more unpredictability and weak-key collision resistance are almost sufficient conditions because the $\mathcal{F}^f_{\text{2H-OPRF}}$ functionality is very close to the $\mathcal{F}_{\text{OPRF}}$ functionality: E.g., *for any* f the $\mathcal{F}^f_{\text{2H-OPRF}}$ functionality can be securely plugged into the OPAQUE and PPSS protocols [31,29]. In fact, we are not aware of any application of $\mathcal{F}_{\text{OPRF}}$ which cannot use $\mathcal{F}^f_{\text{2H-OPRF}}$ instead, regardless of the function f. We give a heuristic explanation of why $\mathcal{F}^f_{\text{2H-OPRF}}$ should be very close to $\mathcal{F}_{\text{OPRF}}$ and we show that they are equivalent in the case of $f(x,k) := x^k$ used by 2HashDH.

A quantum-safe OPRF based on Legendre symbols. The second major contribution of our paper is an efficient, plausibly post-quantum secure instantiation of the 2Hash OPRF framework, based on Oblivious Transfer (OT), Zero-Knowledge (ZK) proofs, and Legendre symbols. We use a function $f_{\mathsf{LSeq}} : \mathbb{F}_p \times \mathbb{F}_p \to \{-1,0,1\}^{\ell_{\mathsf{com}}+\ell_{\mathsf{eval}}}$ defined as

$$f_{\mathsf{LSeq}}(h,K) := \left(\left(\frac{K+l_1}{p}\right), \ldots, \left(\frac{K+l_{\ell_{\mathsf{com}}}}{p}\right), \left(\frac{K+h+l'_1}{p}\right), \ldots, \left(\frac{K+h+l'_{\ell_{\mathsf{eval}}}}{p}\right) \right),$$

where p is a large prime, and $\mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}, \mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$ are randomly sampled vectors. For appropriately chosen parameters $p, \ell_{\mathsf{com}}, \ell_{\mathsf{eval}}$ we can show that this function is one-more unpredictable and key-collision resistant, which means we can use f

[4] Strictly speaking, it is only necessary that it is hard to find (x, x', K, K') such that $f(H(x), K) = f(H(x), K')$ is a collision and $f(H(x'), K) \neq f(H(x'), K')$ is a non-collision. E.g., if f ignores some of the bits of its keys the key-collision resistance would be broken, but the 2Hash OPRF could still be secure.

to construct a 2Hash OPRF. We then design an efficient UC-secure two-party evaluation protocol for f_{LSeq}, to obtain an actively secure plausibly post-quantum secure OPRF from the 2Hash OPRF framework.

The first ℓ_{com} Legendre symbols are independent of the input h to improve efficiency. To prove one-more unpredictability, we need λ Legendre symbols to depend on h, while for weak key-collision resistance, we need more Legendre symbols, but they don't necessarily need to depend on h. Since $\left(\frac{K+l_i}{p}\right)$ is cheaper to compute than $\left(\frac{K+h+l'_i}{p}\right)$ with our secure evaluation protocol, we use only $\ell_{\mathsf{eval}} = \lambda$ Legendre symbols that depend on h, and we let the remaining ℓ_{com} symbols be independent of h. Because the input-independent legendre symbols are collision-resistant by themselves we get a verifiable OPRF for free, although we do not prove this formally in this paper.

Our two-party protocol uses Vector Oblivious Linear Evaluation (VOLE), which lets a server input two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^n$, and lets a user input $h \in \mathbb{F}_p$. At the end of the VOLE, the user learns $\mathbf{u} + h \cdot \mathbf{v}$, while the server learns nothing about the user's input h. We use VOLE to evaluate the input-dependent part of f_{LSeq} as follows: The server picks a random vector $\mathbf{a} \in (\mathbb{F}_p^\times)^{\ell}_{\mathsf{eval}}$, and sends $\mathbf{u} = \{(K + l'_i)a_i^2\}_{i\in[\ell_{\mathsf{eval}}]}$ and $\mathbf{v} = \{a_i^2\}_{i\in[\ell_{\mathsf{eval}}]}$ to the VOLE protocol. The user sends his input h and learns $\mathbf{o} = \mathbf{u} + h \cdot \mathbf{v} = \{(K + h + l'_i)a_i^2\}_{i\in[\ell_{\mathsf{eval}}]}$. Since $(K+h+l'_i)$ is masked by a random nonzero square, the user learns the Legendre symbol of $K + h + l'_i$, but nothing more. Which is exactly what we want. The server sends the input-independent part of f_{LSeq} to the user in the clear. To make the protocol secure against malicious adversaries, we add a zero-knowledge proof that lets the Server prove he behaved honestly, i.e. he proves knowledge of $(K, \mathbf{a}) \in \mathbb{F}_p \times (\mathbb{F}_p^\times)^{\ell_{\mathsf{eval}}}$ that is consistent with the input-independent part of f_{LSeq} and the $\mathbf{u}, \mathbf{v}$ vectors that were entered into the VOLE protocol.[5] However, this means we cannot use the VOLE protocol in a black-box way, because we need to prove a statement about its input $\mathbf{u}, \mathbf{v}$. To work around this problem we use an extended VOLE functionality which we call VOLE+, which in addition to $\mathbf{o}$ also outputs a random hashing key $\boldsymbol{\gamma}$ for a universal hash function $H_{\boldsymbol{\gamma}}$ as well as the hash values $c_{\mathbf{u}} = H_{\boldsymbol{\gamma}}(\mathbf{u})$ and $c_{\mathbf{v}} = H_{\boldsymbol{\gamma}}(\mathbf{v})$. This is all we need to tie the zero-knowledge proof to the VOLE input. The server first commits to $(K, \mathbf{a})$, then the VOLE+ protocol is run, and then the server proves that $(K, \mathbf{a})$ is consistent with the hash values $c_{\mathbf{u}}$ and $c_{\mathbf{v}}$, and the input-independent part of the evaluation that was sent in the clear.[6] We show that VOLE+ can be instantiated from *subset VOLE*, which in turn is known to be instantiable from Oblivious Transfer (OT).

[5] This is a slight simplification. In the actual protocol, instead of using the zero-knowledge proof to prove that $a_i \neq 0$ for all $i \in [\ell_{\mathsf{eval}}]$ (which would be somewhat costly), we let the user learn $\{(K + l'_i)a_i^2 a_{i+1}^2\}_{i\in[\ell_{\mathsf{eval}}]}$ and we let the user abort if there at least two zeroes in the output.

[6] We use $H_{\boldsymbol{\gamma}}(\mathbf{x}) := \langle \boldsymbol{\gamma}, \mathbf{x} \rangle + r$ as the universal hash function because we need the hash to be an $\mathbb{F}_p$-affine function, which also makes it very cheap to prove in zero-knowledge for a public hashing key $\boldsymbol{\gamma}$.

This results in a practical secure evaluation protocol for f_{LSeq}, and hence we obtain a practical OPRF protocol. The total communication cost of the OPRF protocol is less than 1 MB and we estimate that running the protocol would require approximately half a second when run by two parties with a standard modern CPU over a connection with 30 ms of latency, and 25Mbps of bandwidth. This is orders of magnitude faster than existing quantum-safe actively secure OPRF protocols with composable security guarantees.

1.3 Organization

We start with preliminaries on Legendre symbols and secure function evaluation, i.e., specifying $\mathcal{F}^{f}_{\mathrm{SFE}}$, in Section 2. Section 3 presents our 2Hash OPRF framework, including a compiler from $\mathcal{F}^{f}_{\mathrm{SFE}}$ to 2Hash OPRFs for appropriate functions f. To warm up, we then showcase how to use our compiler to build 2Hash OPRFs from block ciphers in Section 4. We then turn to our main result, a 2Hash OPRF from OT and Legendre symbols: Section 5 details the underlying function f_{LSeq}, proves relevant properties of it, and shows how to securely evaluate it using VOLE+. The same section gives concrete parameters and timings. For space constraints, we defer a protocol for building VOLE+ from VOLE to Appendix D. The appendix further contains a lower bound on the security of 2Hash OPRFs derived with our framework (Appendix A), the full proof of our compiler (Appendix B), and the full proof of our secure function evaluation protocol for the Legendre-based function f_{LSeq} (Appendix C).

2 Preliminaries

Notation. Let $p > 2$ be a prime and let $\mathbb{F}_p$ denote the finite field of order p. For two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^k$ we denote by $\mathbf{u} * \mathbf{v}$ the entry-wise product of the vectors, i.e., $(\mathbf{u} * \mathbf{v})_i = u_i v_i$ for all i in $\{1, \ldots, k\}$. And we denote by $\mathbf{u}^2$ the entry-wise square of $\mathbf{u}$, i.e., $\mathbf{u}^2 = \mathbf{u} * \mathbf{u}$. Let $\mathbf{1}_k$ be the vector of length k whose entries are all equal to 1. For $\mathbf{a} \in \mathbb{F}_p^k$, we denote by $\mathrm{rot}(\mathbf{a})$ the vector obtained by rotating the entries of $\mathbf{a}$ by one position, i.e., if $\mathbf{a}' = \mathrm{rot}(\mathbf{a})$ then $a_i = a'_{i+1}$ for all $i \in [k-1]$ and $a_k = a'_1$. We denote by $\mathbf{u} \| \mathbf{v}$ the concatenation of the vectors $\mathbf{u}$ and $\mathbf{v}$. If $y \in \mathbb{F}_p$ is a quadratic residue, we denote by $\sqrt{y}$ the unique element $x \in [0, p/2]$ such that $x^2 = y$. For two vectors $\mathbf{u}, \mathbf{v}$ of the same length n, we denote by $\langle \mathbf{u}, \mathbf{v} \rangle$ the standard dot product, i.e., $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} u_i v_i$. We denote by $\mathsf{dom}(F)$ the set of values function F is defined on.

Legendre Symbols, Legendre PRF, and related assumptions. An element x is a quadratic residue modulo p if an element y exists such that $y^2 = x \bmod p$. For a prime number p and natural number x, the *Legendre symbol* $\left(\frac{x}{p}\right)$ is defined as follows:

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \mod p = \begin{cases} 1 & \text{if } x \neq 0 \text{ is a quadratic residue modulo } p \\ -1 & \text{if } x \text{ is a quadratic non-residue modulo } p \\ 0 & \text{if } x \equiv 0 \text{ modulo } p \end{cases}$$

The Legendre symbol is multiplicative, meaning that for $a, b \in \mathbb{F}_p$ it holds that $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$. The idea of using the Legendre symbols as a pseudorandom number generator goes back to a paper published in 1988 by Damgård [17]. Later, the Legendre pseudorandom function with a single-bit output was defined as $\mathrm{PRF}_k(x) = \left(\frac{k+x}{p}\right)$. The security of this PRF is equivalent to the *Decision Shifted Legendre Symbol* (DSLS) problem [23].

Definition 1 (Decisional Shifted Legendre Symbol (DSLS) Problem). *Let k be chosen randomly and let $\mathcal{O}_{Leg}$ be an oracle that on input x outputs $\left(\frac{k+x}{p}\right)$, and let $\mathcal{O}_R$ be a random oracle that maps elements from $\mathbb{Z}_p$ to $\{-1, 1\}$. The DSLS problem asks to distinguish between $\mathcal{O}_{Leg}$ and $\mathcal{O}_R$ with non-negligible advantage.*

The *DSLS assumption* is then the assumption that there is no efficient polynomial-time algorithm that solves the DSLS problem. Let $g \in \mathbb{F}_p$ be a canonical non-square known to all parties participating in our protocols, e.g., the smallest positive integer such that $\left(\frac{g}{p}\right) = -1$. We will use the fact that $\left(\frac{a}{p}\right) = x$ if and only if there exists $b \neq 0$ such that $ab^2 = E(x)$, where $E(0) = 0, E(1) = 1$, and $E(-1) = g$. Such b can be efficiently computed as $b := \sqrt{E(x)/a}$ if $a \neq 0$ and $b := 1$ otherwise.

Secure function evaluation. We define a secure function evaluation functionality $\mathcal{F}^f_{\mathrm{SFE}}$ that allows leakage of a function of the server's input. Namely, $\mathcal{F}^f_{\mathrm{SFE}}$ allows computation of some function $f := (f_{\mathsf{pub}}, f_{\mathsf{sec}})$ with two inputs, one provided by a User and one provided by a Server. The first component f_{pub} only depends on the Server's input, and is leaked to the adversary. The user learns the function output (including the f_{pub} part), while the server does not learn anything. The functionality is depicted in Fig. 1. We will later use $\mathcal{F}^f_{\mathrm{SFE}}$ to build OPRF protocols.[7]

3 A framework for 2Hash OPRFs

We first recap the definition of OPRFs in the UC framework by Jarecki et al. [31,29], depicted in Figure 2 ignoring the gray parts of the code. To improve the readability, we deviate from the original version of $\mathcal{F}_{\mathrm{OPRF}}$ in [29] by dropping prefixes (which are irrelevant for our work), dropping evaluation tickets that dispense with the need for online extraction (also irrelevant for our work) and

[7] While a formal treatment of Verifiable OPRFs in the 2HashOPRF framework is beyond the scope of this paper, we claim that as soon as the public, key-independent, part f_{pub} of f is collision-resistant (which is the case for our OPRF based on Legendre symbols), the OPRF becomes verifiable for free. Then $f_{\mathsf{pub}}(K)$ can be used as a public tag for the server with key K, and a user can reject the OPRF output if the secure function evaluation protocol does not output the expected tag. Collision resistance of $f_{\mathsf{pub}}(K)$ means that no two keys can have the same tag.

Functionality $\mathcal{F}^{f}_{\mathbf{SFE}}$

Parse $f := (f_{\mathsf{pub}}, f_{\mathsf{sec}})$.

Input:

On $(\textsc{UserInput}, sid, h)$ from U with $h \in \mathcal{I}$, store $\langle sid, U, h\rangle$. Finally, send $(\textsc{UserInput}, sid, U)$ to $\mathcal{A}^*$

On $(\textsc{ServerInput}, sid, K)$ from S with $K \in \mathcal{K}$, store $\langle sid, S, K\rangle$. Finally, send $(\textsc{ServerInput}, sid, S, f_{\mathsf{pub}}(K))$ to $\mathcal{A}^*$

Output:

On $(\textsc{Output}, sid)$ from $\mathcal{A}^*$, retrieve $\langle sid, U, h\rangle$ and $\langle sid, S, K\rangle$ and output $(sid, f_{\mathsf{sec}}(h, K))$ to U.

Fig. 1: Ideal secure function evaluation functionality $\mathcal{F}^{f}_{\mathrm{SFE}}$, parameterized by a function $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}}) : \mathcal{I} \times \mathcal{K} \to \mathcal{Y}$, where f_{pub} is modeling information leaked about the server's input.

by splitting up the OfflineEval and RcvComplete interfaces into their honest and malicious versions. We start by describing how the functionality works. $\mathcal{F}_{\mathrm{OPRF}}$ models the interaction of multiple users with a single server. The functionality is initialized by a server S sending an Initmessage. This is then the unique server used for this session and the functionality stores a table $F_{\mathrm{honest}}()$ of truly random values representing the server's OPRF output values. The server can always query his "own" PRF through OfflineEval, while the adversary can query all other function tables $F_{\mathrm{malicious}}(\cdot, K^*)$ through OfflineEval by specifying an adversarial key K^*. The functionality models adaptive compromise, which allows the adversary (with the environment's permission) to send a Compromisemessage and then arbitrarily query the server's table $F_{\mathrm{honest}}()$ through OfflineEval.

Finally, there are the *Online Evaluation* interfaces, which allow the user and server together to run the OPRF protocol in some subsession *ssid*. It requires the user to send an Evalmessage with its input and the server to send a SndrCompletemessage. The adversary can then send a RcvCompletemessage to cause the session to complete. The adversary learns when the parties send their messages and, in the RcvCompletemessage, can specify an alternative table $F_{\mathrm{malicious}}(\cdot, K^*)$ from which the user will receive its output. This models the network adversary participating in the role of the server using its own key K^*.

The counter for the server *tx* ensures that the number of times the server's table is evaluated is limited by the number of times the server participated in the protocol. Each time the server sends a SndrCompletemessage, it is incremented, and each time the OPRF of the server is evaluated, it is again decremented. Furthermore, the functionality guarantees random outputs even

Ideal OPRF functionality

The OPRF functionality is parameterized by a public PRF output length λ and a function $f : \mathcal{I} \times \mathcal{K} \to \mathcal{Y}$. It maintains functions H_1, $T_{\text{preview}}(\cdot,\cdot)$, $F_{\text{honest}}(\cdot), F_{\text{malicious}}(\cdot)$ initially undefined everywhere, an initially empty set $T_{\text{programmed}}$ and an initially empty list $\mathcal{K}$ of adversarial keys. The first time an undefined value $F_{\text{honest}}(x)$, $F_{\text{malicious}}(x, K)$, or $T_{\text{preview}}(x, i)$ is referenced, $\mathcal{F}^{f}_{\text{2H-OPRF}}$ chooses $r \leftarrow\$ \{0,1\}^{\lambda}$ and sets $F_i(x) := r$, or $T_{\text{preview}}(x, i) := r$. Similarly, if $\mathsf{H}_1(x)$ is referenced for the first time $\mathcal{F}^{f}_{\text{2H-OPRF}}$ chooses $h \leftarrow\$ \mathcal{I}$ and sets $\mathsf{H}_1(x) := r$.

Initialization:
On message $(\textsc{Init}, sid)$ from party S, if this is the first $\textsc{Init}$ message for sid send $(\textsc{Init}, sid, S)$ to $\mathcal{A}^*$. From now on use the tag S to denote the unique entity which sent the $\textsc{Init}$ message for the session identifier sid. (Ignore all subsequent $\textsc{Init}$ messages for sid.)

Server compromise:
On message $(\textsc{Compromise}, sid)$ from $\mathcal{A}^*$, declare S as $\textsc{Compromised}$.
Note: Message $(\textsc{Compromise}, sid)$ requires permission from the environment. //If S is corrupted, then it is declared $\textsc{Compromised}$ as well.

Offline evaluation of honest function:
On $(\textsc{OfflineEval}, sid, ssid, x)$ from $P \in \{S, \mathcal{A}^*\}$, ignore if $P = \mathcal{A}^*$ and S is not $\textsc{Compromised}$. Otherwise, send $(\textsc{OfflineEval}, sid, ssid, F_{\text{honest}}(x))$ to P.

Offline evaluation of adversarial functions:
On $(\textsc{OfflineEval}, sid, ssid, K^*, x)$ from $\mathcal{A}^*$, run $\textsc{Correlate}(K^*)$, and send $(\textsc{OfflineEval}, sid, ssid, F_{\text{malicious}}(x, K^*))$ to $\mathcal{A}^*$.

Online evaluation:
- On $(\textsc{Eval}, sid, ssid, S', x)$ from $P \in \{U, \mathcal{A}^*\}$, send $(\textsc{Eval}, sid, ssid, P, S')$ to $\mathcal{A}^*$. Record $\langle ssid, P, x\rangle$
- On $(\textsc{SndrComplete}, sid, ssid')$ from S, send $(\textsc{SndrComplete}, sid, ssid', S)$ to $\mathcal{A}^*$, record $\langle S, ssid\rangle$.
- On $(\textsc{RcvCompleteHonest}, sid, ssid, P)$ from $\mathcal{A}^*$, ignore this message if there is no record $\langle ssid, P, x\rangle$ stored. Else:
  - If S is not $\textsc{Compromised}$, ignore this message if there is no record $\langle S, ssid\rangle$
  - Send $(\textsc{Eval}, sid, ssid, F_{\text{honest}}(x))$ to P.
- On $(\textsc{RcvCompleteMalicious}, sid, ssid, P, K^*)$ from $\mathcal{A}^*$, with $K^* \in \mathcal{K}$, ignore this message if there is no record $\langle ssid, P, x\rangle$ stored. Otherwise, run $\textsc{Correlate}(K^*)$ and send $(\textsc{Eval}, sid, ssid, F_{\text{malicious}}(x, K^*))$ to P.

Random oracles:
On (H_1, sid, x) from $\mathcal{A}^*$, reply with $(\mathsf{H}_1, sid, \mathsf{H}_1(x))$.
On $(\mathsf{H}_2, sid, x, y^*)$ from $\mathcal{A}^*$:
- For the oldest element K^* in $\mathcal{K}$ that satisfies $f(\mathsf{H}_1(x), K^*) = y^*$, set $t \leftarrow F_{\text{malicious}}(x, K^*)$.
- If there is no such element, set $t \leftarrow T_{\text{preview}}(x, y^*)$
- Send (H_2, sid, t) to P

Procedure $\textsc{Correlate}(K^*)$: //Assign previewed values to the $F_{K^*}()$ function table
- Append K^* to $\mathcal{K}$.
- For all $(x, y) \in \mathsf{dom}(T_{\text{preview}}) \setminus T_{\text{programmed}}$ with $f(\mathsf{H}_1(x), K^*) = y$:
  - Set $F_{\text{malicious}}(K^*, x) := T_{\text{preview}}(x, y)$
  - Add (x, y) to $T_{\text{programmed}}$

Fig. 2: Ideal OPRF functionalities: $\mathcal{F}_{\text{OPRF}}$ [29] (without gray parts) and $\mathcal{F}^{f}_{\text{2H-OPRF}}$ (including gray parts).

in the case of malicious servers and requires that the user's input and output remain hidden from both the server and the adversary.

In Figure 2, by including the gray parts, we define a functionality $\mathcal{F}^{f}_{\text{2H-OPRF}}$ that is slightly weaker than $\mathcal{F}_{\text{OPRF}}$ but more closely captures the guarantees of 2Hash OPRFs. We stress that $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is a *dedicated tool* for analyzing 2Hash OPRFs, which we reflect in the functionality's name. In a nutshell, $\mathcal{F}^{f}_{\text{2H-OPRF}}$ captures that 2Hash-PRF, i.e., functions of the form $\mathsf{H}_2(x, f(\mathsf{H}_1(x), K)$ can be computed not only from inputs x, K, but also from x, y where $y = f(\mathsf{H}_1(x), K)$. $\mathcal{F}^{f}_{\text{2H-OPRF}}$ hence incorporates adversarial evaluation interfaces for such function evaluation from pairs x, y. To ensure consistency, $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is parametrized with a function f and maintains the oracles $\mathsf{H}_1, \mathsf{H}_2$, to automatically "correlate" adversarial function tables $F_{\text{malicious}}(\cdot, K^*)$ with evaluations x, y if $y = f(\mathsf{H}_1(x), K^*)$. We describe these adversarial evaluation interfaces of $\mathcal{F}^{f}_{\text{2H-OPRF}}$, which are the only interfaces that are added on top of $\mathcal{F}_{\text{OPRF}}$, more formally below.

- $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is parametrized by a function $f : \mathcal{I} \times \mathcal{K} \to \mathcal{Y}$ that takes as input a key and a PRF input. f is the core function of the 2Hash PRF.
- $\mathcal{F}^{f}_{\text{2H-OPRF}}$ maintains a random oracle H_1, the "inner" random oracle of a 2Hash PRF. It is implemented as a truly random function.
- $\mathcal{F}^{f}_{\text{2H-OPRF}}$ maintains a random oracle H_2, the "outer" random oracle of a 2Hash PRF. The oracle is queried with inputs (x, y). If y is generated from some adversarial key K^*, i.e., $f(\mathsf{H}_1(x), K^*) = y$, then $\mathcal{F}^{f}_{\text{2H-OPRF}}$ "programs" $F_{\text{malicious}}(x, K^*)$ into $\mathsf{H}_2(x, y)$. Otherwise, $\mathcal{F}^{f}_{\text{2H-OPRF}}$ samples a uniform value r stored as $T_{\text{preview}}(x, y)$ and replies with that value.
- Whenever the adversary introduces a new adversarial key K^* into $\mathcal{F}_{\text{OPRF}}$ (i.e., via OfflineEval or RcvCompleteMalicious), $\mathcal{F}^{f}_{\text{2H-OPRF}}$ runs procedure Correlate(K^*) that iterates through the still unassigned previewed tuples (x, y) in $T_{\text{preview}} \setminus T_{\text{programmed}}$. For each such tuple that satisfies $y = f(\mathsf{H}_1(x), K^*)$, $\mathcal{F}^{f}_{\text{2H-OPRF}}$ sets $F_{\text{malicious}}(x, K^*) = T_{\text{preview}}(x, y)$, i.e., it "programs" $F_{\text{malicious}}(x, K^*)$ to the previewed value r. Once assigned, a tuple is added to $T_{\text{programmed}}$, i.e., each previewed value can only be programmed into at most one adversarial function.

We stress that these additional evaluation interfaces of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ *never leak any information about the honest function* table $F_{\text{honest}}()$, because they give out random values that are only ever written into adversarial function tables $F_{\text{malicious}}()$ by the Correlate function. Our definition also takes inspiration from "lazy extraction" relaxations [1], because it dispenses with the need to online-extract PRF keys from adversarial function evaluations, allowing for more efficient OPRF constructions. We now formalize the intuition about $\mathcal{F}^{f}_{\text{2H-OPRF}}$ being almost as secure as $\mathcal{F}_{\text{OPRF}}$: we show that $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is as strong as $\mathcal{F}_{\text{OPRF}}$ already if it is efficiently decidable whether two pairs $(x, y), (x', y')$ were generated from the same key.

Lemma 1. *Relative to a random oracle H_1^* and an associated oracle $\mathcal{O}_{\mathsf{key-eq}}$ that decides whether for two pairs $(x, y), (x', y')$, there exists a key K such that $f(H_1^*(x), K) = y$ and $f(H_1^*(x'), K) = y'$, $\mathcal{F}^f_{2H\text{-}OPRF}$ UC-emulates $\mathcal{F}_{OPRF}$.*

Proof. We give a simulator Sim interacting with $\mathcal{F}_{\text{OPRF}}$ that emulates the adversarial evaluation interfaces of $\mathcal{F}^f_{\text{2H-OPRF}}$. The idea is that, instead of extracting and submitting adversarial keys to $\mathcal{F}_{\text{OPRF}}$, if Sim encounters a pair (x^*, y^*) for which it does not know a key yet, it simply uses $x^*||y^*$ as an *identifier* of the unknown key. To ensure consistency, Sim uses his oracle to identify if further pairs (x', y') are generated from the same (unknown) key, and consequently uses identifier $x^*||y^*$ for those. If the adversary later reveals a key K^* through OfflineEval or RcvComplete for which $f(\mathsf{H}_1(x^*), K^*) = y^*$, Sim again uses $x^*||y^*$ when relaying the queries to $\mathcal{F}_{\text{OPRF}}$. Overall, the key-equality oracle gives Sim a heads-up over $\mathcal{F}^f_{\text{2H-OPRF}}$: through the oracle, Sim already knows which adversarial key table a tuple x, y is eventually correlated with, and hence the adversarial tables in $\mathcal{F}_{\text{OPRF}}$ are always already consistent with whatever the Correlate function will eventually do. More formally, Sim maintains initially empty lists $\mathcal{K}$ and $\mathcal{I}$. Sim relays all queries between $\mathcal{A}^*$ and $\mathcal{F}_{\text{OPRF}}$ unchanged except for the following:

- Sim answers queries (H_1, sid, x) using H_1^*
- On $(\text{OfflineEval}, sid, ssid, K^*, x)$ from $\mathcal{A}^*$, Sim does:
  - If $\exists x||y \in \mathcal{I}$ such that $f(\mathsf{H}_1(x), K^*) = y$, send $(\text{OfflineEval}, sid, ssid, x||y, x)$ to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$
  - Otherwise, append K^* to $\mathcal{K}$, relay the query to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$.
- On $(\text{RcvCompleteMalicious}, sid, ssid, P, K^*)$ from $\mathcal{A}^*$, Sim does:
  - If $\exists x||y \in \mathcal{I}$ s.t. $f(\mathsf{H}_1(x), K^*) = y$, send $(\text{RcvCompleteMalicious}, sid, ssid, P, x||y)$ to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$
  - Otherwise, append K^* to $\mathcal{K}$, relay the query to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$.
- On $(\mathsf{H}_2, sid, x^*, y^*)$ from $\mathcal{A}^*$, Sim does:
  - If $\exists x||y \in \mathcal{I}$ such that $\mathcal{O}_{\mathsf{key-eq}}(x, y, x^*, y^*) = 1$, send $(\text{OfflineEval}, sid, ssid, x||y, x^*)$ to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$
  - If no such entry is found, for the oldest $K^* \in \mathcal{K}$ with $f(\mathsf{H}_1(x^*), K^*) = y^*$, send $(\text{OfflineEval}, sid, ssid, K^*, x^*)$ to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$
  - If no such key is found, append $x^*||y^*$ to $\mathcal{I}$, send $(\text{OfflineEval}, sid, ssid, x^*||y^*, x^*)$ to $\mathcal{F}_{\text{OPRF}}$ and relay the response to $\mathcal{A}^*$.

Because H_1 is perfectly simulated and the other three interfaces do the same as in $\mathcal{F}^f_{\text{2H-OPRF}}$ in case $K^* \in \mathcal{K}$ is found, the only difference between the real and the ideal world occurs whenever a tuple x, y is submitted to any of them *before* the corresponding key was revealed. We hence analyze the case where x, y is submitted to H_2 where the key is unknown. In the real world, $\mathcal{F}^f_{\text{2H-OPRF}}$ replies with a fresh random value to this query. The same happens in the ideal world, where

$(\textsc{OfflineEval}, sid, ssid, x^*, x^*||y^*)$ and $(\textsc{OfflineEval}, sid, ssid, x^*, x||y)$ queries to $\mathcal{F}_{\text{OPRF}}$ both result in a fresh random value. It is left to argue that, whenever the $\textsc{Correlate}$ function programs a preview $T_{\text{preview}}(x, y)$ into table $F_{\text{malicious}}(\cdot, K^*)$, query $(\mathsf{H}_2, sid, x, y)$ was answered from the unique table in $\mathcal{F}_{\text{OPRF}}$ that is identified with some $x^*||y^*$ such that $f(\mathsf{H}_1(x^*), K^*) = y^*$. This holds because, upon $(\mathsf{H}_2, sid, x, y)$, Sim invokes the key-equality oracle, which ensures the correct routing of the H_2 query to the unique table in $\mathcal{F}_{\text{OPRF}}$ which generates the output for tuples generated from K^*.

For some keyed functions f, Lemma 1 can be used to show that $\mathcal{F}^f_{\text{2H-OPRF}}$ is just as strong as $\mathcal{F}_{\text{OPRF}}$, as shown in the following corollary.

Corollary 1. *Let G be a prime-order group and let f be the keyed function*

$$f : G \setminus \{1_G\} \times \mathbb{F}_p^\times \to G \setminus \{1_G\} : f(x, k) := x^k$$

used by 2HashDH. Then $\mathcal{F}^f_{\text{2H-OPRF}}$ UC-emulates $\mathcal{F}_{\text{OPRF}}$.

Proof. This is because for $f(x, k) := x^k$ it is possible to efficiently instantiate the oracles H_1^* and $\mathcal{O}_{\mathsf{key-eq}}$: The oracle $H_1^* : \{0,1\}^* \to G \setminus \{1_G\}$ can be instantiated by maintaining a table of pairs $\{(x, a_x)\}$, where $a_x \in \mathbb{F}_p^\times$ is chosen uniformly at random for each $x \in \{0,1\}^*$ and answering a query $H_1^*(x)$ with g^{a_x}. Then, queries $((x, y), (x', y'))$ to $\mathcal{O}_{\mathsf{key-eq}}$ should be answered positively if and only if $y \neq 1_G$ and $y' \neq 1_G$ are valid group elements and $y^{1/a_x} = y'^{1/a_{x'}}$, which can be decided efficiently.

We argue heuristically using Lemma 1 that, if f is weakly key-collision resistant, then (non-pathological) protocols that use $\mathcal{F}_{\text{OPRF}}$ can be securely instantiated from $\mathcal{F}^f_{\text{2H-OPRF}}$ instead. Heuristically, the oracles H_1^* and $\mathcal{O}_{\mathsf{key-eq}}$ are not useful for an adversary. This is because the weak key-collision resistance implies that $(x, y = f(H_1^*(x), K))$ is a commitment to the key K, and if we model this as an ideal commitment scheme then it would not be possible for the adversary to come up with valid commitments for unknown keys, so the $\mathcal{O}_{\mathsf{key-eq}}$ oracle would be useless: Either (x, y) or (x', y') is an invalid commitment and then the $\mathcal{O}_{\mathsf{key-eq}}$-oracle outputs 0, or the adversary already knows the underlying keys and could check if they are the same without help from the oracle. If the oracles are useless, then any secure protocol in the $\mathcal{F}_{\text{OPRF}}$-hybrid model stays secure in the $\mathcal{F}_{\text{OPRF}}, H_1^*, \mathcal{O}_{\mathsf{key-eq}}$-hybrid model. Lemma 1 says that it is then also secure in the $\mathcal{F}^f_{\text{2H-OPRF}}, H_1^*, \mathcal{O}_{\mathsf{key-eq}}$-hybrid model (and hence also in the $\mathcal{F}^f_{\text{2H-OPRF}}$-hybrid model).

However, we stress that we are not aware of any $\mathcal{F}_{\text{OPRF}}$ application in the literature that cannot be proven to be secure using $\mathcal{F}^f_{\text{2H-OPRF}}$ as well, so there is no need to rely on this heuristic argument. Below we give some applications of OPRFs that can be instantiated from $\mathcal{F}^f_{\text{2H-OPRF}}$ directly.

3.1 Applications of $\mathcal{F}^f_{\text{2H-OPRF}}$

As explained above, $\mathcal{F}^f_{\text{2H-OPRF}}$ is almost as strong as $\mathcal{F}_{\text{OPRF}}$, and in particular, it provides the same protection as $\mathcal{F}_{\text{OPRF}}$ when it comes to the honest server's random function. *We are not aware of any application of $\mathcal{F}_{OPRF}$ that does not work with $\mathcal{F}^f_{2H\text{-}OPRF}$ instead.* To argue this more formally, we first state a lower bound on the security guarantees of $\mathcal{F}^f_{\text{2H-OPRF}}$.

Lemma 2. *$\mathcal{F}^f_{2H\text{-}OPRF}$ UC-emulates $\mathcal{F}_{corOPRF}$.*

The proof can be found in Appendix A, where we also depict $\mathcal{F}_{\text{corOPRF}}$. By the UC composition theorem, we can hence replace $\mathcal{F}_{\text{corOPRF}}$ by $\mathcal{F}^f_{\text{2H-OPRF}}$ in applications. In particular, $\mathcal{F}^f_{\text{2H-OPRF}}$ (and its realizations provided in this work) can be plugged into OPAQUE, which can be instantiated from $\mathcal{F}_{\text{corOPRF}}$ [32].

However, Jarecki et al. [32] show that $\mathcal{F}_{\text{corOPRF}}$ is not sufficient to instantiate *password-protected secret sharing* (PPSS) or threshold OPAQUE. The reason is that $\mathcal{F}_{\text{corOPRF}}$ allows the adversary to correlate adversarial function tables with the honest function, and hence $\mathcal{F}_{\text{corOPRF}}$ leaks information about the inputs of honest sessions to the adversary. Jarecki et al. argue that this allows servers to verify guesses of the client's inputs online, which is particularly devastating when OPRFs are used on low-entropy inputs such as passwords (e.g., in PPSS). Their argument about OPAQUE working with $\mathcal{F}_{\text{corOPRF}}$ is that OPAQUE anyway allows the server to verify online guesses.

We claim that $\mathcal{F}^f_{\text{2H-OPRF}}$ can be used to instantiate, e.g., the PPSS scheme of Jarecki et al. [29]. Intuitively, this is because $\mathcal{F}^f_{\text{2H-OPRF}}$ does not allow the adversary to learn anything about honest evaluations, i.e., it perfectly protects the honest function table $F_{\text{honest}}()$. A bit more formally, we can adapt their simulator ([29], Figure 9) to export $\mathcal{F}^f_{\text{2H-OPRF}}$'s adversarial function evaluation interfaces from which it cannot extract a key to the PPSS simulator. We can essentially treat such an adversarial function evaluation the same way as step 9), case 1. Here, they are replied to using freshly chosen random values, and the PPSS simulator does not require knowledge of the key behind such evaluations beyond that.

3.2 Generic compiler: 2Hash OPRFs from secure evaluations of f

In this section, we have so far presented and discussed $\mathcal{F}^f_{\text{2H-OPRF}}$, a dedicated functionality for assessing the security of 2Hash OPRFs. We now move on to building such protocols. While the "insecurity" of $\mathcal{F}^f_{\text{2H-OPRF}}$ is mostly insignificant for applications as discussed above, it allows us to construct *practically efficient* OPRFs from presumably quantum-safe building blocks from secure multi-party computation (MPC). MPC yields methods for securely evaluating any function. In particular, we can attempt to build 2Hash OPRFs from MPC as follows:

1. The user computes $\mathsf{H}_1(x)$
2. User (holding H_1) and server (holding K) engage in a 2-party secure computation of f, letting the user compute $y := f(\mathsf{H}_1(x), K)$. Formally, we capture this step by functionality $\mathcal{F}^f_{\text{SFE}}$ (Figure 1).
3. The user computes and outputs $\mathsf{H}_2(x, y)$.

We now investigate whether this approach formalized in Figure 3, can yield a $\mathcal{F}^f_{\text{2H-OPRF}}$-secure OPRF. We start by defining two properties of families of functions.

– *(n, q)-One-More Unpredictability.* Let $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}}) \leftarrow\$ \mathcal{F}$. The probability that an efficient adversary gets n random inputs $x_1, \ldots, x_n$ together with some information $f_{\mathsf{pub}}(K)$ about the key then queries a $f_{\mathsf{sec}}(\cdot, K)$ oracle q times and queries its verification oracle $\mathcal{V}^f_K(\cdot, \dot{)}$ such that it outputs q+1-times 1 on distinct inputs is negligible, where $\mathcal{V}^f_K(i^*, y^*)$ outputs 1 if $f_{\mathsf{sec}}(x_{i^*}, K) = y^*$ and 0 else. More precisely, for every PPT $\mathcal{A}$ it holds that

$$\mathbf{Adv}^{\text{OMU}}_{\mathcal{F},\mathcal{A}}(n, q) := \Pr\left[(i_k, y_k)_{k\in[q+1]} \leftarrow \mathcal{A}^{f_{\mathsf{sec}}(\cdot,K),\mathcal{V}^f_K(\cdot,\cdot)}((x_k)_{k\in[n]}, f, f_{\mathsf{pub}}(K))\right] \le \eta,$$

where $\mathcal{V}^f_K(i_k, y_k) = 1 \forall k \in [q + 1]$, η is negligible in λ, and the probability is taken over $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}}) \leftarrow\$ \mathcal{F}$, $K \leftarrow\$ \mathcal{K}$, $(x_1, \ldots, x_n) \leftarrow\$ \mathcal{I}^n$ and the random coins of $\mathcal{A}$.

– *n-Weak Key-Collision Resistance.* For uniformly random $x_1, \ldots, x_n \in \mathcal{I}$ the probability that an adversary finds two different keys that map one of the x_i to the same output is negligible. More precisely, for every PPT $\mathcal{A}$ we have that

$$\mathbf{Adv}^{\text{WKCR}}_{\mathcal{F},\mathcal{A}}(n) := \Pr\left[\begin{matrix} f_{\mathsf{sec}}(x_i, K) = f_{\mathsf{sec}}(x_i, K') \\ \wedge K \neq K' \end{matrix} : (K, K', i) \leftarrow \mathcal{A}((x_k)_{k\in[n]}, f)\right] \le \eta,$$

where η is negligible in λ and the probability is taken over $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}}) \leftarrow\$ \mathcal{F}$, $(x_1, \ldots, x_n) \leftarrow\$ \mathcal{I}^n$ and the random coins of $\mathcal{A}$.

Theorem 1. *$\Pi^{\mathsf{H}_1,\mathsf{H}_2,f}_{OPRF}$ UC-realizes $\mathcal{F}^f_{2H\text{-}OPRF}$ in the $\mathcal{F}^f_{SFE}$-hybrid model if f is uniformly chosen from a one-more unpredictable and weakly collision-resistant function family with $|\mathcal{I}| = 2^{\Omega(\lambda)}$, and $\mathsf{H}_1, \mathsf{H}_2$ are modeled as random oracles.*

More precisely, for every efficient real-world adversary $\mathcal{A}^*$ against $\Pi^{\mathsf{H}_1,\mathsf{H}_2,f}_{OPRF}$ the ideal-world execution with $\mathcal{F}^f_{2H\text{-}OPRF}$ and the simulator Sim *from Fig. 10 is such that for every efficient environment $\mathcal{Z}$ there exists an efficient adversary $\mathcal{B}$ against the weak key-collision resistance of f and an efficient adversary $\mathcal{B}'$ against the one-more unpredictability of f such that*

$$\mathbf{Dist}^{\Pi^{\mathsf{H}_1,\mathsf{H}_2,f}_{OPRF}, \{\mathcal{F}^f_{2H\text{-}OPRF}, \mathsf{Sim}\}}_{\mathcal{Z}}(\lambda) \le \mathbf{Adv}^{\text{WKCR}}_{\mathcal{F},\mathcal{B}}(n_{\mathsf{H}_1}) + \mathbf{Adv}^{\text{OMU}}_{\mathcal{F},\mathcal{B}'}(n_{\mathsf{H}_1}, n_S) + \frac{n_{\mathsf{H}_1}}{|\mathcal{I}|},$$

Compiler $\Pi_{\mathbf{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f}$

Public Parameters: function $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}}) \leftarrow\!\$\ \mathcal{F}$, output length λ.
Initialization:
On input (Init, sid): S picks $K \leftarrow\!\$\ \mathbb{Z}_p$ and stores $\langle sid, K\rangle$

Server Compromise:
On $(\text{Compromise}, sid)$, if there is a record $\langle sid, K\rangle$ reveal K to $\mathcal{A}^*$

Offline Evaluation:
On $(\text{OfflineEval}, sid, x)$, the server retrieves $\langle sid, K\rangle$ and outputs $(\text{OfflineEval}, sid, \mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K)))$

Online Evaluation:
- On $(\text{Eval}, sid, ssid, S', x)$, U sends $(\text{UserInput}, ssid, \mathsf{H}_1(x))$ to $\mathcal{F}_{\text{SFE}}^{f}$.
- On $(\text{SndrComplete}, sid, ssid')$, S sends $(\text{ServerInput}, ssid, K)$ to $\mathcal{F}_{\text{SFE}}^{f}$.
- On receiving $(ssid, y)$ from $\mathcal{F}_{\text{SFE}}^{f}$, U outputs $(\text{Eval}, sid, ssid, \mathsf{H}_2(x, y))$.

Fig. 3: The compiler $\Pi_{\text{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f}$ in the $\mathcal{F}_{\text{SFE}}^{f}$-hybrid model, with hash functions $\mathsf{H}_1, \mathsf{H}_2$.

where n_{H_1} is the number of H_1 queries that $\mathcal{Z}$ makes and n_S is the number of SndrComplete *messages that $\mathcal{Z}$ sends.*

Proof sketch. When we realize $\mathcal{F}_{\text{2H-OPRF}}^{f}$ by an oblivious evaluation of $\text{PRF}_K(x) := \mathsf{H}_2(x, f(\mathsf{H}_1(x), K)$ through a 2-party computation of $f()$, the client obviously learns *more* about K than just $\text{PRF}_K(x)$. In the proof we need to argue that the intermediate values $f(*, K)$ do not help in breaking the strong pseudorandomness guarantees of $\mathcal{F}_{\text{2H-OPRF}}^{f}$, namely that the honest server's PRF remains pseudorandom on every still unqueried input. This is ensured by drawing f uniformly from a one-more unpredictable function family: despite having seen multiple tuples $(x_1, f(\mathsf{H}_1(x_1), K)), ..., (x_q, f(\mathsf{H}_1(x_q), K))$, where K is the simulated key of the honest server, the adversary is not able to come up with *another* tuple $(x_{q+1}, f(\mathsf{H}_1(x_{q+1}), K))$.

Another imperfection of $\text{PRF}_K(x) := \mathsf{H}_2(x, f(\mathsf{H}_1(x), K)$ is that there could exist key collisions, i.e., two different keys K, K' such that for some input x it holds that $f(\mathsf{H}_1(x), K) = f(\mathsf{H}_1(x), K')$. With such a collision, a user who evaluates x twice, once with a server using K and once with a server using K', would compute the same output in both evaluations. This is not considered a secure OPRF, because the function's outputs can then signal equality of user inputs. Indeed, neither $\mathcal{F}_{\text{OPRF}}$ nor $\mathcal{F}_{\text{2H-OPRF}}^{f}$ signal equality of user inputs, even when facing a malicious server that potentially found a key collision. To prove that an oblivious evaluation of $\text{PRF}_K(x) := \mathsf{H}_2(x, f(\mathsf{H}_1(x), K))$ UC-emulates $\mathcal{F}_{\text{2H-OPRF}}^{f}$, we hence need to rule out that the adversary finds key collisions for f, which is captured by drawing f from a family of key-collision resistant functions.

Since $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is already case-tailored for evaluating the security of oblivious evaluations of functions of the form $\mathrm{PRF}_K(x) := \mathsf{H}_2(x, f(\mathsf{H}_1(x), K)$, the work of the simulator is actually quite minimal: most queries are simply relayed between the adversary and $\mathcal{F}^{f}_{\text{2H-OPRF}}$. For example, answers to random oracles $\mathsf{H}_1, \mathsf{H}_2$ queries are not chosen by Sim but by $\mathcal{F}^{f}_{\text{2H-OPRF}}$, and the programming part of these oracles is already hard-coded in the corresponding $\mathcal{F}^{f}_{\text{2H-OPRF}}$ interfaces through the CORRELATE procedure. Furthermore, using $\mathcal{F}_{\text{SFE}}$ as a building block, online extraction from corrupted users and server is immediate. The main complexity of the formal argument hence lies in verifying that the "programming" of previewed values (i.e., values given out by the H_2 interface) by the CORRELATE procedure of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ consistently reflects the real function tables $\mathrm{PRF}_K(\cdot)$ for all adversarial keys K. We defer the full proof to Appendix B.

4 Instantiation from AES

To instantiate Theorem 1, we need to (a) choose a one-more unpredictable and key-collision resistant function family, and (b) instantiate $\mathcal{F}^{f}_{\text{SFE}}$ for a randomly sampled f from that family. As a warm-up, we showcase a potential instantiation from a block cipher. For this, we first give a function family $\mathcal{F}_{\mathsf{E}}$, based on a block cipher E, and show that this function family is one-more unpredictable if E is a PRF and key-collision resistant if E is modeled as an ideal cipher. Together with *Theorem* 1 and any maliciously secure multi-party computation for binary circuits, which realizes $\mathcal{F}^{f_{\mathsf{E}}}_{\text{SFE}}$ for $f_{\mathsf{E}} \in \mathcal{F}_{\mathsf{E}}$, yields a $\mathcal{F}^{f}_{\text{2H-OPRF}}$-secure OPRF.

Let $\mathsf{E} : \{0,1\}^{2\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ be a block cipher.

$$\begin{aligned} f_{\mathsf{E}} : \{0,1\}^{2\lambda} \times \{0,1\}^{\lambda} &\to \{0,1\}^{2\lambda} \\ (h, K) &\mapsto (\mathsf{E}_K(h)), \end{aligned}$$

We define

$$\mathcal{F}_{\mathsf{E}} \coloneqq \{(f_{\mathsf{pub}} = \perp, f_{\mathsf{sec}} = f_{\mathsf{E}})\}.$$

By setting $f_{\mathsf{pub}} = \perp$ we denote that the corresponding protocol $\mathcal{F}^{f_{\mathsf{E}}}_{\text{SFE}}$ will have no leakage on the key.

Lemma 3. *If E is modeled as a PRP then $\mathcal{F}_{\mathsf{E}}$ is (n, q)-one-more unpredictable. More concretely, if $\mathcal{A}$ is an adversary for the one-more unpredictability game that makes Qv queries to the verification oracle, then there exists an adversary against the PRP property of E with roughly the same running time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\text{OMU}}_{\mathcal{F},\mathcal{A}}(n, q) \leq \mathbf{Adv}^{\text{PRP}}_{\mathsf{E},\mathcal{B}}(q_v + q) + \frac{n(n-1)}{2^{2\lambda+1}} + \frac{q_V}{2^{2\lambda} - q_V - q}$$

Proof. Suppose the unpredictability game implements $\mathsf{E}_K(\cdot) = \mathcal{O}_R(\cdot)$ as a truly random permutation. We first consider the event that there is a repeated value in the list $x_1, \ldots, x_n$ produced by the one-more unpredictability game. Since uniformly random $x, x' \leftarrow\!\$ \{0,1\}^{2\lambda}$ are equal with probability $2^{-2\lambda}$. It follows from a union bound over all $0 < i < j \leq n$ that the probability that there is a collision is bounded by $n(n-1)/2^{2\lambda+1}$. If there are no collisions, and $\mathcal{A}$ made one more successful $\mathcal{V}_K^f(i, y)$ query than $f_{\mathsf{E}}(x, K)$ queries then the adversary needs to have guessed at least one $\mathsf{E}_K(x_i)$ value. Let q_V be the number of queries to the $\mathcal{V}_K^f$ oracle and q the number of queries to $f_{\mathsf{E}}(\cdot, K)$ that $\mathcal{A}$ makes. (Note that $q_V \geq q + 1$ as there might be queries on which $\mathcal{V}_K^f$ outputs 0.)

The probability of guessing the value of $\mathsf{E}_K(x_i)$ in one attempt is bounded by

$$\frac{1}{2^{2\lambda} - q_V - q},$$

because $\mathsf{E}_K(x_i)$ could take any value in $\{0,1\}^{2\lambda}$ except at most the q_V values that were already tried, and the q values that were output by the oracle $f_{\mathsf{E}}(\cdot, K)$ (as there are no collisions in a permutation). Then, a union bound over all q_V guesses of $\mathcal{A}$ says that the winning probability of $\mathcal{A}^*$ is at most

$$\frac{n(n-1)}{2^{2\lambda+1}} + \frac{q_V}{2^{2\lambda} - q_V - q}, \tag{1}$$

when $\mathsf{E}_K(\cdot)$ is implemented as a random oracle.

Now let $\mathcal{B}$ be an adversary for the PRP game that simulates the unpredictability game for $\mathcal{A}^*$ and outputs 1 if $\mathcal{A}^*$ wins. If $\mathcal{B}$ is interacting with $\mathsf{E}_K(\cdot)$, then it outputs 1 with probability $\mathbf{Adv}^{\mathrm{OMU}}_{\mathcal{F},\mathcal{A}}(n, q)$ otherwise, we have proven that it outputs 1 with probability at most (1), so the inequality in the theorem statement follows.

□

Lemma 4. *If E is modeled as an ideal cipher then $\mathcal{F}_{\mathsf{E}}$ is key-collision resistant, and hence also (n)-weak key collision resistant for any n. More precisely, let $\mathcal{A}$ be an adversary that makes Q queries to the ideal cipher, then the probability that $\mathcal{A}$ outputs a key-collision is at most*

$$\frac{(Q+2)(Q+1)}{2(2^{2\lambda} - Q - 1)}.$$

Proof. We can assume without generatlity (but increasing Q by 2) that before outputting a key collision x_i, K, K' with $\mathsf{E}_K(x_i) = \mathsf{E}_{K'}(x_i) = y$ the adversary has queried $\mathsf{E}_K(x_i)$ and $\mathsf{E}_{K'}(x_i)$.

Let $(K_i, x_i, y_i = \mathsf{E}_{K_i}(x_i))$ for $i \in [Q+2]$ be the i-th input-output pair learned by the adversary, either through an encryption query on (K_i, x_i) or a decryption query on (K_i, y_i).

For any $i < j$ the probability that the i-th and the j-th query resulted in the first key collision is at most $1/(2^{2\lambda} - Q - 1)$ because if the j-th query was an encryption query then we can only get a collision if $K_i \neq K_j$ and $x_i = x_j$ and if the $\mathsf{E}_{K_j}(x_j)$ query resulted in $y_j = \mathsf{E}_{K_i}(x_i)$. This happens with probability at most $1/(2^{2\lambda} - Q - 1)$, because at most $Q+1$ outputs of the random permutation E_K have been fixed already, and the remaining $2^{2\lambda} - Q - 1$ outputs are all equaly likely. Similarly, if the j-th query was a decryption query, then to get a fresh collision we need $K_i \neq K_j$ and $y_j = y_i$, and the query must have resulted in $x_j = x_i$, which is one out of at most $2^{2\lambda} - Q - 1$ still available preimages of the random permutation E_K.

Now the theorem statement follows from a union bound over all $1 \leq i < j \leq Q + 2$.

5 Instantiation from OT and Legendre Symbols

To instantiate Theorem 1, we need to (a) choose a one-more unpredictable and key-collision resistant function family, and (b) instantiate $\mathcal{F}^f_{\mathrm{SFE}}$ for a randomly sampled f from that family. We start with (a) below, followed by (b) in Section 5.1.

We define a function f_{LSeq} for creating a particular sequence of Legendre symbols. f_{LSeq} is parameterized by public vectors $\mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}$ and $\mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$ and defined as follows:

$$\begin{aligned} f^{\mathbf{l},\mathbf{l}'}_{\mathsf{LSeq}} &: \mathbb{F}_p \times \mathbb{F}_p \to \{-1, 0, 1\}^{\ell_{\mathsf{com}} + \ell_{\mathsf{eval}}} \\ (h, K) &\mapsto \left(\left\{ \left(\frac{K + l_i}{p} \right) \right\}_{i \in [\ell_{\mathsf{com}}]}, \left\{ \left(\frac{h + K + l'_i}{p} \right) \right\}_{i \in [\ell_{\mathsf{eval}}]} \right) \end{aligned}$$

We then set $f_{\mathsf{pub}}(K)^{\mathbf{l}} := \left(\left(\frac{K+l_1}{p}\right), \ldots, \left(\frac{K+l_{\ell_{\mathsf{com}}}}{p}\right) \right)$. We define a family of functions as

$$\mathcal{F}_{\mathsf{LSeq}} := \{ (f_{\mathsf{pub}}, f_{\mathsf{sec}} = f^{\mathbf{l},\mathbf{l}'}_{\mathsf{LSeq}} \mid \mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}, \mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}} \}.$$

For convenience we will usually write f_{LSeq} instead of $f^{\mathbf{l},\mathbf{l}'}_{\mathsf{LSeq}}$.

Before we prove unpredictability, we first prove a supporting lemma. Note that f_{LSeq} is defined in a bit-wise manner. There can be inputs $x \neq x'$ such that the s-th bit of $f_{\mathsf{LSeq}}(x, K)$ is the same Legendre symbol as the t-th bit of $f_{\mathsf{LSeq}}(x', K)$. Therefore, we must ensure that the adversary can not learn too much about the bits of the sequences $f_{\mathsf{LSeq}}(x_i, K)$ by querying its oracle $f_{\mathsf{LSeq}}(\cdot, K)$ on other inputs. Concretely, we bound in the next lemma the probability that with one query to $f_{\mathsf{LSeq}}(\cdot, K)$ the adversary can learn information about more than one sequence $f_{\mathsf{LSeq}}(x_i, K)$.

Lemma 5. *Let* $(x_1,\dots,x_n) \leftarrow\!\$\ \mathbb{Z}_p$ *and* $(l_1,\dots,l_{\ell_{\mathsf{eval}}}) \leftarrow\!\$\ \mathbb{Z}_p$. *Let* $p_{\text{2SeqOverlap}}$ *be the probability that there exists an* x^* *and indices* $i \neq j$ *such that* $f_{\mathsf{LSeq}}(x^*, K)$ *overlaps with both the sequence* $f_{\mathsf{LSeq}}(x_i, K)$ *and with the sequence* $f_{\mathsf{LSeq}}(x_j, K)$. *We have*

$$p_{\text{2SeqOverlap}} \leq \frac{n^2 \cdot (\ell_{\mathsf{eval}}+1)^4}{8p}.$$

Proof. We are interested in tuples $(x^*, i, j, s, t, u, v) \in \mathbb{Z}_p \times [n]^2 \times [\ell_{\mathsf{eval}}]^4$ such that

$$(x^* + l_s = x_i + l_t) \wedge (x^* + l_u = x_j + l_v) \tag{2}$$

holds. We have that

$$\Pr_{x_i, x_j, l_s, l_t, l_u, l_v \leftarrow\$\mathbb{Z}_p}[(x^* + l_s = x_i + l_t) \wedge (x^* + l_u = x_j + l_v)] \leq \frac{1}{p^2}.$$

If (x^*, i, j, s, t, u, v) is such that Eq. (2) holds then

– (x^*, j, i, u, v, s, t) also satisfies Eq. (2), so we can always arrange $i < j$,
– $(x^* - l_v + l_s, i, j, v, t, u, s)$ also satisfies Eq. (2), so we can arrange $s \leq v$, and
– $(x^* - l_t + l_u, i, j, s, u, t, v)$ also satisfies Eq. (2), so we can arrange $t \leq u$.

Therefore, when we take a union bound over all tuples, it is sufficient to only consider tuples with $i < j$, $s \leq v$ and $t \leq u$, of which there are

$$p \cdot \frac{n(n-1)}{2} \cdot \left(\frac{\ell_{\mathsf{eval}}(\ell_{\mathsf{eval}}+1)}{2}\right)^2 < \frac{p \cdot n^2(\ell_{\mathsf{eval}}+1)^4}{8}.$$

So, the union bound yields

$$p_{\text{2SeqOverlap}} \leq \frac{n^2 \cdot (\ell_{\mathsf{eval}}+1)^4}{8p}.$$

□

Lemma 6. *The family of Legendre symbol sequences* $\mathcal{F}_{\mathsf{LSeq}}$ *as defined in Section 2 is* (n, q)*-one-more unpredictable under the DSLS assumption. More concretely, let* $\mathcal{A}$ *be an adversary for the* (n, q)*-one-more-unpredictability of* $\mathcal{F}_{\mathsf{LSeq}}$ *which makes* q_v *queries to the verification oracle. Then there exists an adversary* $\mathcal{B}$*, with roughly the same running time as* $\mathcal{A}$*, for the DSLS game such that*

$$\mathbf{Adv}^{\text{OMU}}_{\mathcal{F}_{\mathsf{LSeq}},\mathcal{A}}(n, q) \leq \mathbf{Adv}^{\text{DSLS}}_{p,\mathcal{B}}(1^\lambda) + (n^2 \cdot {\ell_{\mathsf{eval}}}^4)/8p + q_v 2^{-\ell_{\mathsf{eval}}}.$$

Proof. $\mathcal{B}$ chooses inputs $x_1,\dots,x_n \leftarrow\!\$\ \mathbb{Z}_p$ and public parameters $\mathbf{l} \leftarrow\!\$\ \mathbb{F}_p^{\ell_{\mathsf{com}}}, \mathbf{l}' \leftarrow\!\$\ \mathbb{F}_p^{\ell_{\mathsf{eval}}}$. Next, $\mathcal{B}$ queries $l_1 \dots, l_{\ell_{\mathsf{com}}}$ to its real-or-random oracle $\mathcal{O}_R$ to obtain $f_{\mathsf{pub}}(K)$. Then, $\mathcal{B}$ internally runs $\mathcal{A}$ on these inputs and answers all queries that $\mathcal{A}$ does to its oracles $f_{\mathsf{LSeq}}(\cdot, K)$ and $\mathcal{V}^f_K(\cdot,\cdot)$ by forwarding the individual Legendre symbols to its oracle $\mathcal{O}_R$. More precisely, on a $f_{\mathsf{LSeq}}(\cdot, K)$-query x by

$\mathcal{A}$, $\mathcal{B}$ sends the queries $x + l'_1, \dots, x + l'_{\ell_{\mathsf{eval}}}$ to $\mathcal{O}_R$. On receiving the outputs $o_1, \dots, o_{\ell_{\mathsf{eval}}}$ from $\mathcal{O}_R$, $\mathcal{B}$ gives $y \coloneqq (o_1, \dots, o_{\ell_{\mathsf{eval}}})$ to $\mathcal{A}$. Similarly, when $\mathcal{A}$ queries its $\mathcal{V}^f_K$ on input (i, y) then $\mathcal{B}$ sends the queries $x_i + l'_1, \dots, x_i + l'_{\ell_{\mathsf{eval}}}$ to $\mathcal{O}_R$. On receiving the outputs $o_1, \dots, o_{\ell_{\mathsf{eval}}}$ from $\mathcal{O}_R$, $\mathcal{B}$ checks if $y = o_1, \dots, o_{\ell_{\mathsf{eval}}}$ and answers accordingly to $\mathcal{A}$. Finally, if $\mathcal{A}$ makes $q+1$ distinct queries where the check $y = o_1, \dots, o_{\ell_{\mathsf{eval}}}$ holds, $\mathcal{B}$ outputs 1. When $\mathcal{A}$ terminates and made less than $q+1$ verifying queries then $\mathcal{B}$ outputs 0.

Lemma 5 says that the adversary can learn information about at most sequence $f_{\mathsf{LSeq}}(x_i, K)$ for each query to $f_{\mathsf{LSeq}}(\cdot, K)$ except with probability $(n^2 \cdot {\ell_{\mathsf{eval}}}^4)/8p$. That means, with overwhelming probability, the adversary can obtain with q different bits of at most q Legendre sequences. But then, there must be an index i^* such that $\mathcal{A}$ has no information about any of the bits of $f_{\mathsf{LSeq}}(x_{i^*}, K)$. Note that if $\mathcal{O}_R$ is implemented as a truly random function then $\mathcal{A}$ can guess $\mathcal{O}_R(x_{i^*} + l'_1), \dots, \mathcal{O}_R(x_{i^*} + l'_{\ell_{\mathsf{eval}}})$ for a new x_{i^*} at most with probability $q_v 2^{-\ell_{\mathsf{eval}}}$. (Predicting the ℓ_{com} first bits is trivial.)

In contrast, if $\mathcal{O}_R$ is implemented as $\left(\frac{\cdot + K}{p}\right)$ for some uniformly random $K \in \mathbb{Z}_p$ then the view of $\mathcal{A}$ is distributed exactly as in the unpredictability game. Thus, $\mathcal{A}$'s prediction is right with probability ϵ. The inequality from the theorem statement follows. □

Before we argue that $\mathcal{F}_{\mathsf{LSeq}}$ has key-collision resistance in Lemma 8, we first show a supporting lemma about Legendre symbols.

Lemma 7. *Let $p > 2$ be a prime and $K, K' \in \mathbb{Z}_p$ with $K \neq K'$. Then the number of $x \in \mathbb{Z}_p$ such that $\left(\frac{K+x}{p}\right) = \left(\frac{K'+x}{p}\right)$ is exactly $\frac{p-3}{2}$.*

Proof. $\left(\frac{K+x}{p}\right) = \left(\frac{K'+x}{p}\right)$ means that $(K+x)(K'+x)$ is a nonzero square, and curves of the form $(K+x)(K'+x) = y^2$ have $p+1$ projective points, of which $p-1$ are affine points. Two of these points are with $y = 0$, and the remaining $p-3$ come in pairs (x, y),$(x, -y)$. Each pair corresponds to an x such that $\left(\frac{K+x}{p}\right) = \left(\frac{K'+x}{p}\right)$. So the total number of such pairs is $(p-3)/2$. □

Lemma 8. *The family of Legendre symbol sequences $\mathcal{F}_{\mathsf{LSeq}}$ as defined in Section 2 is weakly key-collision resistant. More precisely, for any adversary $\mathcal{A}$ we have $\mathbf{Adv}^{\mathrm{WKCR}}_{\mathcal{F}_{\mathsf{LSeq}}, \mathcal{A}}(n) \leq np^2 2^{-\ell_{\mathsf{com}} - \ell_{\mathsf{eval}} - 1}$.*

Proof. A key collision corresponds to a tuple (i, K, K') with $K \neq K'$ and such that for all $t \in [\ell_{\mathsf{com}}]$ and $t' \in [\ell_{\mathsf{eval}}]$ it holds that

$$\left(\frac{l_t + K}{p}\right) = \left(\frac{l_t + K'}{p}\right) \text{ and } \left(\frac{x_i + l'_t + K}{p}\right) = \left(\frac{x_i + l'_t + K'}{p}\right). \tag{3}$$

Because of Lemma 7, for every tuple (i, K, K') with $K \neq K'$ the probability over the random choice of the offsets $\mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}$ and $\mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$ that (i, K, K')

corresponds to a collision is at most

$$((p-3)/2p)^{\ell_{\mathsf{com}}+\ell_{\mathsf{eval}}} \leq 2^{-\ell_{\mathsf{com}}-\ell_{\mathsf{eval}}} .$$

Now by a union bound over all the $np(p-1)/2$ tuples, we get that even a computationally unbounded adversary can find a collision with probability at most $np^2 2^{-\ell_{\mathsf{com}}-\ell_{\mathsf{eval}}-1}$. $\square$

5.1 $\mathcal{F}_{\mathbf{SFE}}^{f_{\mathsf{LSeq}}}$ instantiation from VOLE$^+$ and ZK proofs.

We now instantiate the $\mathcal{F}_{\mathrm{SFE}}$ functionality from Figure 1 for $f = f_{\mathsf{LSeq}}$, i.e., functionality $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$.

VOLE and VOLE$^+$. The VOLE functionality lets a server input two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^{\ell}$, and the client input a field element h. The server does not learn anything, and the client learns $\mathbf{o} = \mathbf{u} + h \cdot \mathbf{v}$. As explained in Section 1.2, our protocol uses an extended VOLE functionality which we call VOLE$^+$. This functionality, in addition to $\mathbf{o}$, also outputs a hashing key $\boldsymbol{\gamma}$ and two check values $c_{\mathbf{u}} = \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r_{\mathbf{u}}$ and $c_{\mathbf{v}} = \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$, where $r_{\mathbf{u}}$ and $r_{\mathbf{v}}$ are inputs from the server.

In Appendix D we give a protocol that realizes the $\mathcal{F}_{\mathrm{VOLE}^+}$ functionality in the $\mathcal{F}_{\mathrm{sVOLE}}$-hybrid model. We give an overview of the protocol here, and we refer to Appendix D for the details of the protocol and the security proof. The protocol uses the $\mathcal{F}_{\mathrm{sVOLE}}$ (subset vole) functionality, which is parameterized by an arbitrary non-empty subset $S_{\Delta} \subset \mathbb{F}_p$ and outputs a random vole correlation $(\Delta, \mathbf{o}), (\mathbf{u}, \mathbf{v})$ such that $\mathbf{o} = \mathbf{u} + \Delta \cdot \mathbf{v}$, where Δ is sampled from S_{Δ}. The functionality is endemic, meaning that corrupted parties are allowed to choose their outputs [39]. We describe the functionality in Fig. 12. The OT-based protocol of [42] instantiates this functionality with a constant number of rounds and an amount of communication that is independent of n and p, but with a runtime linear in $|S_{\Delta}|$, which means the protocol is only efficient if the subset S_{Δ} is small.

Our protocol for the $\mathcal{F}_{\mathrm{VOLE}^+}$ functionality is similar to the protocol of [42,8]. It first uses the subset vole functionality to generate k random subset vole correlations $\mathbf{o}_i = \mathbf{u}_i + \Delta_i \mathbf{v}_i$, where $\Delta_i \in S_{\Delta}$. Then, the correlations are derandomized so all the $\mathbf{v}_i$ become equal to $\mathbf{v}_1$. After the derandomization, we can take a random linear combination of the k subset VOLE correlations with coefficients $\boldsymbol{\lambda} \in \mathbb{F}_p^k$ to produce a single "full" VOLE correlation $\mathbf{o} = \mathbf{u}' + \Delta \mathbf{v}_1$, where $\mathbf{u}' = \sum_{i=1}^k \lambda_i \mathbf{u}_i$ and $\Delta = \sum_{i=1}^k \lambda_i \Delta_i$. Finally, this correlation is derandomized to make Δ match the receiver's input h, and to make $\mathbf{u}', \mathbf{v}_1$ match the sender's input $\mathbf{u}, \mathbf{v}$.

For malicious security, it is important to check that the sender really derandomizes the $\mathbf{v}_i$ so that they all become the same value. To do this check the receiver generates a challenge $\boldsymbol{\gamma} \in \mathbb{F}_p^l$ and asks the sender to send $c_{\mathbf{u}_i} = \langle \boldsymbol{\gamma}, \mathbf{u}_i \rangle$ for all $i \in [k]$ and $c_{\mathbf{v}_1} = \langle \boldsymbol{\gamma}, \mathbf{v}_1 \rangle$. The receiver checks if $\langle \boldsymbol{\gamma}, \mathbf{o}_i \rangle = c_{\mathbf{u}_i} + \Delta_i \cdot c_{\mathbf{v}_1}$ for all $i \in [k]$. In our protocol, we let the receiver output $\boldsymbol{\gamma}$ and the check value $c_{\mathbf{v}}$

(after the derandomization), so we essentially get the "$^+$"-part of the VOLE$^+$ functionality for free.

However, the probabilistic checks allow for a selective failure attack. If a misbehaving sender derandomizes $\mathbf{v}_i$ incorrectly, he can still send $c_{\mathbf{u}_i}$, so that the $\langle \boldsymbol{\gamma}, \mathbf{o}_i \rangle = c_{\mathbf{u}_i} + \Delta_i \cdot c_{\mathbf{v}_1}$ check holds for one of the values in S_Δ. This way, if the receiver does not abort (which happens with probability at most $1/|S_\Delta|$), then the sender knows the value of Δ_i. The use-cases of [42,8] can tolerate this kind of selective failure attack, so they did not try to prevent this attack. In contrast, for our use case, a selective failure attack against the VOLE$^+$ functionality would translate to a selective failure attack against the OPRF, which breaks the UC security. So we need to fix the problem. The solution is to slightly increase the number of subset vole correlations, so that even if the sender learns a limited number of the Δ_i values, the remaining Δ_i's still have enough entropy, to argue security with the leftover hash lemma. Concretely, this means that we need $k \geq (\log p + 2s)/\log(|S_\Delta|)$ to get s bits of statistical security.

In Appendix D we prove the following theorem.

Theorem 2. *The protocol $\Pi^{p,\ell,k}_{VOLE^+}$ of Fig. 13 UC-realizes $\mathcal{F}_{VOLE+}$ in the $\mathcal{F}_{sVOLE}$-hybrid model if $k > (\log p + 2s)/\log |S_\Delta|$, where s is a statistical security parameter, assuming secure and authenticated channels. More precisely, for every adversary there is an efficient simulator such that the view of $\mathcal{Z}$ in the ideal world is statistically close to its view in the real world with statistical distance bounded by $2^{-s} + \binom{k}{2}p^{-1}$.*

Zero-knowledge proofs. We define a ideal zero-knowledge functionality in Fig. 5. The functionality $\mathcal{F}^{p,n}_{\mathrm{ZK}}$ first lets a prover input a witness $\mathbf{w} \in \mathbb{F}_p^n$. Then it lets the prover and verifier input a set of multivariate degree-d polynomials $f_1, \ldots, f_t \in \mathbb{F}_p[x_1, \ldots, x_n]_{\leq d}$, and the functionality will only output $\top$ to the verifier if the witness $\mathbf{w}$ satisfies $f_i(\mathbf{w}) = 0$ for all $i \in [t]$. Note that the polynomial constraints can be chosen after the prover has committed to the witness $\mathbf{w}$, which is a property we rely upon. Our functionality differs from the functionality of Yang, Sarkar, Weng, and Wang [46] only because we fix the length n of the witness $\mathbf{w}$, and we require that the prover inputs the entire witness $\mathbf{w} \in \mathbb{F}_p^n$ at once. In contrast, the functionality of [46] allows the witness to be arbitrarily long, and it can be extended dynamically. Since our functionality is more restrictive, the Quicksilver protocol of [46] securely realizes our $\mathcal{F}_{\mathrm{ZK}}$ functionality.

Description of the protocol and security analysis. Figure 6 gives a description of the protocol. The rationale behind the protocol was given in Section 1.2.

Theorem 3. *Let $f^{\mathbf{l},\mathbf{l}'}_{\mathsf{LSeq}} \xleftarrow{\$} \mathcal{F}_{\mathsf{LSeq}}$. The protocol $\Pi^{f_{\mathsf{LSeq}}}_{SFE}$ from Fig. 6 UC-realizes $\mathcal{F}^{f_{\mathsf{LSeq}}}_{SFE}$ in the $\mathcal{F}_{VOLE+}, \mathcal{F}_{\mathsf{ZK}}$-hybrid model.*

Proof sketch. For a *malicious client*, the simulator extracts the client's input h from its input to $\mathcal{F}_{\mathrm{VOLE+}}$. The simulator then submits h to $\mathcal{F}_{\mathrm{SFE}}$ on behalf

Functionality $\mathcal{F}^{p,n}_{\mathbf{VOLE+}}$

Honest user input:
On input $(\textsc{UserInput}, sid, h)$ from U with $h \in \mathbb{F}_p$, sample $\boldsymbol{\gamma} \leftarrow \mathbb{F}_p^n$ and store $\langle \textsc{UserInput}, sid, h, \boldsymbol{\gamma} \rangle$ and send $(\textsc{UserInput}, sid, \boldsymbol{\gamma}, U)$ to $\mathcal{A}^*$.

Malicious user input:
On input $(\textsc{UserInput}, sid, h, \boldsymbol{\gamma})$ from $\mathcal{A}^*$ with $h \in \mathbb{F}_p$ and $\boldsymbol{\gamma} \in \mathbb{F}_p^n$, ignore if U is not corrupt. Otherwise, store $\langle \textsc{UserInput}, sid, h, \boldsymbol{\gamma} \rangle$.

Server input:
On input $(\textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}})$ from S with $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^n$ and $r_{\mathbf{u}}, r_{\mathbf{v}} \in \mathbb{F}_p$ store $\langle \textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}} \rangle$ and send $(\textsc{ServerInput}, sid, U)$ to $\mathcal{A}^*$.

Output:
On input $(\textsc{Output}, sid, P)$ from $\mathcal{A}^*$ with $P \in \{\mathcal{A}^*, U, S\}$, retrieve $\langle \textsc{UserInput}, sid, h, \boldsymbol{\gamma} \rangle$, $\langle \textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}} \rangle$ and do:
- Compute $\mathbf{o} \leftarrow \mathbf{u} + h\mathbf{v}$, $c_{\mathbf{u}} = \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r_{\mathbf{u}}$, and $c_{\mathbf{v}} = \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$
- If $P = \mathcal{A}^*$, output $(\textsc{Output}, sid, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$ to $\mathcal{A}^*$
- If $P = U$, output $(\textsc{Output}, sid, \mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$ to U
- If $P = S$, output $(\textsc{Output}, sid, \boldsymbol{\gamma})$ to S.

Fig. 4: Ideal functionality $\mathcal{F}^{p,n}_{\mathrm{VOLE+}}$, for a prime p and integer n.

Functionality $\mathcal{F}^{p,n}_{\mathbf{ZK}}$

Input witness:
On input $(\textsc{VfInput}, sid)$ from $\mathcal{V}$ store $\langle \textsc{VfInput}, sid, \mathcal{V} \rangle$ and send $(\textsc{VfInput}, sid, \mathcal{V})$ to $\mathcal{A}^*$.

On input $(\textsc{PrvInput}, sid, \mathbf{w})$ from $\mathcal{P}$ with $\mathbf{w} \in \mathbb{F}_p^n$, store $\langle \textsc{PrvInput}, sid, \mathcal{P}, \mathbf{w} \rangle$ and send $(\textsc{PrvInput}, sid, \mathcal{P})$ to $\mathcal{A}^*$.

On input $(\textsc{Committed}, sid)$ from $\mathcal{A}^*$, if there are records $\langle \textsc{VfInput}, sid, \mathcal{V} \rangle$ and $\langle \textsc{PrvInput}, sid, \mathcal{P}, \mathbf{w} \rangle$, send $(\textsc{Committed}, sid)$ to $\mathcal{V}$ and $\mathcal{P}$.

Prove polynomial constraints:
On input $(\textsc{Prove}, sid, \{f_i\}_{i \in [t]})$ from $\mathcal{V}$ store $\left\langle \textsc{Prove}, sid, \mathcal{V}, \{f_i\}_{i \in [t]} \right\rangle$ and send $(\textsc{Prove}, sid, \{f_i\}_{i \in [t]})$ to $\mathcal{A}^*$.

On input $(\textsc{Prove}, sid, \{f'_i\}_{i \in [t']})$ from $\mathcal{P}$ store $\left\langle \textsc{Prove}, sid, \mathcal{P}, \{f'_i\}_{i \in [t']} \right\rangle$ and send $(\textsc{Prove}, sid, \{f'_i\}_{i \in [t']})$ to $\mathcal{A}^*$.

On input $(\textsc{Output}, sid)$ from $\mathcal{A}^*$, retrieve $\langle \textsc{PrvInput}, sid, \mathcal{P}, \mathbf{w} \rangle$, $\left\langle \textsc{Prove}, sid, \mathcal{V}, \{f_i\}_{i \in [t]} \right\rangle$, and $\left\langle \textsc{Prove}, sid, \mathcal{P}, \{f'_i\}_{i \in [t']} \right\rangle$. If $\{f_i\}_{i \in [t]} = \{f'_i\}_{i \in [t']}$, and if $f_i(\mathbf{w}) = 0$ for all $i \in [t]$ then output $(sid, \top)$ to U.

Fig. 5: Ideal functionality $\mathcal{F}^{p,n}_{\mathrm{ZK}}$, for a prime p and integer n.

$\Pi_{\text{SFE}}^{f_{\mathsf{LSeq}}}$

Client$(h \in \mathbb{F}_p)$:

Server$(K \in \mathbb{F}_p)$:

$\mathbf{a} \leftarrow\!\!\$ \ (\mathbb{F}_p^{\times})^{\ell_{\mathsf{eval}}}$

$\mathbf{v} \leftarrow \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)$

$\mathbf{u} \leftarrow (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{v}$

$r_{\mathbf{u}}, r_{\mathbf{v}} \leftarrow\!\!\$ \ \mathbb{F}_p$

$\mathbf{e} \leftarrow \left\{ \left(\frac{K + l_i}{p} \right) \right\}_{i \in [\ell_{\mathsf{com}}]}$

Compute $\mathbf{s} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}$ such that $\forall i \in [\ell_{\mathsf{com}}]$:

$(K + l_i) * s_i^2 = E(e_i)$

$\mathbf{w} \leftarrow (K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}})$

$$\begin{tikzcd}
{} & \boxed{\mathcal{F}^{p,1+\ell}_{\text{ZK}}} \arrow[l, "\textsc{Committed}"'] & {} \arrow[l, "\mathbf{w}"'] \\
{} \arrow[r, "h"] & \boxed{\mathcal{F}^{p,\ell_{\mathsf{eval}}}_{\text{VOLE}+}} \arrow[l, "\mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}}"'] \arrow[r, "\boldsymbol{\gamma}"] & {} \arrow[l, "\mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}"'] \\
{} & & {} \arrow[ll, "\mathbf{e}"']
\end{tikzcd}$$

$c_{\mathbf{u}} \leftarrow \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r_{\mathbf{u}}$

$c_{\mathbf{v}} \leftarrow \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$

Both parties locally compute the constraints $\mathcal{F}$:

$f_u(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \boldsymbol{\gamma}, (X \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$

$f_v(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \boldsymbol{\gamma}, \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$

$f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i) Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\mathsf{com}}]$

$\mathcal{F} \leftarrow \{ f_u, f_v, f_e^{(1)}, \ldots, f_e^{(\ell_{\mathsf{com}})} \}$

$$\begin{tikzcd}
{} \arrow[r, "\mathcal{F}"] & \boxed{\mathcal{F}^{p,1+\ell}_{\text{ZK}}} \arrow[dl, "\mathsf{status}"'] & {} \arrow[l, "\mathcal{F}"'] \\
{} & &
\end{tikzcd}$$

Abort if two or more entries of $\mathbf{o}$ are zero. //Protect against $a_i = 0$

$\mathbf{e}' \leftarrow \left\{ \left(\frac{o_i}{p} \right) \right\}_{i \in [\ell_{\mathsf{eval}}]}$

If $e_i = 0$ for some $i \in [\ell_{\mathsf{com}}]$, abort if $f_{\mathsf{LSeq}}(h, -l_i) \neq (\mathbf{e}, \mathbf{e}')$. //Protect against $s_i = 0$

output: $(\mathbf{e}, \mathbf{e}')$

Fig. 6: Secure evaluation of f_{LSeq} for parameters $p, \ell_{\mathsf{com}}, \ell_{\mathsf{eval}} > 2$ and public offsets $\mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}, \mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$. For readability, we simplify the messages to and from the $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{VOLE}+}$ functionalities, e.g., omitting interface names and session id's. If an expected message does not arrive, parties do not continue, e.g., U only inputs h into $\mathcal{F}_{\text{VOLE}+}$ after receiving the Committed output of $\mathcal{F}_{\text{ZK}}$.

of the corrupt client, to obtain $(\mathbf{e}, \mathbf{e}') = f_{\mathsf{LSeq}}(h, K)$ for K being the input of the honest server (unknown to the simulator). Next to $\mathbf{e}$, the only other values seen by the corrupt client are $\mathbf{o}, c_{\mathbf{u}}, c_{\mathbf{v}}$. $c_{\mathbf{u}}, c_{\mathbf{v}}$ do not depend on K through the additive randomization terms $r_{\mathbf{u}}, r_{\mathbf{v}}$ and can hence be chosen at random. Because of the blinding with a square, the only information in $\mathbf{o}$ about the key is $\left(\frac{o_i}{p}\right) = \left(\frac{K+l_i}{p}\right)$ which is however equal to $\left(\frac{e_i}{p}\right)$ and hence the simulator can choose random $\mathbf{o}_i$ among all values with the appropriate Legendre symbol over p. Since the server does not produce any output, this concludes the simulation sketch for the malicious client.

Malicious Server: During an execution of the real protocol, the environment $\mathcal{Z}$ sees only input confirmation notifications from $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{VOLE+}}$ and the $\mathcal{F}_{\text{VOLE+}}$ leakage which is a uniformly random vector $\boldsymbol{\gamma} \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$. These messages are simulated perfectly by Sim, so before seeing the user's output the view of $\mathcal{Z}$ in the real world follows exactly the same distribution as its view in the ideal world. The crucial part of the proof is to ensure that the client's output is indistinguishable, i.e., the simulator is able to detect, with overwhelming probability, whenever the honest client would abort in the real execution. We let the simulator trigger an output at the client as follows:

– Sim sends $(\textsc{Output}, ssid, b)$ to $\mathcal{F}_{\text{SFE}}^{f_{\mathsf{LSeq}}}$ only if

$$a_i \neq 0 \text{ for all } i \in [\ell_{\mathsf{eval}}]\,, \tag{4}$$

$$e_i = \left(\frac{K + l_i}{p}\right) \text{ for all } i \in [\ell_{\mathsf{com}}] \tag{5}$$

$$(K + l_i)s_i^2 = E(e_i) \text{ for all } i \in [\ell_{\mathsf{com}}]\,, \tag{6}$$

$$\mathbf{v} = \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,, \tag{7}$$

$$\mathbf{u} = (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,, \tag{8}$$

$$r_{\mathbf{u}} = r'_{\mathbf{u}} \text{ and } r_{\mathbf{v}} = r'_{\mathbf{v}}\,, \tag{9}$$

$$\mathcal{F} = \{f_u, f_v, f_e^{(1)}, \ldots, f_e^{(\ell_{\mathsf{com}})}\}\,. \tag{10}$$

Conditioned on $\mathcal{A}^*$ sending $(K, \mathbf{a}, \mathbf{s}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}, \mathbf{e})$ on behalf of the corrupt server, and Sim choosing $\boldsymbol{\gamma}$, we have in the ideal world:

– If the conditions (4) to (10) are satisfied then Sim sends $(\textsc{Output}, ssid, b)$ to $\mathcal{F}_{\text{SFE}}^{f_{\mathsf{LSeq}}}$ which outputs $f(h, K)$,

– otherwise $\mathcal{F}_{\text{SFE}}^{f_{\mathsf{LSeq}}}$ does not output anything.

While in the real world conditioned on $\mathcal{A}^*$ sending $(K, \mathbf{a}, \mathbf{s}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}, \mathbf{e})$ and $\mathcal{F}_{\text{VOLE+}}$ outputting $\boldsymbol{\gamma}$ we have the following cases:

– If the conditions (4) to (10) are satisfied then the honest user output $f(h, K)$ with probability 1. Indeed, the user receives $\mathbf{o} = \mathbf{u} + h\mathbf{v}$ from $\mathcal{F}_{\text{VOLE+}}$, so we have $o_i = (K + h + l'_i)(\mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2))_i$ for all $i \in [\ell_{\mathsf{eval}}]$, which means that $\left(\frac{o_i}{p}\right) = \left(\frac{K+h+l'_i}{p}\right)$ because all the a_i are non-zero (4). Because of (5) we also have $\mathbf{e}_i = \left(\frac{K+l_i}{p}\right)$, so the $(\mathbf{e}, \mathbf{e}') = f(h, K)$ check passes in case $e_i = 0$ for some $i \in [\ell_{\mathsf{com}}]$. The conditions $(6) - (10)$ imply that the polynomial constraints are satisfied and so $\mathcal{F}_{\mathrm{ZK}}$ is guaranteed to output $\top$. Moreover, since all the offsets l'_i are distinct, at most one of the entries in $\mathbf{e}'$ can be zero, so the honest user does not abort and outputs $(\mathbf{e}, \mathbf{e}') = f(h, K)$.

- If the condition (4) is not satisfied, then the honest user aborts with probability 1. Because if $a_i = 0$ for some i, then o_i and o_{i+1} will both be zero, which causes the honest user to abort.
- If the conditions (5) and (6) are not both satisfied either one of the $f_e^{(i)}$ polynomials does not evaluate to zero, so $\mathcal{F}_{\mathrm{ZK}}$ outputs $\perp$, or $s_i = 0$ and $(K + l_i) \neq 0$, which causes the $(\mathbf{e}, \mathbf{e}') = f(h, -l_i)$ check to fail. Either way the honest user aborts.
- If (10) is not satisfied $\mathcal{F}_{\mathrm{ZK}}$ outputs $\perp$, and the honest user aborts.
- If $\langle \gamma, \mathbf{u} - (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\rangle \neq r_{\mathbf{u}} - r'_{\mathbf{u}}$ or $\langle \gamma, \mathbf{v} - \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\rangle \neq r_{\mathbf{v}} - r'_{\mathbf{v}}$ then f_u or f_v does not evaluate to zero and $\mathcal{F}_{\mathrm{ZK}}$ outputs $\perp$, and the honest user aborts.

In all the above cases, the user output in the ideal world is the same as in the real world. The only cases that is not yet considered is if $(3), (4)$ or (5) is not satisfied but still

$$\langle \boldsymbol{\gamma}, \mathbf{u} - (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\rangle = r_{\mathbf{u}} - r'_{\mathbf{u}}\,, \text{ and} \tag{11}$$

$$\langle \boldsymbol{\gamma}, \mathbf{v} - \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\rangle = r_{\mathbf{v}} - r'_{\mathbf{v}}\,. \tag{12}$$

This case occurs with probability at most $1/p$ because $\boldsymbol{\gamma}$ is chosen uniformly at random after $K, \mathbf{a}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r'_{\mathbf{u}}, r_{\mathbf{v}}, r'_{\mathbf{v}}$ are determined. If (3) does not hold then the left-hand side of Eq. (12) is uniformly random, which means the equation holds with probability at most $1/p$. Similarly, if (4) does not hold then Eq. (11) holds only with probability $1/p$. Finally, if (3) and (4) hold but (5) does not hold, then the left-hand sides are zero, but one of the right-hand sides is nonzero, so the equations are not both satisfied. We conclude that the distinguishing advantage of $\mathcal{Z}$ is bounded by $1/p$.

We depict the simulator in Figure 11 and the full hybrid argument in Appendix C.

Combining Theorem 1, Theorem 3, Lemma 6 and Lemma 8, using the composition theorem of the UC framework, we get the following result.

Corollary 2. *Let* $f_{\mathsf{LSeq}}^{\mathbf{l},\mathbf{l}'} : \mathbb{F}_p \times \mathbb{F}_p \to \{-1, 0, 1\}^{\ell_{\mathsf{com}} + \ell_{\mathsf{eval}}}$ *sampled uniformly at random from* $\mathcal{F}_{\mathsf{LSeq}}$ *as in Section 5. Then protocol* $\Pi_{OPRF}^{\mathsf{H}_1, \mathsf{H}_2, f_{\mathsf{LSeq}}}$*, with calls to* $\mathcal{F}_{SFE}^{f_{\mathsf{LSeq}}}$ *replaced by the protocol from Figure 6, securely realizes* $\mathcal{F}_{2H\text{-}OPRF}^{f}$ *in the* $\{\mathcal{F}_{VOLE+}^{p,\ell_{\mathsf{eval}}}, \mathcal{F}_{ZK}\}$*-hybrid model under the DSLS assumption, with* $\mathsf{H}_1, \mathsf{H}_2$ *modeled as random oracles.*

Let n_{H_1} be the number of H_1 queries that $\mathcal{Z}$ makes and n_S be the number of SNDRCOMPLETE *messages that $\mathcal{Z}$ sends. Then, for every efficient real-world adversary $\mathcal{A}^*$ against $\Pi_{OPRF}^{\mathsf{H}_1,\mathsf{H}_2,f_{\mathsf{LSeq}}}$ the ideal-world execution with $\mathcal{F}_{2H\text{-}OPRF}^{f}$ there exists a simulator* Sim *such that for every efficient environment $\mathcal{Z}$ there exists an efficient DSLS adversary $\mathcal{B}$ such that*

$$\mathbf{Dist}_{\mathcal{Z}}^{\Pi_{OPRF}^{\mathsf{H}_1,\mathsf{H}_2,f_{\mathsf{LSeq}}},\{\mathcal{F}_{2H\text{-}OPRF}^{f},\mathsf{Sim}\}}(\lambda) \leq \frac{n_{\mathsf{H}_1}p^2}{2^{-\ell_{\mathsf{com}}-\ell_{\mathsf{eval}}-1}} + \mathbf{Adv}_{p,\mathcal{B}}^{\mathrm{DSLS}}(1^\lambda)$$
$$+ \frac{n_{\mathsf{H}_1}^2 \cdot \ell_{\mathsf{eval}}{}^4}{8p} + \frac{n_{\mathsf{H}_2}}{2^{\ell_{\mathsf{eval}}}} + \frac{n_{\mathsf{H}_1}}{p}.$$

To obtain a concrete OPRF protocol from the above corollary, we instantiate the $\mathcal{F}_{\mathrm{ZK}}$ functionality with the Quicksilver protocol [46], and we instantiate the $\mathcal{F}_{\mathrm{VOLE+}}^{p,\ell_{\mathsf{eval}}}$ functionality with our Π_{VOLE^+} protocol from Appendix D, which in turn relies on the subset VOLE functionality $\mathcal{F}_{\mathrm{sVOLE}}$, of which there exist efficient post-quantum realizations from oblivious transfer [42]. We now give estimates on the efficiency of the resulting OPRF.

5.2 Parameters and concrete efficiency.

We instantiate the 2Hash OPRF framework with the function family $\mathcal{F}_{\mathsf{LSeq}}$ aiming for 128 bits of security. We take the following parameters:

- We set $\ell_{\mathsf{eval}} = 128$, since that is the minimum required to achieve unpredictability of $\mathcal{F}_{\mathsf{LSeq}}$.
- The bitsize of the prime p needs to be large enough to ensure the (n, q)-one-more unpredictability of $\mathcal{F}_{\mathsf{LSeq}}$. Lemma 6 reduces the unpredictability of $\mathcal{F}_{\mathsf{LSeq}}$ to the DSLS assumption, with an additive loss of $n^2\ell_{\mathsf{eval}}^4/8p$, where n_{H_1} corresponds to the number of queries that the adversary makes to H_1. We therefore put $\log p = 2 \cdot \lambda + 4 \cdot \log(\ell_{\mathsf{eval}}) - 8 = 281$ to ensure a meaningful reduction. For such a large prime we have a wide security margin against attacks on the DSLS assumption because the best-known attacks against the DSLS problem run in time $\Theta(\max(\sqrt{p \log p}, p \log p/(q\ell_{\mathsf{eval}})^2)$ [10,35] where q is the number of queries that the adversary is allowed to make to the OPRF. Even in the (unrealistic) case where an adversary can make an unbounded number of queries, the best attack requires $2^{144} \gg 2^{128}$ operations. If the adversary is restricted to making a realistic number of queries, e.g. $q \leq 2^{40}$ the security margin is much larger still.
- The ℓ_{com} parameter needs to be large enough to ensure weak collision resistance. Lemma 8 bounds the probability that an adversary finds a key-collision by $p^2 \cdot q_{\mathsf{H}_1} \cdot 2^{-\ell_{\mathsf{com}}-\ell_{\mathsf{eval}}-1}$, so we set $\ell_{\mathsf{com}} = 2\log p + \lambda - \ell_{\mathsf{eval}} - 1 = 561$.
- The statistical security parameter $s = 2^{64}$.
- The parameter $t = \log|S_\Delta|$ used by the sVOLE protocol of [42]. This t controls a trade-off between communication and computational efficiency.

For our estimates we use $t = 8$, as recommended by the author of [42] in a WAN setting.

Message complexity. If we instantiate the $\mathcal{F}_{\text{VOLE+}}$ functionality by the protocol of Appendix D, and the $\mathcal{F}_{\text{ZK}}$ functionality with the Quicksilver proof system, then the overall protocol has a message complexity of 9 (first message sent by the server). The first 5 messages are for the commitment phase of the zero-knowledge proof and the input-independent setup of the VOLE$^+$ protocol (which can be sent in parallel), then two messages to finish the VOLE$^+$ protocol, and then two more messages to finish the zero-knowledge proof.

Communication complexity. The most significant contributions to the communication cost are as follows:

- The cost of the base OT's. If we use the generic OT of [4] instantiated with Kyber-512, aiming for 128 bits of security we estimate the communication cost to be 296 KB, coming from the user sending λ Kyber-512 public keys, and the server sending $2\lambda = 512$ Kyber-512 ciphertexts. We believe this step could be optimized by using a batched OT primitive rather than using λ repetitions of a single-use OT protocol.
- The $1 + \lceil(\log p + 2s)/t\rceil = 53$ vectors of length $\ell_{\mathsf{eval}} + 1 = 129$ over $\mathbb{F}_p$ sent during the execution of the VOLE$^+$ protocol. These vectors have a combined size of 234.5 KB.
- The $\lceil \lambda/t \rceil = 16$ vectors of length $\ell_{\mathsf{eval}} + \ell_{\mathsf{com}} + \deg(\mathcal{F}) = 694$ sent during the setup phase of the Quicksilver zero-knowledge proof system. These vectors have a combined size of 380.9 KB.

This makes for a combined communication cost of 911 KB.

Time. To make things concrete, we assume the user and client are communicating over a network with a latency of 30 ms and a throughput of 25 Mbps. We then estimate that just sending back and forth the messages takes $9 \cdot 30$ ms +911 KB /25 Mbps = 0.57 seconds. We expect communication to be the bottleneck for our protocol since the computational cost of the protocol is quite low. The base OT requires $O(\lambda)$ executions of Kyber-512, whose cost is negligible compared to 0.56 seconds, similarly, the cost of computing a small amount of Legendre symbols is negligible. The most computationally costly part of the protocol is expanding many PRG seeds into vectors of length $\ell_{\mathsf{eval}} + 1$ and $\ell_{\mathsf{eval}} + \ell_{\mathsf{com}} + \deg(\mathcal{F})$ for the VOLE$^+$ and Quicksilver protocol respectively. For the proposed parameters the total amount of PRG output is approximately 152.4 MB, which would take less than 40 ms to expand using an AES-based PRG on modern CPUs with the AES-NI instruction set.

Conclusion and future work. We expect our OPRF protocol to be orders of magnitude more efficient than existing quantum-safe OPRF protocols, and we see a lot of opportunities for optimization. E.g., one can use batched base OTs or replace Legendre symbols with power residue symbols to decrease the ℓ_{com} and ℓ_{eval} parameters, which would drastically reduce the communication cost of the VOLE$^+$ protocol and the zero-knowledge proofs. We believe a tighter reduction from DSLS to the one-more unpredictability of $\mathcal{F}_{\mathsf{LSeq}}$ is possible, which would allow us to use a smaller prime p. Lastly, we can hope to improve the function evaluation protocol, perhaps by further weakening the $\mathcal{F}_{\text{SFE}}$ functionality, which would require us to strengthen our main theorem to prove that the weaker $\mathcal{F}_{\text{SFE}}$ functionality is still enough to achieve $\mathcal{F}^{f}_{\text{2H-OPRF}}$. This would be similar to the 2-hash DH OPRF, which uses a very efficient evaluation protocol for a weaker (but still sufficiently strong) version of the $\mathcal{F}_{\text{SFE}}$ functionality. We leave the task of exploring these optimizations and producing an optimized implementation of our protocol for future work.

As noted already in Section 2, our Legendre-based OPRF is *verifiable*, i.e., a client can efficiently decide whether the evaluation was performed correctly by the server. The 2HashDH OPRF lends itself not only to verifiability [27] but also to other properties of OPRFs such as, e.g., partially oblivious evaluation [21,9] or threshold evaluation [30]. We leave it for future work to extend our framework to capture such properties of OPRFs, e.g., to add verifiability to $\mathcal{F}^{f}_{\text{2H-OPRF}}$ and formally prove our compiler to realize it given a key-collision resistant f_{pub}, or to give threshold versions of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ and $\mathcal{F}^{f}_{\text{SFE}}$ that allow to lift our compiler to the threshold setting.

References

1. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_10
2. Albrecht, M.R., Davidson, A., Deo, A., Gardham, D.: Crypto dark matter on the torus: Oblivious PRFs from shallow PRFs and FHE. Cryptology ePrint Archive, Report 2023/232 (2023), https://eprint.iacr.org/2023/232
3. Albrecht, M.R., Davidson, A., Deo, A., Smart, N.P.: Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 261–289. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75248-4_10
4. Badrinarayanan, S., Masny, D., Mukherjee, P.: Efficient and tight oblivious transfer from PKE with tight multi-user security. In: Ateniese, G., Venturi, D. (eds.) ACNS 22. LNCS, vol. 13269, pp. 626–642. Springer, Heidelberg (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_31
5. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 433–444. ACM Press (Oct 2011). https://doi.org/10.1145/2046707.2046758
6. Basso, A.: A post-quantum round-optimal oblivious PRF from isogenies. Cryptology ePrint Archive, Report 2023/225 (2023), https://eprint.iacr.org/2023/225
7. Basso, A., Kutas, P., Merz, S.P., Petit, C., Sanso, A.: Cryptanalysis of an oblivious PRF from supersingular isogenies. In: Tibouchi, M., Wang, H. (eds.) ASI-

ACRYPT 2021, Part I. LNCS, vol. 13090, pp. 160–184. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92062-3_6

8. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_19
9. Baum, C., Frederiksen, T.K., Hesse, J., Lehmann, A., Yanai, A.: PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In: IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020. pp. 587–606. IEEE (2020). https://doi.org/10.1109/EUROSP48549.2020.00044, https://doi.org/10.1109/EuroSP48549.2020.00044
10. Beullens, W., Beyne, T., Udovenko, A., Vitto, G.: Cryptanalysis of the Legendre PRF and generalizations. IACR Trans. Symm. Cryptol. **2020**(1), 313–330 (2020). https://doi.org/10.13154/tosc.v2020.i1.313-330
11. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
12. Boneh, D., Ishai, Y., Passelègue, A., Sahai, A., Wu, D.J.: Exploring crypto dark matter: New simple PRF candidates and their applications. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018, Part II. LNCS, vol. 11240, pp. 699–729. Springer, Heidelberg (Nov 2018). https://doi.org/10.1007/978-3-030-03810-6_25
13. Boneh, D., Kogan, D., Woo, K.: Oblivious pseudorandom functions from isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 520–550. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_18
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
15. Casacuberta, S., Hesse, J., Lehmann, A.: Sok: Oblivious pseudorandom functions. In: 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022. pp. 625–646. IEEE (2022). https://doi.org/10.1109/EUROSP53844.2022.00045, https://doi.org/10.1109/EuroSP53844.2022.00045
16. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
17. Damgård, I.: On the randomness of Legendre and Jacobi sequences. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 163–172. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34799-2_13
18. Das, P., Hesse, J., Lehmann, A.: DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIACCS 22. pp. 682–696. ACM Press (May / Jun 2022). https://doi.org/10.1145/3488932.3517389
19. Davies, G.T., Faller, S.H., Gellert, K., Handirk, T., Hesse, J., Horváth, M., Jager, T.: Security analysis of the WhatsApp end-to-end encrypted backup protocol. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 330–361. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_11

20. Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 517–547. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_18
21. Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 547–562. USENIX Association (Aug 2015)
22. Faller, S., Ottenhues, A., Ottenhues, J.: Composable oblivious pseudo-random functions via garbled circuits. In: Aly, A., Tibouchi, M. (eds.) LATINCRYPT 2023. pp. 249–270. Springer Nature Switzerland, Cham (2023), https://doi.org/10.1007/978-3-031-44469-2_13
23. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 430–443. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978332
24. Hazay, C.: Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 90–120. Springer, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46497-7_4
25. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_10
26. Heimberger, L., Hennerbichler, T., Meisingseth, F., Ramacher, S., Rechberger, C.: Oprfs from isogenies: Designs and analysis. Cryptology ePrint Archive, Paper 2023/639 (2023), https://eprint.iacr.org/2023/639, https://eprint.iacr.org/2023/639
27. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_13
28. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: 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, Saarbrücken, Germany, March 21-24, 2016. pp. 276–291. IEEE (2016)
29. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 276–291. IEEE (2016)
30. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 17. LNCS, vol. 10355, pp. 39–58. Springer, Heidelberg (Jul 2017). https://doi.org/10.1007/978-3-319-61204-1_3
31. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15
32. Jarecki, S., Krawczyk, H., Xu, J.: On the (in)security of the diffie-hellman oblivious PRF with multiplicative blinding. In: Garay, J. (ed.) PKC 2021, Part II. LNCS,

vol. 12711, pp. 380–409. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75248-4_14

33. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_34
34. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1447–1464. USENIX Association (Aug 2019)
35. Kaluđerović, N., Kleinjung, T., Kostić, D.: Cryptanalysis of the generalised legendre pseudorandom function. Open Book Series **4**(1), 267–282 (2020), https://doi.org/10.2140/obs.2020.4.267
36. Khovratovich, D.: Key recovery attacks on the Legendre PRFs within the birthday bound. Cryptology ePrint Archive, Report 2019/862 (2019), https://eprint.iacr.org/2019/862
37. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 818–829. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978381
38. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Report 2022/1026 (2022), https://eprint.iacr.org/2022/1026
39. Masny, D., Rindal, P.: Endemic oblivious transfer. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 309–326. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354210
40. Mouris, D., Masny, D., Trieu, N., Sengupta, S., Buddhavarapu, P., Case, B.: Delegated private matching for compute. Cryptology ePrint Archive, Paper 2023/012 (2023), https://eprint.iacr.org/2023/012
41. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: 38th FOCS. pp. 458–467. IEEE Computer Society Press (Oct 1997). https://doi.org/10.1109/SFCS.1997.646134
42. Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 657–687. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_23
43. Seres, I.A., Horváth, M., Burcsi, P.: The legendre pseudorandom function as a multivariate quadratic cryptosystem: security and applications. Applicable Algebra in Engineering, Communication and Computing pp. 1–31 (2023)
44. Team, F.P.P.: Silk - Privacy Pass Client (accessed Jan 28 2024), https://addons.mozilla.org/en-US/firefox/addon/privacy-pass/
45. WhatsApp: Security of End-To-End Encrypted Backups (September 10, 2021), https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf
46. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484556

A Correlated OPRFs and their relation to $\mathcal{F}^{f}_{\text{2H-OPRF}}$

An alternative UC security notion for OPRFs called *Correlated OPRF* was introduced by Jarecki et al. as a weaker notion, used to show the security of the Diffie-Hellman OPRF with multiplicative blinding [32]. While this is a weaker notion, it still suffices for various applications, such as OPAQUE [31]. In the following, we will first present the Correlated OPRF functionality.

Ideal functionality Again, we slightly modify the original correlated OPRF definition by removing the prefixes. That means, upon the user or server sending their input messages for online evaluation, we no longer allow the adversary to make the parties output some value and link a client and server session together.

The new definition can be found in Fig. 7. The main difference to $\mathcal{F}_{\text{OPRF}}$ is that for any two PRF functions F_1, F_2 we now allow them to be correlated on a single value of the adversaries choosing. That means when a new function F' is referenced for the first time, the adversary can provide a list of pairs (F_i, x_i) and the functionality will ensure that $F'(x_i) = F_i(x_i)$, assuming that each function F_i only occurs once in this list. The ideal functionality models this by storing this information as a graph. The instantiated functions are saved in a set of nodes $\mathcal{N}$, and we have a set of edges $\mathcal{E}$ that model the correlations. Each edge between two nodes has a label x, indicating the value on which the two nodes are correlated. When adding a new node to the graph (i.e., the first time a function is referenced), the CORRELATE function models adding the specified edges to the graph. The practical difference compared to the previous functionality is that a new attack is permitted. In certain cases, a corrupted server can test if a client has previously interacted with the server with a value x. In particular, if the higher-level application allows the server to detect if a client outputs the same output in two interactions, then by answering the two interactions with different tables F_i, F_j where they are correlated on a value x, with $\mathcal{F}_{\text{corOPRF}}$ the adversary can with high probability detect if the user's input was the value x in both interactions or not by comparing whether the outputs were the same or not.

Proof of Lemma 2 Proof idea: $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is $\mathcal{F}_{\text{OPRF}}$ plus a preview interface. All queries to the "$\mathcal{F}_{\text{OPRF}}$-part" of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ are simulated using the corresponding interfaces at $\mathcal{F}_{\text{corOPRF}}$, which, as it is weaker than $\mathcal{F}_{\text{OPRF}}$, inherits all those interfaces. We now need to argue that the previewing interfaces of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ (middle) can be simulated with the correlation interfaces of $\mathcal{F}_{\text{corOPRF}}$ (left). The right-hand side of this sketch shows the overall idea: previews are obtained from fresh keys $\bar{S}$, and they are correlated with a new adversarial key S^*_1 *if* the respective f equation is satisfied.

We now specify how the previewing part of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ is simulated. First, H_1 is simulated with the same code as in $\mathcal{F}^{f}_{\text{2H-OPRF}}$, which is indistinguishable because the code does not depend on any values hidden from the simulator. If

Ideal Correlated OPRF functionality $\mathcal{F}_{\text{corOPRF}}$

The OPRF function is parameterised by a public PRF output length λ. For every i and x the value $F_i(x)$ is initially undefined. The first time an undefined value $F_i(x)$ is referenced $\mathcal{F}_{\text{corOPRF}}$ sets $F_i(x) \leftarrow\$ \{0,1\}^\lambda$.

Initialisation:
On message (Init, *sid*) from party S, if this is the first Initmessage for *sid* set $tx = 0$ and send (Init, *sid*, S) to $\mathcal{A}^*$. From now on use the tag S to denote the unique entity which sent the Initmessage for the session identifier *sid*. (Ignore all subsequent Initmessages for *sid*.) Finally, set $\mathcal{N} \leftarrow [S], \mathcal{E} \leftarrow \{\}, \mathcal{G} \leftarrow (\mathcal{N}, \mathcal{E})$

Server Compromise:
On message (Compromise, *sid*) from $\mathcal{A}^*$, declare S as Compromised.
Note: Message (Compromise, *sid*) requires permission from the environment. //If S is corrupted, then it is declared Compromisedas well.

Offline Evaluation:
On (OfflineEval, *sid* , S^*, x, L) from $P \in \{S, \mathcal{A}^*\}$ do:

- If $P = \mathcal{A}^*$ and $S^* \notin \mathcal{N}$: append S^* to $\mathcal{N}$ and run Correlate(S^*, L)
- Ignore message if $P = \mathcal{A}^*$, S not Compromised, and $(S^*, S, x) \in \mathcal{E}$
- Send (OfflineEval, *sid*, $F_{S^*}(x)$) to P if (i) $P = S$ and $S^* = S$ or (ii) $P = \mathcal{A}^*$ and either $S^* \neq S$ or S Compromised

Online Evaluation:

- On (Eval, *sid*, *ssid*, S', x) from $P \in \{U, \mathcal{A}^*\}$, send (Eval, *sid*, *ssid*, P, S') to $\mathcal{A}^*$. Record $\langle ssid, P, x\rangle$
- On (SndrComplete, *sid*, *ssid'*) from S, send (SndrComplete, *sid*, *ssid'*, S) to $\mathcal{A}^*$, set tx++
- On (RcvComplete, *sid*, *ssid*, P, S^*, L) from $\mathcal{A}^*$, ignore this message if there is no record $\langle ssid, P, x\rangle$ stored. Else:
  - If $S^* \notin \mathcal{N}$: Append S^* to $\mathcal{N}$, run Correlate(S^*, L)
  - If S is not Compromisedand ($S^* = S$ $[(S^*, S, x) \in \mathcal{E}$ and $P = \mathcal{A}^*]$):
    If $tx = 0$ ignore this message. Else decrement tx
  - Send (Eval, $sid, ssid, F_{S^*}(x)$) to P

Correlate (S^*, L):
Reject if $|L| > 1$. Reject if L contains an element (j, x) with $j = S$.
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 (S^*, j, x) to $\mathcal{E}$ and set $F_{S^*}(x) \leftarrow F_j(x)$

Fig. 7: The Correlated OPRF functionality $\mathcal{F}_{\text{corOPRF}}$. The changes to $\mathcal{F}_{\text{OPRF}}$ are highlighted using grey boxes. Changes to achieve a *strong* correlated OPRF are highlighted using dashed lines.

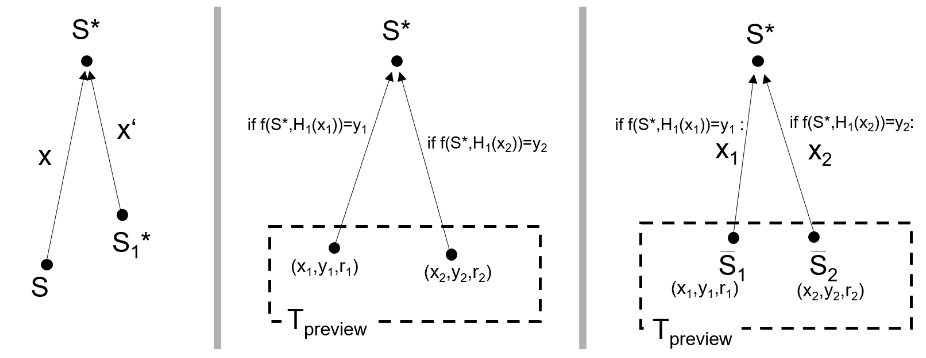

Fig. 8: Illustration of different correlation techniques related to Lemma 2. **Left:** $\mathcal{F}_{\text{corOPRF}}$ allows to correlate each new adversarial key (S^*) with all previous keys (S, S_1^*) on adversarially-chosen inputs (x, x'), including the honest key S. **Middle:** $\mathcal{F}^f_{\text{2H-OPRF}}$ *automatically* correlates each new adversarial key (S^*) by checking consistency with f. It never correlates with the honest key. **Right:** Our simulator of Lemma 2 creates dummy keys ($\bar{S}_1, \bar{S}_2$) to obtain previews, and correlates them for specific inputs if they turn out to be consistent with a new adversarial key (S^*). It never exploits its power to correlate with the honest key.

adv queries H_2 with y^*, x, the simulator first checks whether for any adversarial K^* it holds that $f(K^*, H_1(x)) = y^*$ and, if so, ignores the preview request. Otherwise, it chooses a uniformly random value $\bar{S}$. The simulator then submits $\bar{S}, x$ to OfflineEval, obtaining reply r that it outputs to $\mathcal{A}^*$. The simulator also sets as $\bar{S} := T_{\text{preview}}(y^*, x)$. The only thing left to take care of is the effect of the calls of the Correlate function in $\mathcal{F}^f_{\text{2H-OPRF}}$. First, observe that the Correlate procedure as well only depends on values that the adversary submitted, and hence the simulator can execute the code as in $\mathcal{F}^f_{\text{2H-OPRF}}$, with the only exception being the command "Set $F_{K^*}(x) := T_{\text{preview}}(K^*, x)$", i.e., the programming of a previewed value into an adversarial function table $F_{K^*}(\cdot)$. To achieve this programming, the simulator uses the correlation interface at $\mathcal{F}_{\text{corOPRF}}$, i.e., whenever the adversary submits K^* to either OfflineEval or RcvComplete, the simulator sets $L = \emptyset$ and for all previously submitted (y^*, x) for which $f(K^*, H_1(x)) = y^*$, it adds $(T_{\text{preview}}(y^*, x), x)$ to L. L is then appended to the OfflineEval resp. RcvComplete call that the simulator passes on to $\mathcal{F}_{\text{corOPRF}}$. This way, $\mathcal{F}_{\text{corOPRF}}$ will program $F_{K^*}(x)$ to $F_{\bar{S}}(x) = r$, and the adversarial function tables in $\mathcal{F}^f_{\text{2H-OPRF}}$ and $\mathcal{F}_{\text{corOPRF}}$ match.

B Proof of Theorem 1

Proof. Without loss of generality, we assume that the adversary $\mathcal{A}^*$ is the dummy adversary [14], who does nothing other than pass along all its messages to and from $\mathcal{Z}$. The simulator is shown in Fig. 10. We assume that the server identifier

The simulator maintains functions H_1, $T_{\text{preview}}(\cdot,\cdot)$ initially undefined everywhere, and initially empty sets $\mathcal{K}, T_{\text{programmed}}$. $\mathsf{dom}(T_{\text{preview}})$ the set of tuples these functions are already defined on, i.e., upon initialization we have $\mathsf{dom}(T_{\text{preview}}) = \emptyset$. The first time an undefined value $H_1(x)$ is referenced, the simulator chooses $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ and sets $H_1(x) := r$.

Messages from $\mathcal{F}_{corOPRF}$:
On $(\textsc{Init}, sid)$ from $\mathcal{F}_{\text{corOPRF}}$, forward to $\mathcal{A}^*$.
On $(\textsc{Eval}, sid, ssid, P, S')$ from $\mathcal{F}_{\text{corOPRF}}$, forward to $\mathcal{A}^*$.
On $(\textsc{SndrComplete}, sid, ssid, S)$ from $\mathcal{F}_{\text{corOPRF}}$, forward to $\mathcal{A}^*$.
On from $\mathcal{F}_{\text{corOPRF}}$, forward to $\mathcal{A}^*$.

Messages from the adversary $\mathcal{A}^*$:
On $(\textsc{Compromise}, sid)$ from $\mathcal{A}^*$, forward to $\mathcal{F}_{\text{corOPRF}}$.

On $(\textsc{OfflineEval}, sid, ssid, K^*, x)$ from $\mathcal{A}^*$:
- Add K^* to $\mathcal{K}$ and run $L \leftarrow \mathsf{getL}(K^*)$
- Send $(\textsc{OfflineEval}, sid, ssid, K^*, x, L)$ to $\mathcal{F}_{\text{corOPRF}}$
- Upon receiving back $(\textsc{OfflineEval}, sid, ssid, r)$, forward it to $\mathcal{A}^*$.

On $(\textsc{RcvComplete}, sid, ssid, P, K^*)$ from $\mathcal{A}^*$
- Add K^* to $\mathcal{K}$ and run $L \leftarrow \mathsf{getL}(K^*)$
- Send $(\textsc{RcvComplete}, sid, ssid, P, K^*, L)$ to $\mathcal{F}_{\text{corOPRF}}$.

On $H_1(x, sid)$ from $\mathcal{A}^*$, reply with $H_1(x)$

On (H_2, sid, y^*, x) from $\mathcal{A}^*$:
- Iterate through list $\mathcal{K}$. If for any $K^* \in \mathcal{K}$ it holds that $f(K^*, H_1(x)) = y^*$, set $\bar{S} \leftarrow K^*$ and $\bar{S} := T_{\text{preview}}(y^*, x)$, otherwise choose $\bar{S} \leftarrow \{0,1\}^\lambda$ //Abusing the semantics of the T_{preview} list: Simulator needs to remember $\bar{S}$, not r.
- Send $(\textsc{OfflineEval}, sid, ssid, \bar{S}, x)$ to $\mathcal{F}_{\text{corOPRF}}$ and receive back $(\textsc{OfflineEval}, sid, ssid, r)$
- Send (H_2, sid, r) to $\mathcal{A}^*$.

<u>Procedure $\mathsf{getL}(K^*)$:</u>
```
∘ Set $L = \emptyset$
∘ For all $(y^*, x) \in \mathsf{dom}(T_{\text{preview}}) \setminus T_{\text{programmed}}$:
∘     if $f(K^*, H_1(x)) = y^*$ then add $(T_{\text{preview}}(y^*, x), x)$ to $L$ and $(y^*, x)$ to $T_{\text{programmed}}$
∘ return $L$
```

Fig. 9: Simulator for Lemma 2.

is some unique value such that if we choose some random $K' \in \mathcal{K}$ it will be unequal to the server identifier.

We now show a sequence of hybrid experiments $\mathbf{G}_0,\ldots,\mathbf{G}_{10}$ where, starting from the real-world execution, we make small incremental changes until we reach the ideal-world execution with the above simulator. We write $\Pr[\mathbf{G}_i]$ as the probability that the environment outputs 1 in game $\mathbf{G}_i$. We also write $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}})$.

Game $\mathbf{G}_0$: This is the real world execution of $\textsc{Exec}_{\Pi_{\mathrm{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f},\mathcal{A}^*,\mathcal{Z}}$

Game $\mathbf{G}_1$: Create functionality and simulator. In this game, we create dummy parties and a functionality $\mathcal{F}$ such that the parties forward all their inputs to $\mathcal{F}$. We also introduce a simulator Sim and let $\mathcal{F}$ forward all inputs it receives from the dummy parties to Sim. The simulator internally executes the code of $\Pi_{\mathrm{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f}$ for all parties on their respective inputs. In particular, Sim chooses a uniformly random key $K \leftarrow\$ \mathcal{K}$ for the honest server and uses this key in protocol executions with the honest server. Finally, we equip $\mathcal{F}$ with an interface that allows Sim to make any honest party output a value provided by Sim. Then Sim makes these parties output whatever they would output according to the internal execution of $\Pi_{\mathrm{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f}$.

Note that these are only syntactical changes and the protocol is executed as before. Thus, we have

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Game $\mathbf{G}_2$: H_1 from $\mathcal{F}$. In this game we add the H_1 interface to $\mathcal{F}$ and also change Sim such that it uses this interface. First, we let Sim record all user input that is being send to $\mathcal{F}_{\mathrm{SFE}}^f$. That means, on a message $(\textsc{UserInput}, ssid, h)$ to $\mathcal{F}_{\mathrm{SFE}}^f$ the simulator records $\left\langle \mathcal{F}_{\mathrm{SFE}}^f, ssid, U, h \right\rangle$. Then, on a new $\mathsf{H}_1(x, sid)$ query Sim forwards x to $\mathcal{F}_{\text{2H-OPRF}}^f$'s H_1 interface. When $\mathcal{F}_{\text{2H-OPRF}}^f$ responds with (H_1, sid, h) Sim checks if there is a record $\left\langle \mathcal{F}_{\mathrm{SFE}}^f, ssid, U, h \right\rangle$. If there exists such a record then Sim aborts. We call this event E_{guess}. Else Sim stores $\langle \mathsf{H}_1, sid, x, h\rangle$.

Note that $\mathcal{F}_{\text{2H-OPRF}}^f$ answers the H_1 queries with uniformly random values. So $\mathbf{G}_2$ and $\mathbf{G}_1$ only differ if E_{guess} occurs. But that means that the adversary sent some input h to $\mathcal{F}_{\mathrm{SFE}}^f$ and afterwards submitted a query $\mathsf{H}_1(x) = h$. But $\mathsf{H}_1(x)$ is chosen by $\mathcal{F}_{\text{2H-OPRF}}^f$ uniformly at random. Let n_{H_1} be the number of H_1 queries. Then, E_{guess} happens at most with probability

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| = \Pr[E_{\text{guess}}] \leq \frac{n_{\mathsf{H}_1}}{|\mathcal{I}|}.$$

Game $\mathbf{G}_3$: Add $\textsc{OfflineEval}$ and H_2 interfaces. We now augment the functionality $\mathcal{F}$ with tables $F_{\text{honest}}(\cdot)$ and $F_{\text{malicious}}(\cdot)$, which are initially uninitialized and on the first reference, are set to a random element in $\{0,1\}^\lambda$.

Simulator $\mathsf{Sim}(sid)$

Initially, set $\mathcal{K} := []$. Parse $f = (f_{\mathsf{pub}}, f_{\mathsf{sec}})$.

On a $\mathsf{H}_1(x, sid)$ query if $\mathsf{H}_1(x)$ is undefined forward the query to $\mathcal{F}^f_{\text{2H-OPRF}}$'s H_1 interface. When $\mathcal{F}^f_{\text{2H-OPRF}}$ responds with (H_1, sid, h) check if there is a record $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$. If there is such a record then abort $//E_{\text{guess}}$.
Else store $\langle \mathsf{H}_1, sid, x, h\rangle$ and send $\mathsf{H}_1(x) := h$ as response to $\mathcal{A}^*$.

On $(\textsc{Init}, sid, S)$ from $\mathcal{F}^f_{\text{2H-OPRF}}$ choose $K \leftarrow\$ \mathcal{K}$ and store $\langle S, sid, K\rangle$. //Honest server's key.

On $(\textsc{Compromise}, sid)$ from $\mathcal{A}^*$, send $(\textsc{Compromise}, sid)$ to $\mathcal{F}^f_{\text{2H-OPRF}}$, retrieve $\langle S, sid, K\rangle$ and send K to $\mathcal{A}^*$. Record that S is compromised.

On $(\textsc{UserInput}, ssid, h)$ to $\mathcal{F}^f_{\text{SFE}}$ on behalf of a malicious U store $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$. If there is a record $\langle \mathsf{H}_1, sid, x, h\rangle$ send $(\textsc{Eval}, sid, ssid^*, S, x)$ to $\mathcal{F}^f_{\text{2H-OPRF}}$.

On $(\textsc{ServerInput}, ssid, K')$ to $\mathcal{F}^f_{\text{SFE}}$ on behalf of a malicious S' store $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, S, K' \right\rangle$ and add K' to $\mathcal{K}$. Send $(\textsc{ServerInput}, sid, S, f_{\mathsf{pub}}(K))$ to $\mathcal{A}^*$.

On $(\textsc{Output}, ssid)$ to $\mathcal{F}^f_{\text{SFE}}$ from $\mathcal{A}^*$:
- If there are records $\langle U, ssid\rangle$ and $\langle S, ssid\rangle$ send $(\textsc{RcvCompleteHonest}, sid, ssid, U)$ to $\mathcal{F}^f_{\text{2H-OPRF}}$ //Both honest
- If there are records $\langle U, ssid\rangle$ and $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, S, K' \right\rangle$ then send $(\textsc{RcvCompleteMalicious}, sid, ssid, U, K')$ to $\mathcal{F}^f_{\text{2H-OPRF}}$ //Honest U malicious S
- If there are records $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$ and $\langle S, ssid\rangle$ then retrieve $\langle S, sid, K\rangle$ and send $(ssid, f_{\mathsf{sec}}(h, K))$ to U. //Malicious U honest S
- If there are records $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$ and $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, S, K' \right\rangle$ then send $(ssid, f_{\mathsf{sec}}(h, K'))$ to U in the name of $\mathcal{F}^f_{\text{SFE}}$ //Both malicious

On $(\textsc{Eval}, sid, ssid, U, S')$ from $\mathcal{F}^f_{\text{2H-OPRF}}$, record $\langle U, ssid\rangle$ and send $(\textsc{UserInput}, ssid, U)$ to $\mathcal{Z}$ in the name of $\mathcal{F}^f_{\text{SFE}}$. //Honest user would call $\mathcal{F}^f_{\text{SFE}}$

On $(\textsc{SndrComplete}, sid, ssid, S)$ from $\mathcal{F}^f_{\text{2H-OPRF}}$, record $\langle S, ssid\rangle$ and send $(\textsc{ServerInput}, ssid, S)$ to $\mathcal{A}^*$ in the name of $\mathcal{F}^f_{\text{SFE}}$ //Honest server would call $\mathcal{F}^f_{\text{SFE}}$.

On a $\mathsf{H}_2(x, y, sid)$ query if $\mathsf{H}_2(x, y)$ is undefined, retrieve $\langle S, sid, K\rangle$. Then:
- If $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K)$ then retrieve $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$ and $\langle S, ssid\rangle$. Send $(\textsc{RcvCompleteHonest}, sid, ssid, \mathsf{Sim})$ to $\mathcal{F}^f_{\text{2H-OPRF}}$. If there is no response from $\mathcal{F}^f_{\text{2H-OPRF}}$ then abort $//E_{\text{excess}}$.
  Else when $\mathcal{F}^f_{\text{2H-OPRF}}$ answers with $(\textsc{Eval}, sid, ssid, r)$ set $\mathsf{H}_2(x, y) := r$
- Else if there exist two keys $K', K'' \in \mathcal{K}$ such that $K' \neq K''$ and $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K') = f_{\mathsf{sec}}(\mathsf{H}_1(x), K'')$ then abort. $//E_{\text{collision}}$.
- Else if for all $K' \in \mathcal{K}$ it holds that $y \neq f_{\mathsf{sec}}(x, K')$ then send (H_2, sid, y, x) to $\mathcal{F}^f_{\text{2H-OPRF}}$. When $\mathcal{F}^f_{\text{2H-OPRF}}$ answers with $r \in \{0,1\}^\lambda$ set $\mathsf{H}_2(x, y) := r$.
- Else there is exactly one key $K' \in \mathcal{K}$ such that $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K')$. Send $(\textsc{OfflineEval}, sid, ssid, K', x)$ to $\mathcal{F}^f_{\text{2H-OPRF}}$. When $\mathcal{F}^f_{\text{2H-OPRF}}$ answers with $r \in \{0,1\}^\lambda$ set $\mathsf{H}_2(x, y) := r$.

Fig. 10: The simulator that shows that $\Pi^{\mathsf{H}_1,\mathsf{H}_2,f}_{\text{OPRF}}$ UC-realizes $\mathcal{F}^f_{\text{2H-OPRF}}$. We assume that Sim ignores a message when it tries to retrieve a record that does not exist.

Furthermore, we add the OFFLINEEVAL interfaces. Like in $\mathcal{F}^{f}_{\text{2H-OPRF}}$, the OFFLINEEVAL interface for adversarial functions, given a message (OFFLINEEVAL, $sid, ssid, K^*, x$) from $\mathcal{A}^*$, responds with (OFFLINEEVAL, $sid, ssid, F_{\text{malicious}}(K^*, x)$) and CORRELATE($K^*$) is called. For now, the OFFLINEEVAL interface for the honest function only responds with (OFFLINEEVAL, $sid, ssid, F_{\text{honest}}(x)$) if the query comes from $\mathcal{A}^*$. In contrast to the honest function OFFLINEEVAL interface of $\mathcal{F}^{f}_{\text{2H-OPRF}}$ requests from $P = S$ are forwarded to Sim and answered by Sim with $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K))$. Finally, we add the H_2 interface to $\mathcal{F}$ exactly as it is in $\mathcal{F}^{f}_{\text{2H-OPRF}}$.

Since currently, no one uses these functionality tables or the new interfaces, we have that

$$\Pr[\mathbf{G}_3] = \Pr[\mathbf{G}_2].$$

Game $\mathbf{G}_4$: Abort on key-collisions. In this game, we change Sim.

When Sim receives some K' as $\mathcal{A}^*$'s input to $\mathcal{F}^{f}_{\text{SFE}}$ then Sim stores this key in a list $\mathcal{K}$. Next, Sim checks on every $\mathsf{H}_2(x, y)$ query if there exist two keys $K', K'' \in \mathcal{K}$ such that $K' \neq K''$ and $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K') = f_{\mathsf{sec}}(\mathsf{H}_1(x), K'')$. If that is the case then Sim aborts. We call this event $E_{\text{collision}}$. We reduce breaking the n_{H_1}-key-collision resistance of f to $E_{\text{collision}}$ happening.

Let $\mathcal{Z}$ be an environment that interacts with $\mathbf{G}_4$ such that $E_{\text{collision}}$ happens and let n_{H_1} be the number of H_1 queries that $\mathcal{Z}$ does. We construct an adversary $\mathcal{B}$ for the n_{H_1}-weak key-collision resistance game that internally runs $\mathcal{Z}$ in $\mathbf{G}_4$ and plays the role of the simulator. $\mathcal{B}$ receives $h_1, \ldots, h_{n_{\mathsf{H}_1}}$ as challenge and answers the i-th H_1 query x_i with $\mathsf{H}_1(x_i) = h_i$ and records (h_i, i). When $\mathcal{Z}$ eventually sends a H_2 query (x^*, y^*) that satisfies the abort condition from $E_{\text{collision}}$, i.e., $\mathcal{B}$ recorded K, K' in $\mathcal{K}$ such that $y = f_{\mathsf{sec}}(\mathsf{H}_1(x^*), K) = f_{\mathsf{sec}}(\mathsf{H}_1(x^*), K')$ then $\mathcal{B}$ retrieves the record $(\mathsf{H}_1(x^*), i^*)$ and outputs (K, K', i^*), which is a key collision.

$$\Pr[\mathbf{G}_4] = \Pr[\mathbf{G}_3] \leq \mathbf{Adv}^{\text{WKCR}}_{\mathcal{F},\mathcal{B}}(n_{\mathsf{H}_1}).$$

Game $\mathbf{G}_5$: Programming H_2 for malicious keys. Next, we modify how a hash function request $\mathsf{H}_2(x, y)$ is answered for some $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K')$ with $K' \neq K$, where K is the honest server's key. Instead of answering with a uniformly random value in $\{0,1\}^\lambda$ the simulator proceeds as follows:

1. If for all $K' \in \mathcal{K}$ it holds that $y \neq f_{\mathsf{sec}}(\mathsf{H}_1(x), K')$ then Sim sends (H_2, sid, y, x) to $\mathcal{F}$. When $\mathcal{F}$ answers with $r \in \{0,1\}^\lambda$ the simulator sets $\mathsf{H}_2(x, y) := r$.
2. Else, send (OFFLINEEVAL, $sid, ssid, K', x$) to $\mathcal{F}$. When $\mathcal{F}$ answers with $r \in \{0,1\}^\lambda$ the simulator sets $\mathsf{H}_2(x, y) := r$.

In the case of Item 1, Sim did not yet record a key that maps x to y under f. So, either this H_2 query is not linked to some interaction between user and server or the environment used a key $K^* \in \mathcal{K}$ to internally compute $y = f_{\mathsf{sec}}(x, K^*)$ but has not yet used K^* in a protocol execution. That means that Sim cannot extract the key yet. To that end Sim uses the preview interface H_2 of $\mathcal{F}^f_{\text{2H-OPRF}}$ to set $\mathsf{H}_2(x, y)$ to some uniformly random value. Also, note that $\mathcal{F}$ always provides Sim with a value $t = T_{\text{preview}}(y, x)$. That is because Sim uses the list $\mathcal{K}$ to keep track of already used keys. $\mathcal{F}$ only adds keys to its own list $\mathcal{K}$ if Sim sends them to $\mathcal{F}$ (via OfflineEval or RcvCompleteMalicious). So if Sim does not know a matching key, neither does $\mathcal{F}$.

In the case of Item 2, Sim already recorded a key K^* such that $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K^*)$ and retrieves the corresponding output value $F_{\text{malicious}}(K^*, x)$ from $\mathcal{F}$. Still, this is a uniformly random value in $\{0,1\}^\lambda$. In particular, the OfflineEval interface triggers $\mathcal{F}$'s procedure Correlate(K^*). That means, if there were previous queries $\mathsf{H}_2(x', y')$ with $y' = f_{\mathsf{sec}}(\mathsf{H}_1(x'), K')$ that were answered by Sim using the H_2 interface that $\mathcal{F}$ ensured for them that $F_{\text{malicious}}(K', x) = y'$ and $F_{\text{malicious}}(K', x) = y$.

Also, note that we did *not* change how the output of an honest user is produced yet. The simulator still gets the honest user's input x and makes the user output $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K')$. In other words, the random tables $F_{\text{malicious}}(K', \cdot)$ are only used to program H_2 so far.

Thus, we have
$$\Pr[\mathbf{G}_5] = \Pr[\mathbf{G}_4].$$

Game $\mathbf{G}_6$: Honest server evaluations. In this game, we change $\mathcal{F}$ and Sim. The goal is to change responses to H_2 queries (x, y) where $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K)$ for the key K which Sim uses to simulate the honest server. To this end, we add the online evaluation interfaces Eval, SndrComplete, and RcvCompleteHonest to $\mathcal{F}$. We add them exactly as they are in $\mathcal{F}^f_{\text{2H-OPRF}}$, except that Eval still forwards the user's input x to Sim. Again, we do not yet change how output for honest users is generated.

Sim uses the new interfaces as follows:

– When there is a message (UserInput$, ssid, h)$ to $\mathcal{F}^f_{\text{SFE}}$ the simulator now checks if there is a record $\langle \mathsf{H}_1, sid, x, h\rangle$. If that is the case then Sim sends (Eval$, sid, ssid^*, S, x)$ to $\mathcal{F}$.

– To answer a hash query $\mathsf{H}_2(x, y)$ where $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K)$ the simulator checks if there is a corresponding record $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h\right\rangle$ with $h = \mathsf{H}_1(x)$ and a record $\langle S, ssid\rangle$. If one of the records does not exist then Sim aborts. We call this event E_{excess}. Else if the records exist then Sim sends a (RcvCompleteHonest$, sid, ssid, \mathsf{Sim})$ message to $\mathcal{F}$. When Sim receives a response (Eval$, sid, ssid, r)$ it sets $\mathsf{H}_2(x, y) := r$. As a result, $\mathsf{H}_2(x, y)$ is now programmed to $F_{\text{honest}}(x)$ (instead of a random value).

First, note that $\mathbf{G}_6$ and $\mathbf{G}_5$ do not differ in case Sim does not abort. Because then H_2 has uniformly random outputs as in $\mathbf{G}_5$ and. In the following we argue that $\mathcal{F}$ does not ignore the $(\textsc{RcvCompleteHonest}, sid, ssid, \mathsf{Sim})$ message.

Let $\mathcal{Z}$ be an environment such that in an interaction with $\mathbf{G}_6$ the event E_{excess} occurs, let n be the number of H_1 queries that $\mathcal{Z}$ does, and let q be the number of tuples of messages $(\textsc{Eval}, \textsc{SndrComplete})$ that $\mathcal{F}$ receives. In other words q is the number of protocol executions in the ideal world. We reduce breaking the (n, q)-one-more unpredictability of f to provoking E_{excess}. The reduction $\mathcal{B}'$ receives $(h_1, \ldots, h_n) \in \mathcal{I}^n$ and $f_{\mathsf{pub}}(K)$ as challenge. Then, $\mathcal{B}'$ internally runs $\mathbf{G}_6$ with $\mathcal{Z}$ and $\mathcal{F}$, where $\mathcal{B}'$ plays the role of the simulator and the functionality except that $\mathcal{B}'$ answers the i-th H_1 query x_i to $\mathcal{F}$ with $\mathsf{H}_1(x_i) = h_i$ and that Sim uses its $f_{\mathsf{sec}}(\cdot, K)$-oracle instead of choosing a key for the honest server. That means, whenever Sim receives a message $(\textsc{SndrComplete}, sid, ssid, S)$ it stores $\langle S, ssid\rangle$ and sends $(\textsc{ServerInput}, sid, S, f_{\mathsf{pub}}(K))$ to $\mathcal{Z}$. When Sim receives an $\mathcal{F}^f_{\text{SFE}}$-messages $(\textsc{UserInput}, ssid, h)$, and $(\textsc{Output}, ssid)$ for a $ssid$ where Sim has a corresponding record $\langle S, ssid\rangle$ then Sim forwards $h \in \mathcal{I}$ as a query to its $f_{\mathsf{sec}}(\cdot, K)$ oracle. When the oracle returns $f_{\mathsf{sec}}(h, K)$ then Sim gives $(ssid, f_{\mathsf{sec}}(h, K))$ to the respective user. (Note that at this point, the honest user's code is still executed by Sim on their input x. That means, both, malicious and honest users send a $(\textsc{UserInput}, ssid, h)$ message to $\mathcal{F}^f_{\text{SFE}}$. Also, the honest user computes its output still as $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K))$. Now, whenever $\mathcal{B}'$ receives a query $\mathsf{H}_2(x, y)$ it uses its $\mathcal{V}^f_K(i, y)$ oracle to check if $y = f_{\mathsf{sec}}(h_i = \mathsf{H}_1(x), K)$ holds.

Note that Sim queries its $f_{\mathsf{sec}}(\cdot, K)$ oracle if two conditions are satisfied. (1) Sim received messages $(\textsc{UserInput}, ssid, h)$, and $(\textsc{Output}, ssid)$. In this case Sim sends an $\textsc{Eval}$ message to $\mathcal{F}$. (2) Sim stored $\langle S, ssid\rangle$. This means that Sim received a message $(\textsc{SndrComplete}, sid, ssid, S)$ from $\mathcal{F}$. Therefore, we can upper-bound the number of $f_{\mathsf{sec}}(\cdot, K)$ oracle queries by q. E_{excess} implies that there was at least one query $\mathsf{H}_2(x, y)$ with $y = f_{\mathsf{sec}}(\mathsf{H}_1(x), K)$ but without a matching record $\left\langle \mathcal{F}^f_{\text{SFE}}, ssid, U, h \right\rangle$ or a matching record $\langle S, ssid\rangle$. Consequently, Sim did either not send an $\textsc{Eval}$ message for this $ssid$ or there was no $\textsc{SndrComplete}$ message for this $ssid$. In other words, $\mathcal{Z}$ computed y without a protocol execution. This corresponds to $\mathcal{B}'$ making q queries to its $f_{\mathsf{sec}}(\cdot, K)$ oracle and $q + 1$ distinct queries to its $\mathcal{V}^f_K(\cdot, \cdot)$ oracle, such that $\mathcal{V}^f_K$ outputs 1.

Therefore, $\mathcal{B}'$ wins the game whenever E_{excess} occurs. We get

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \mathbf{Adv}^{\text{OMU}}_{\mathcal{F},\mathcal{B}'}(n, q).$$

Game $\mathbf{G}_7$: Allow offline evaluation by S. In this game we change the honest function's $\textsc{OfflineEval}$ interface of $\mathcal{F}$ such that $\mathcal{F}$ now also answers to $\textsc{OfflineEval}$ messages from S, exactly as in $\mathcal{F}^f_{\text{2H-OPRF}}$.

In $\mathbf{G}_6$ we ensured that $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K)) = F_{\text{honest}}(x)$, where $F_{\text{honest}}(x)$ is $\mathcal{F}$'s random table for the honest user. The honest function's OfflineEval interfaces of $\mathcal{F}$ sends exactly this value as output to S. That means $\mathcal{Z}$ can verify that $\text{OfflineEval}(x) = \mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K))$.

We get

$$\Pr[\mathbf{G}_7] = \Pr[\mathbf{G}_6].$$

Game $\mathbf{G}_8$: Honest user output from $\mathcal{F}$. In this game, we change $\mathcal{F}$ and Sim such that Sim produces output for an honest user using $\mathcal{F}$. For this, we add the RcvCompleteMalicious interface to $\mathcal{F}$. Further, when Sim receives a message $(\text{Eval}, sid, ssid, U, S, x)$ from $\mathcal{F}$, instead of running the user's code on x according to $\Pi_{\text{OPRF}}^{\mathsf{H}_1,\mathsf{H}_2,f}$, Sim now disregards x and sends the message $(\text{UserInput}, ssid, U)$ to $\mathcal{Z}$ in the name of $\mathcal{F}_{\text{SFE}}^{f}$. Then we distinguish two cases:

– If Sim receives a $\mathcal{F}_{\text{SFE}}^{f}$-message $(\text{ServerInput}, ssid, K')$ from $\mathcal{A}^*$ in the name of some malicious server and a $\mathcal{F}_{\text{SFE}}^{f}$-message $(\text{Output}, ssid)$ from $\mathcal{A}^*$ then Sim sends $(\text{RcvCompleteMalicious}, sid, ssid, U, K')$ to $\mathcal{F}$ to provide U with output.

  The RcvCompleteMalicious message makes $\mathcal{F}$ send $F_{\text{malicious}}(K', x)$ as output to U. The environment can check now that $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K')) = F_{\text{malicious}}(K', x)$. But in $\mathbf{G}_5$ we programmed $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K'))$ exactly to this value.

– If Sim receives a message $(\text{SndrComplete}, sid, ssid, S)$ from $\mathcal{F}$ then Sim sends the message $(\text{ServerInput}, ssid, S, f_{\mathsf{pub}}(K))$ to $\mathcal{A}^*$ in the name of $\mathcal{F}_{\text{SFE}}^{f}$. Note that Sim can compute $f_{\mathsf{pub}}(K)$ as it only depends on K On a $\mathcal{F}_{\text{SFE}}^{f}$-message $(\text{Output}, ssid)$ from $\mathcal{A}^*$ the simulator sends $(\text{RcvCompleteHonest}, sid, ssid, U)$ to $\mathcal{F}$ to provide U with output.

  The RcvCompleteHonest message makes $\mathcal{F}$ send $F_{\text{honest}}(x)$ as output to U. Again, we need to ensure that $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K)) = F_{\text{honest}}(x)$. But in $\mathbf{G}_6$ $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K))$ was programmed exactly to this value. Also note that the number of $(\text{RcvCompleteHonest}, sid, ssid, U)$ messages did not change compared to $\mathbf{G}_6$ as still every honest user output requires Sim to send one RcvCompleteHonest message. Therefore the counter tx is still never exceeded.

Combining the two cases, we get

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

Game $\mathbf{G}_9$: Add Compromise interface. We now add the Compromise interface to the ideal functionality $\mathcal{F}$. If $\mathcal{A}^*$ calls this interface then S is marked as Compromised. When receiving a Compromise message, the simulator now forwards the compromise message to the ideal functionality in addition to revealing K to $\mathcal{A}^*$. Note that after a compromise, $\mathcal{Z}$ knows the honest server's key K and can thus, check if previous evaluations of the honest function's OfflineEval, outputs of users that interacted with S, and outputs of $\mathsf{H}_2(x, f_{\mathsf{sec}}(\mathsf{H}_1(x), K))$ were answered consistently with each other. However, $\mathbf{G}_6$ ensured that all of the above values are sampled from the same table $F_{\text{honest}}(x)$ of $\mathcal{F}$.

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

Game $\mathbf{G}_{10}$: Remove user input for Sim. In this game we take away the additional information about the user inputs x the simulator still gets from $\mathcal{F}$ on every Eval query. We also removed the dummy interfaces that allowed the simulator to make any party output whatever the simulator wanted. As the simulator did not use either anymore, the distribution of the experiment does not change when the simulator does not get this information. Thus, we have

$$\Pr[\mathbf{G}_{10}] = \Pr[\mathbf{G}_9].$$

Note that with these modifications we arrived at $\mathcal{F} = \mathcal{F}^f_{\text{2H-OPRF}}$ and the described simulator works as described in Fig. 10.

□

C Proof of Theorem 3

Game $\mathbf{G}_0$: The real execution. The first game is the adversary $\mathcal{A}^*$ playing with the real protocol depicted in Figure 6 in the $\mathcal{F}_{\text{ZK}}, \mathcal{F}_{\text{VOLE+}}$-hybrid model.

Game $\mathbf{G}_1$: Moving real execution into the simulator. The next game is the environment playing with an ideal functionality that has all code from $\mathcal{F}^{f_{\mathsf{LSeq}}}_{\text{SFE}}$ but also passes all its inputs to a simulator that simulates the execution of the real protocol with an honest server and simulated ideal functionalities $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{VOLE+}}$. $\mathcal{F}^{f_{\mathsf{LSeq}}}_{\text{SFE}}$ also relays protocol outputs of the simulator to the respective parties. That is, the functionality resembles already $\mathcal{F}^{f_{\mathsf{LSeq}}}_{\text{SFE}}$, but it does not guarantee secrecy of inputs yet and its output interface is not yet used by the simulator. These regrouping of code within machines is only syntactical and hence we have

$$\Pr[\mathbf{G}_1] = \Pr[\mathbf{G}_0].$$

Game $\mathbf{G}_2$: Abort upon collision in offset vectors. We let the simulator abort whenever two entries of $\mathbf{l}$ or $\mathbf{l}'$ are the same.

Because f_{LSeq} is sampled uniformly at random from $\mathcal{F}_{\mathsf{LSeq}}$, we can bound the probability that a collision happens using a Birthday Bound. We have $\mathbf{l} \in \mathbb{F}_p^{\ell_{\mathsf{com}}}$ and $\mathbf{l}' \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$, and hence a collision occurs in $\mathbf{l}$ with probability at most $\frac{\ell_{\mathsf{com}}(\ell_{\mathsf{com}}-1)}{2p}$ and in $\mathbf{l}'$ with probability at most $\frac{\ell_{\mathsf{eval}}(\ell_{\mathsf{eval}}-1)}{2p}$. By a Union Bound, we get

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq \frac{\ell_{\mathsf{com}}^2 + \ell_{\mathsf{eval}}^2}{2p}.$$

Game $\mathbf{G}_3$: Simulate $\mathcal{F}_{\mathbf{ZK}}$ without witness in case of a corrupt user. We now change the simulation in case the user is corrupt. Sim does not input $\mathbf{w}$ into $\mathcal{F}_{\mathrm{ZK}}$ anymore but instead just sends COMMITTED to the user. The simulated $\mathcal{F}_{\mathrm{ZK}}$ hence doesn't check if the witness $\mathbf{w}$ satisfies the polynomial constraints $\mathcal{F}$. Instead, Sim lets $\mathcal{F}_{\mathrm{ZK}}$ output $(\textsc{Output}, ssid, \top)$ if the prover and verifier input the same $\mathcal{F}$. Since in the case of an honest server, the polynomial constraints are always satisfied, this does not change the view of the environment $\mathcal{Z}$ and we have

$$\Pr[\mathbf{G}_3] = \Pr[\mathbf{G}_2].$$

Game $\mathbf{G}_4$: Extract from a corrupt user. We change the code of the simulator in case of a corrupt user. Sim takes h, input by the corrutp user into $\mathcal{F}_{\mathrm{VOLE+}}$ and sends it to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$, which replies with $(\mathbf{e}, \mathbf{e}') = f_{\mathsf{LSeq}}(h, K)$. The simulator uses the value of $\mathbf{e}$ instead of computing it himself from the server's input K. This is only a syntactical change and hence we have

$$\Pr[\mathbf{G}_4] = \Pr[\mathbf{G}_3].$$

Game $\mathbf{G}_5$: Simulate $\mathcal{F}_{\mathbf{VOLE+}}$ in case of a corrupt user. In case of a corrupt user, instead of simulating $\mathcal{F}_{\mathrm{VOLE+}}$, Sim picks $\mathbf{o} \in \mathbb{F}_p^{\ell_{\mathsf{eval}}}$ uniformly at random subject to $\left(\frac{o_i}{p}\right) = e_i'$ for all $i \in [\ell_{\mathsf{eval}}]$, and picks $c_{\mathbf{u}}, c_{\mathbf{v}} \in \mathbb{F}_p$ uniformly at random. Sim sends $(\textsc{Output}, \mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$ to $\mathcal{A}^*$. This does not change the distribution of $\mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}}$ so the view of $\mathcal{Z}$ does not change and we have

$$\Pr[\mathbf{G}_5] = \Pr[\mathbf{G}_4].$$

Game $\mathbf{G}_6$: Remove K from simulation. The simulator no longer uses K, so we can change the ideal functionality so that it does not pass K to the simulator anymore.

Game $\mathbf{G}_7$: Abort upon $c_{\mathbf{u}}, c_{\mathbf{v}}$ collision. We let the simulator abort if a corrupt server submits $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}})$ to $\mathcal{F}_{\mathrm{ZK}}$ and $\mathbf{u}', \mathbf{v}', r_{\mathbf{u}}', r_{\mathbf{v}}'$ to $\mathcal{F}_{\mathrm{VOLE+}}$ such that $(r_{\mathbf{u}}, r_{\mathbf{v}}) \neq (r_{\mathbf{u}}', r_{\mathbf{v}}')$ and the following two equations hold:

$$\langle \boldsymbol{\gamma}, \mathbf{u}' \rangle + r_{\mathbf{u}}' = \langle \boldsymbol{\gamma}, (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{u}}$$
$$\langle \boldsymbol{\gamma}, \mathbf{v}' \rangle + r_{\mathbf{v}}' = \langle \boldsymbol{\gamma}, \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}$$

Since $\boldsymbol{\gamma}$ is chosen at random from $\mathbb{F}_p$ after the adversary is committed to all other values, the probability for both collisions to happen is bounded by $1/p$. We hence have

$$|\Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6]| \leq 1/p.$$

Game $\mathbf{G}_8$: Functionality produces user output for honest sessions. We change the simulator to send $(\textsc{Output}, ssid)$ to $\mathcal{F}_{\text{SFE}}^{f_{\mathsf{LSeq}}}$ upon $\mathcal{A}^*$ sending $(\textsc{Output}, ssid)$ to $\mathcal{F}_{\text{ZK}}$ for an honest session, i.e., both U and S are honest. Note that this results in $\mathcal{F}_{\text{SFE}}^{f_{\mathsf{LSeq}}}$ sending output $f_{\mathsf{LSeq}}(h, K)$ to U, for h being input by the user and K being input by the server. It follows from the correctness of the protocol that the output is indistinguishable from the view of $\mathcal{Z}$ and we have

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

Game $\mathbf{G}_9$: No output for mismatching $\mathcal{F}$ in $\mathcal{F}_{\mathbf{VOLE+}}$. We terminate the simulation in case of a corrupt server sending $\mathcal{F}$ to $\mathcal{F}_{\text{ZK}}$ such that not all of the following hold, where $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \boldsymbol{\gamma}$ denote the input and output of the server to $\mathcal{F}_{\text{VOLE+}}$:

$$\mathcal{F} = \{f_u, f_v, f_e^{(1)}, \ldots, f_e^{(\ell_{\mathsf{com}})}\} \text{ where} \tag{13}$$
$$f_u(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \boldsymbol{\gamma}, (X \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{Y}^2 * \text{rot}(\mathbf{Y}^2) \rangle\,, \tag{14}$$
$$f_v(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \boldsymbol{\gamma}, \mathbf{Y}^2 * \text{rot}(\mathbf{Y}^2) \rangle\,, \tag{15}$$
$$f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i) Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\mathsf{com}}]\,, \tag{16}$$
$$c_{\mathbf{u}} = \langle \gamma, \mathbf{u} \rangle + r'_{\mathbf{u}}, c_{\mathbf{v}} = \langle \gamma, \mathbf{v} \rangle + r'_{\mathbf{v}}\,. \tag{17}$$

In $\mathbf{G}_8$, the honest user computed $c_{\mathbf{u}}, c_{\mathbf{v}}$ as in equation (13) and received $\boldsymbol{\gamma}$ as output. Because $\mathcal{F}_{\text{ZK}}$ does not send status if the polynomials of both parties differ, the user in $\mathbf{G}_8$ did not produce any output in case $\mathcal{F}$ submitted by the server does not satisfy (13)-(17). We hence have

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

Game $\mathbf{G}_{10}$: No output for mismatching $(r_{\mathbf{u}}, r_{\mathbf{v}}), (r'_{\mathbf{u}}, r'_{\mathbf{v}})$. We terminate the simulation in case a corrupt server inputs $r_{\mathbf{u}}, r_{\mathbf{v}}$ into $\mathcal{F}_{\text{ZK}}$ and $r'_{\mathbf{u}}, r'_{\mathbf{v}}$ into $\mathcal{F}_{\text{VOLE+}}$ such that these tuples differ. I.e., we only continue the simulation if

$$(r_{\mathbf{u}}, r_{\mathbf{v}}) = (r'_{\mathbf{u}}, r'_{\mathbf{v}}). \tag{18}$$

Let $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \mathbf{u}', \mathbf{v}', r'_{\mathbf{u}}, r'_{\mathbf{v}}, \boldsymbol{\gamma})$ denote the values sent and seen by the corrupt server. In $\mathbf{G}_9$, because of game $\mathbf{G}_7$ we know that either

$$\langle \boldsymbol{\gamma}, \mathbf{u}' \rangle + r'_{\mathbf{u}} \neq \langle \boldsymbol{\gamma}, (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{u}}, \text{ or}$$
$$\langle \boldsymbol{\gamma}, \mathbf{v}' \rangle + r'_{\mathbf{v}} \neq \langle \boldsymbol{\gamma}, \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}.$$

Consequently, the user in $\mathbf{G}_9$ will abort because either the first (f_u) or second (f_v) polynomial relation of $\mathcal{F}_{\mathrm{ZK}}$ will not verify. We hence have

$$\Pr[\mathbf{G}_{10}] = \Pr[\mathbf{G}_9].$$

Game $\mathbf{G}_{11}$: No output for invalid witness. We terminate the simulation in case a corrupt server submits $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \mathbf{u}, \mathbf{v}, \boldsymbol{\gamma}, \mathbf{e})$ (as of $\mathbf{G}_{10}$, we only have to consider corrupt servers sending the same randomness $r_{\mathbf{u}}, r_{\mathbf{v}}$ to both hybrid functionalities, hence we only list it once) such that not all of the following hold:

$$(K + l_i)s_i^2 = E(e_i) \text{ for all } i \in [\ell_{\mathsf{com}}]\,, \tag{19}$$
$$\mathbf{v} = \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,, \tag{20}$$
$$\mathbf{u} = (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,. \tag{21}$$
$$\tag{22}$$

In $\mathbf{G}_{10}$, if any of the above equations does not verify, $\mathcal{F}_{\mathrm{ZK}}$ does not output status and hence if the simulation termines in $\mathbf{G}_{11}$, the user does not produce an output in $\mathbf{G}_{10}$ as well. We have

$$\Pr[\mathbf{G}_{11}] = \Pr[\mathbf{G}_{10}].$$

The simulator of $\mathbf{G}_{11}$ now handles all the cases where $\mathcal{F}_{\mathrm{ZK}}$ does not output status, which happens either because a corrupt server sends polynomials that do not match his $\mathcal{F}_{\mathrm{VOLE+}}$ inputs ($\mathbf{G}_9$), sends mismatching $r_{\mathbf{u}}, r_{\mathbf{v}}$ values to the hybrid functionalities ($\mathbf{G}_{10}$), or sending a non-witness to $\mathcal{F}_{\mathrm{ZK}}$ ($\mathbf{G}_{11}$).

Game $\mathbf{G}_{12}$: No output if any $a_i = 0$. We do not let the simulated user check for zero entries in $\mathbf{o}$ anymore. Instead, we terminate the simulation at the point where the simulated user receives status in case a corrupt server previously submitted $\mathbf{a}$ to $\mathcal{F}_{\mathrm{ZK}}$ with a 0 component, i.e., we only continue if

$$a_i \neq 0 \text{ for all } i \in [\ell_{\mathsf{eval}}]\,. \tag{23}$$

As of $\mathbf{G}_{11}$, we know that $\mathbf{o} = \mathbf{u} + h\mathbf{v} = (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) + h\mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) = (K \cdot \mathbf{1}_k + h + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)$. Because the l'_i are pairwise different, we have

$a_i = 0 \iff \mathbf{a}_i^2 = \mathrm{rot}(\mathbf{a}^2)_{i+1} = 0 \iff \mathbf{o}_i = \mathbf{o}_{i+1} = 0$. Consequently, the simulator terminates in this game if and only if the user aborts in $\mathbf{G}_{11}$ because of the $\mathbf{o}$ check, and we have

$$\Pr[\mathbf{G}_{12}] = \Pr[\mathbf{G}_{11}].$$

Game $\mathbf{G}_{13}$: No output if public key e is wrong. We skip the recomputatino of f_{LSeq} in case of a zero in $\mathbf{e}$. Instead, we terminate the simulation after the honest user received status from $\mathcal{F}_{\mathrm{ZK}}$ in case a corrupt server has sent and seen values $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \mathbf{u}, \mathbf{v}, \boldsymbol{\gamma}, \mathbf{e})$ where $\mathbf{e}$ is not honestly generated from K. I.e., the simulator only continues if

$$\mathbf{e} = \{\left(\frac{K + l_i}{p}\right)\}_{i \in [\ell_{\mathsf{com}}]}. \tag{24}$$

We show that the check in $\mathbf{G}_{12}$ passes if and only if equation 24 holds. First, we show that if the equation is violated, the check failed in $\mathbf{G}_{12}$. As of $\mathbf{G}_{11}$ we know that $(K + l_i)s_i^2 = E(e_i)$ for all $i \in [\ell_{\mathsf{com}}]$. If $s_i \neq 0$ for all $i \in [\ell_{\mathsf{com}}]$ we hence have $\left(\frac{K+l_i}{p}\right) = e_i$ for all $i \in [\ell_{\mathsf{com}}]$. Hence, the above equation can only be violated by a corrupt server choosing $K \neq -l_i, s_i = 0$ for some $i \in [\ell_{\mathsf{com}}]$, allowing him to succeed in the zero knowledge proof but "hide" the i-th entry $\left(\frac{K+l_i}{p}\right)$ of $\mathbf{e}$ by setting $e_i = 0$. In that case, the user in $\mathbf{G}_{12}$ finds an i with $e_i = 0$, computes $f_{\mathsf{LSeq}}(h, -l_i) \neq (\mathbf{e}, \mathbf{e}')$, and aborts.

We now show that if equation (24) holds, the check in $\mathbf{G}_{12}$ passes. Because the l_i are all different, we have $e_i = 0$ for at most one $i \in [\ell_{\mathsf{com}}]$ and, if so, $K = -l_i$. As of $\mathbf{G}_{11}$ we have $\mathbf{o} = \mathbf{u} + h\mathbf{v} = (K \cdot \mathbf{1}_k + h + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)$ and hence $f_{\mathsf{LSeq}}(h, -l_i) = (\{\left(\frac{-li+l_j}{p}\right)\}_{j \in [\ell_{\mathsf{com}}]}, \{\left(\frac{o_i}{p}\right)\}_{j \in [\ell_{\mathsf{eval}}]})$, i.e., the check passes.

Overall, it follows that

$$\Pr[\mathbf{G}_{13}] = \Pr[\mathbf{G}_{12}].$$

Game $\mathbf{G}_{14}$: Functionality generates attacked user's output. In this game, we let the simulator send $(\textsc{Output}, sid)$ to the functionality when simulating an honest user facing a corrupt server, where the user has received status from $\mathcal{F}_{\mathrm{ZK}}$. Because the simulated user of $\mathbf{G}_{13}$ does not perform any abort checks on its own anymore, the environment sees an output of the honest user in $\mathbf{G}_{14}$ if and only it sees an output in $\mathbf{G}_{13}$. It is left to argue that this output looks the same in both games.

The user receives $\mathbf{o} = \mathbf{u} + h\mathbf{v}$ from $\mathcal{F}_{\mathrm{VOLE+}}$ which, because of equations (20),(21),(18) is equal to $(K + h + l'_i)(\mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2))_i$ for all $i \in [\ell_{\mathsf{eval}}]$. Because (23) $a_i = 0$ for all $i \in [\ell_{\mathsf{eval}}]$ we have $\left(\frac{o_i}{p}\right) = \left(\frac{K+h+l'_i}{p}\right)$ for all $i \in [\ell_{\mathsf{eval}}]$, i.e., if the user generates output in this game and the previous one, it is the same.

On $(\textsc{UserInput}, sid, U)$ from $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$, send $(\textsc{VfInput}, sid, U)$.

On $(\textsc{ServerInput}, sid, S, \bar{\mathbf{e}})$ from $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$, store $(sid, \bar{\mathbf{e}})$ and send $(\textsc{PrvInput}, sid, S)$ to $\mathcal{A}^*$.

On $\mathcal{A}^*$ sending $\mathbf{w}$ to $\mathcal{F}_{\mathrm{ZK}}$ on behalf of a corrupt server:
- Parse $\mathbf{w} := (K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}})$, send $(\textsc{ServerInput}, sid, K)$ to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$, retrieve $(\textsc{ServerInput}, sid, S)$. Store $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}})$.
- Send $(\textsc{PrvInput}, sid, S)$ to $\mathcal{A}^*$.

On $\mathcal{A}^*$ sending $(\textsc{VfInput}, sid)$ to $\mathcal{F}_{\mathrm{ZK}}$ on behalf of a corrupt user, send $(\textsc{VfInput}, sid, U)$ to $\mathcal{A}^*$.

On $(\textsc{Committed}, sid)$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathrm{ZK}}$:
- If U is honest, choose $\boldsymbol{\gamma} \xleftarrow{\$} \mathbb{F}_p^n$. If S is corrupt, append $\boldsymbol{\gamma}$ to the record $(K, \dots)$ stored previously, otherwise append it to record $(sid, \mathbf{e})$.
- If U is honest, send $(\textsc{UserInput}, sid, \boldsymbol{\gamma}, U)$ to $\mathcal{A}^*$.
- If S is honest, send $(\textsc{ServerInput}, sid, S)$ to $\mathcal{A}^*$.

On $\mathcal{A}^*$ sending $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$ to $\mathcal{F}_{\mathrm{VOLE+}}$ on behalf of a corrupt server:
- Retrieve $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \boldsymbol{\gamma})$ and append $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$ to the record.
- Abort if $(r_{\mathbf{u}}, r_{\mathbf{v}}) \neq (r'_{\mathbf{u}}, r'_{\mathbf{v}})$, $\langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r'_{\mathbf{u}} = \langle \boldsymbol{\gamma}, (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{u}} \boldsymbol{\gamma}$ and $\langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r'_{\mathbf{v}} = \langle \boldsymbol{\gamma}, \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}$.
- Send $(\textsc{ServerInput}, sid, S)$ to $\mathcal{A}^*$.

On $\mathcal{A}^*$ sending $(\textsc{UserInput}, sid, h, \boldsymbol{\gamma})$ to $\mathcal{F}_{\mathrm{VOLE+}}$ on behalf of a corrupt user:
- Send $(\textsc{UserInput}, sid, h)$ to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$ and receive back $(\textsc{UserInput}, sid, U)$.
- Send $(\textsc{Output}, sid)$ to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$ and receive back (sid, y). Parse $y := (\mathbf{e}, \mathbf{e}') \in \{-1, 0, 1\}^{\ell_{\mathsf{com}}} \times \{-1, 0, 1\}^{\ell_{\mathsf{eval}}}$. Store $(h, \boldsymbol{\gamma}, \mathbf{e}, \mathbf{e}')$.

On $(\textsc{Output}, sid, P)$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathrm{VOLE+}}$:
- If U is corrupt, retrieve $(h, \gamma, \mathbf{e}, \mathbf{e}')$. Pick $c_{\mathbf{u}}, c_{\mathbf{v}}$ at random, pick $\mathbf{o}$ at random such that $\left(\frac{\mathbf{o}_i}{p}\right) = \left(\frac{\mathbf{e}'_i}{p}\right)$, e.g., set $o_i \leftarrow E(e_i) r_i^2$ where $r_i \in \mathbb{F}_p^{\times}$ is chosen uniformly at random. Append $c_{\mathbf{u}}, c_{\mathbf{v}}, \mathbf{o}$ to the stored data.
- Else, if S is corrupt, retrieve $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \boldsymbol{\gamma}, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}})$ and set $c_{\mathbf{u}} := \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r'_{\mathbf{u}}, c_{\mathbf{v}} := \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r'_{\mathbf{v}}$.
- Else (both parties are honest) retrieve $(ssid, \bar{\mathbf{e}}, \boldsymbol{\gamma})$ and sample $c_{\mathbf{u}} \xleftarrow{\$} \mathbb{F}_p, c_{\mathbf{v}} \xleftarrow{\$} \mathbb{F}_p$.
- If $P = \mathcal{A}^*$, send $(\boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$ to $\mathcal{A}^*$.
- If $P = U$ and U is corrupt, send $(\textsc{Output}, sid, \mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$ to $\mathcal{A}^*$.
- If $P = S$, do:
  - If S is corrupt, send $(\textsc{Output}, sid, \gamma)$ to $\mathcal{A}^*$.
  - If S is honest and U corrupt, send $\mathbf{e}$ to U .
  - If S and U are both honest, send $\bar{\mathbf{e}}$ to U.

On $\mathcal{A}^*$ sending $\mathbf{e}$ to the user:
- If S is corrupt, retrieve $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \boldsymbol{\gamma}, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}})$ and append $\mathbf{e}$.
- If S is honest and $\mathbf{e}$ is adversarially-generated, terminate the simulation of the honest user right after sending $\mathcal{F}$ to $\mathcal{F}_{\mathrm{ZK}}$.

On $\mathcal{A}^*$ sending $\mathcal{F}$ to $\mathcal{F}_{\mathrm{ZK}}$ on behalf of a corrupt server:
- Retrieve $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \boldsymbol{\gamma}, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \mathbf{e})$ and set $c_{\mathbf{u}} \leftarrow \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r_{\mathbf{u}}$, $c_{\mathbf{v}} \leftarrow \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$
- Set $f_u(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \boldsymbol{\gamma}, (X \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$
- Set $f_v(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \boldsymbol{\gamma}, \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$
- Set $f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i) Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\mathsf{com}}]$
- Set $\mathcal{F}_U \leftarrow \{f_u, f_v, f_e^{(1)}, \dots, f_e^{(\ell_{\mathsf{com}})}\}$. If $\mathcal{F}_U = \mathcal{F}$, append 1 to the stored data, otherwise append 0.
- Send $(\textsc{Prove}, sid, \mathcal{F}_U)$ to $\mathcal{A}^*$.

On $\mathcal{A}^*$ sending $\mathcal{F}$ to $\mathcal{F}_{\mathrm{ZK}}$ on behalf of a corrupt user:
- Retrieve record $(h, \boldsymbol{\gamma}, \mathbf{e}, \mathbf{e}', c_{\mathbf{u}}, c_{\mathbf{v}}, \mathbf{o})$
- Set $f_u(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \boldsymbol{\gamma}, (X \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$
- Set $f_v(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \boldsymbol{\gamma}, \mathbf{Y}^2 * \mathrm{rot}(\mathbf{Y}^2) \rangle$
- Set $f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i) Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\mathsf{com}}]$
- Set $\mathcal{F}_S \leftarrow \{f_u, f_v, f_e^{(1)}, \dots, f_e^{(\ell_{\mathsf{com}})}\}$. If $\mathcal{F}_S = \mathcal{F}$, append 1 to the stored data, otherwise append 0.
- Send $(\textsc{Prove}, sid, \mathcal{F}_S)$ to $\mathcal{A}^*$.

On $(\textsc{Output}, sid)$ from $\mathcal{A}^*$ to $\mathcal{F}_{\mathrm{ZK}}$:
- If U corrupt, retrieve record $(h, \boldsymbol{\gamma}, \mathbf{e}, \mathbf{e}', c_{\mathbf{u}}, c_{\mathbf{v}}, \mathbf{o}, b)$ and if $b = 1$, send $(ssid, \top)$ to $\mathcal{A}^*$
- If both parties are honest, send $(\textsc{Output}, sid)$ to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$
- If S is corrupt, retrieve $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, \boldsymbol{\gamma}, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \mathbf{e}, b)$ and terminate if $b = 0$. Otherwise, Sim sends $(\textsc{Output}, ssid)$ to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$ only if

$$
\begin{aligned}
&a_i \neq 0 \text{ for all } i \in [\ell_{\mathsf{eval}}]\,,\\
&(K + l_i) s_i^2 = E(e_i) \text{ for all } i \in [\ell_{\mathsf{com}}]\,,\\
&\mathbf{v} = \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,,\\
&\mathbf{u} = (K \cdot \mathbf{1}_k + \mathbf{l}') * \mathbf{a}^2 * \mathrm{rot}(\mathbf{a}^2)\,,\\
&r_{\mathbf{u}} = r'_{\mathbf{u}} \text{ and } r_{\mathbf{v}} = r'_{\mathbf{v}}\,,\\
&\mathbf{e} = \{\left(\frac{K + l_i}{p}\right)\}_{i \in [\ell_{\mathsf{com}}]}\,.
\end{aligned}
$$

Fig. 11: Simulator for Theorem 3 interacting with $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$ for simulating protocol $\mathit{\Pi}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$ from Fig. 6, including hybrid functionalities $\mathcal{F}_{\mathrm{ZK}}, \mathcal{F}_{\mathrm{VOLE+}}$.

Game $\mathbf{G}_{15}$: Keep user input from simulator. Because the simulation does not depend on an honest user's input h anymore, we can change the functionality to not forward h to the simulator. The functionality of $\mathbf{G}_{15}$ is hence equal to $\mathcal{F}_{\mathrm{SFE}}^{f_{\mathsf{LSeq}}}$, and we have reached the ideal execution through only a syntactical last change, i.e.,

$$\Pr[\mathbf{G}_{15}] = \Pr[\mathbf{G}_{14}].$$

This concludes our proof. The simulator of the ideal execution $\mathbf{G}_{15}$ is depicted in Figure 11.

D VOLE$^+$

We state the ideal functionality $\mathcal{F}_{\mathrm{sVOLE}}$ of Roy [42] that our protocol for VOLE$^+$ relies on in Figure 12. Our protocol Π_{VOLE^+} for realizing the VOLE$^+$ functionality is described in Fig. 13.

Theorem 4. *The protocol $\Pi^{p,\ell,k}_{VOLE^+}$ of Fig. 13 UC-realizes $\mathcal{F}_{VOLE+}$ in the $\mathcal{F}_{sVOLE}$-hybrid model if $k > (\log p + 2s)/\log|S_\Delta|$, where s is a statistical security parameter, assuming secure and authenticated channels. More precisely, for every adversary there is an efficient simulator such that the view of $\mathcal{Z}$ in the ideal world is statistically close to its view in the real world with statistical distance bounded by $2^{-s} + \binom{k}{2}p^{-1}$.*

Proof (sketch).
Both parties are honest. Since we assume secure channels, and since the lengths and the number of messages sent by honest parties is independent of the parties' inputs, the case where both the sender and receiver are honest can be simulated by running the real protocol with dummy inputs.

Malicious user. To simulate an interaction with a malicious user, the simulator first simulates k runs of the $\mathcal{F}_{\mathrm{sVOLE}}$ protocol. The malicious user is allowed to pick his outputs $\mathbf{o}_i, \Delta_i$, and if he doesn't do this, the simulator picks them at random and sends them to the user. The honest server only uses his inputs to form the last message $(\mathbf{u}', \mathbf{v}')$, so all the other messages can be simulated perfectly by following the protocol. When the adversary sends $\boldsymbol{\lambda}, \Delta'$, the simulator extracts the Client's input $h := \Delta' + \sum_{i=1}^{k} \lambda_i \Delta_i$ and sends $(h, \boldsymbol{\gamma})$ to the $\mathcal{F}_{\mathrm{VOLE+}}$ functionality. (Here we used that a corrupted user is allowed to dictate the $\boldsymbol{\gamma}$ vector used by $\mathcal{F}_{\mathrm{VOLE+}}$.) Let the honest servers' input (which is unknown to the simulator) be $\mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}$, then the simulator receives $\mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}}$ from $\mathcal{F}_{\mathrm{VOLE+}}$, where $\mathbf{o} = \mathbf{u} + h\mathbf{v}$, $c_{\mathbf{u}} = \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + r_{\mathbf{u}}$, and $c_{\mathbf{v}} = \langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$. The final message $(\mathbf{u}', \mathbf{v}')$, is simulated perfectly as follows. To compute $\mathbf{v}'$, sample the first ℓ entries of $\mathbf{v}'$ uniformly at random and set the last coefficient of $\mathbf{v}'$ such that $\langle \boldsymbol{\gamma} \| 1, \mathbf{v}' \rangle = c_{\mathbf{v}} - c_{\mathbf{v}_1}$. To compute $\mathbf{u}'$, set $\mathbf{o}_{\ell+1} := c_{\mathbf{u}} + hc_{\mathbf{v}} - \langle \boldsymbol{\gamma}, \mathbf{o} \rangle$ and

Functionality $\mathcal{F}_{\mathbf{sVOLE}}^{p,n,S_\Delta}$

User input:
On input $(\textsc{UserInput}, sid)$ from U store $\langle \textsc{UserInput}, sid, \bot, \bot \rangle$ and send $(\textsc{UserInput}, sid, U)$ to $\mathcal{A}^*$.

Malicious User input:
On input $(\textsc{UserInput}, sid, \mathbf{o}, \Delta)$ from $\mathcal{A}^*$ with $\mathbf{o} \in \mathbb{F}_p^n$ and $\Delta \in S_\Delta$, if U is not corrupted ignore the message. Otherwise, store $\langle \textsc{UserInput}, sid, \mathbf{o}, \Delta \rangle$.

Server input:
On input $(\textsc{ServerInput}, sid)$ from S store $\langle \textsc{ServerInput}, sid, \bot, \bot \rangle$ and send $(\textsc{ServerInput}, sid, S)$ to $\mathcal{A}^*$.

Malicious Server input:
On input $(\textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v})$ from $\mathcal{A}^*$ with $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^n$, if S is not corrupted ignore the message. Otherwise store $\langle \textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v} \rangle$.

Output:
On input $(\textsc{Output}, sid)$ from $\mathcal{A}^*$, retrieve $\langle \textsc{UserInput}, sid, \mathbf{o}, \Delta \rangle$, $\langle \textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v} \rangle$.

- If U is corrupt, sample $\mathbf{v} \leftarrow\$ \mathbb{F}_p^n$ and set $\mathbf{u} \leftarrow \mathbf{o} - \Delta \cdot \mathbf{v}$
- If S is corrupt, sample $\Delta \leftarrow\$ S_\Delta$ and set $\mathbf{o} \leftarrow \mathbf{u} + \Delta \cdot \mathbf{v}$
- If all are honest, sample $\mathbf{u}, \mathbf{v} \leftarrow\$ \mathbb{F}_p^n$ and $\Delta \leftarrow\$ S_\Delta$ and set $\mathbf{o} \leftarrow \mathbf{u} + \Delta \cdot \mathbf{v}$.

Output $(\textsc{Output}, sid, \mathbf{o}, \Delta)$ to U and $(\textsc{Output}, sid, \mathbf{u}, \mathbf{v})$ to S.

Fig. 12: Ideal functionality for endemic subset VOLE $\mathcal{F}_{\mathrm{sVOLE}}^{p,n,S_\Delta}$, for a prime p, vector length n and a non-empty subset $S_\Delta \subset \mathbb{F}_p$.

Protocol $\Pi^{p,\ell,k}_{\text{VOLE}^+}$

$\textbf{Client}(h \in \mathbb{F}_p)$:		$\textbf{Server}(\mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}})$:
	Call $\mathcal{F}_{\text{sVOLE}}$ functionality k times:	
	$\xleftarrow{\mathbf{o}_i, \Delta_i} \boxed{\mathcal{F}^{p,\ell+1,S_\Delta}_{\text{sVOLE}}} \xrightarrow{\mathbf{u}_i, \mathbf{v}_i}$	
		$\forall i \in [2,k] : \mathbf{v}'_i \leftarrow \mathbf{v}_1 - \mathbf{v}_i$
$\mathbf{o}'_1 \leftarrow \mathbf{o}_1$	$\xleftarrow{\{\mathbf{v}'_i\}_{i\in[2,k]}}$	
$\forall i \in [2,k]$: $\mathbf{o}'_i \leftarrow \mathbf{o}_i + \Delta_i \mathbf{v}'_i$		
$\boldsymbol{\gamma} \leftarrow\$ \mathbb{F}_p^\ell$	$\xrightarrow{\boldsymbol{\gamma}}$	$\forall i \in [k] : c_{\mathbf{u}_i} \leftarrow \langle \boldsymbol{\gamma} \Vert 1, \mathbf{u}_i \rangle$
		$c_{\mathbf{v}_1} \leftarrow \langle \boldsymbol{\gamma} \Vert 1, \mathbf{v}_1 \rangle$
	$\xleftarrow{\{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1}}$	
Abort if for any $i \in [k] : \langle \boldsymbol{\gamma} \Vert 1, \mathbf{o}'_i \rangle \neq c_{\mathbf{u}_i} + \Delta_i c_{\mathbf{v}_1}$.		
$\boldsymbol{\lambda} \leftarrow\$ \mathbb{F}_p^k$		
$\Delta' \leftarrow h - \sum_{i=1}^{k} \lambda_i \Delta_i$	$\xrightarrow{\boldsymbol{\lambda}, \Delta'}$	$\mathbf{u}' \leftarrow \mathbf{u} \Vert r_{\mathbf{u}} - \sum_{i=1}^{k} \lambda_i \mathbf{u}_i + \Delta' \mathbf{v}_1$
$\mathbf{o} \leftarrow \sum_{i=1}^{k} \lambda_i \mathbf{o}'_i + \mathbf{u}' + h\mathbf{v}'$	$\xleftarrow{\mathbf{u}', \mathbf{v}'}$	$\mathbf{v}' \leftarrow \mathbf{v} \Vert r_{\mathbf{v}} - \mathbf{v}_1$
$c_{\mathbf{v}} \leftarrow c_{\mathbf{v}_1} + \langle \boldsymbol{\gamma} \Vert 1, \mathbf{v}' \rangle$		
$c_{\mathbf{u}} \leftarrow \langle \boldsymbol{\gamma} \Vert 1, \mathbf{o} \rangle - h \cdot c_{\mathbf{v}}$		
Drop the last entry of $\mathbf{o}$.		
output: $(\mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}})$		**output:** $\boldsymbol{\gamma}$

Fig. 13: Protocol $\Pi^{p,\ell,k}_{\text{VOLE}^+}$ for VOLE$^+$ for vectors over $\mathbb{F}_p$ of length ℓ and with k executions of subset VOLE.

$\mathbf{u}' := \mathbf{o}||\mathbf{o}_{\ell+1} - h\mathbf{v}' - \sum_{i=1}^{k} \lambda_i \left(\mathbf{u}_i + \varDelta_i \mathbf{v}_1\right)$.

Malicious server. This is the most interesting case. The simulator first simulates k runs of the $\mathcal{F}_{\text{sVOLE}}$ protocol. The malicious server is allowed to pick his outputs $\mathbf{u}_i, \mathbf{v}_i$, or otherwise the simulator picks them at random and sends them to the server. The simulator obtains $\boldsymbol{\gamma}$ from the VOLE+ functionality. After receiving $\{\mathbf{v}'_i\}_{i\in[2,k]}$, the simulator sends $\boldsymbol{\gamma}$ to the server. Then the simulator receives the check values $\{c_{\mathbf{u}_i}\}_{i\in[k]}$ and $c_{\mathbf{v}_1}$ from the malicious server.

Let $\mathbf{v}'_1 := 0$. The consistency check would cause the honest user to abort unless $\langle \boldsymbol{\gamma}\|1, \mathbf{u}_i + \varDelta_i(\mathbf{v}_i + \mathbf{v}'_i)\rangle = c_{\mathbf{u}_i} + \varDelta_i c_{\mathbf{v}_1}$ for every $i \in [k]$. If the server behaved honestly, all these equations would be satisfied for all values of $\varDelta_i \in S_\varDelta$, however, if the server misbehaves then this doesn't have to be the case. The simulator first checks if all these linear equations have at least one solution in $S_\varDelta$. If this is not the case, then the simulator can abort because he knows the honest user would abort with probability 1 as well. Otherwise, the simulator computes the list G of all indices $i \in [k]$, such that the i-th linear equation has a unique solution $\varDelta_i^* \in S_\varDelta$. Then the simulator aborts with probability $1 - |S_\varDelta|^{-|G|}$ because the honest user would only continue if it happened to be the case that $\varDelta_i = \varDelta_i^*$ for all $i \in G$.

If the simulation continues, then in the real world the adversary learned that $\varDelta_i = \varDelta_i^*$ for all $i \in G$. However, the $k - |G|$ remaining $\varDelta_i$'s are still information-theoretically hidden, which means $\{\varDelta_i\}_{i\in[k]}$ still has $(k - |G|)\log|S_\varDelta|$ bits of min-entropy. It then follows from the leftover hash lemma that, conditioned on all the checks passing, $(\boldsymbol{\lambda}, \varDelta' = h - \sum_{i=1}^{k} \lambda_i \varDelta_i)$ is statistically close to uniform with a statistical distance of at most $2^{-s'}$ with

$$s' = \frac{1}{2}\left((k - |G|)\log|S_\varDelta| - \log p\right) > s - \frac{1}{2}|G|\log|S_\varDelta|\,.$$

Therefore, the simulator can output uniformly random $(\boldsymbol{\lambda}, \varDelta')$, and this only affects the adversary's view by a statistical distance of $2^{-s'}|S_\varDelta|^{-|G|} < 2^{-s}$.

Finally, the server sends the last message $(\mathbf{u}', \mathbf{v}')$. The simulator is done simulating the interaction with the malicious server, but it still needs to extract some adversarial input $(\mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}})$ and send it to $\mathcal{F}_{\text{VOLE+}}$, so that the output of $\mathcal{F}_{\text{VOLE+}}$ is statistically close to the output of the honest user.

If $\mathbf{v}_i + \mathbf{v}'_i \neq \mathbf{v}_j + \mathbf{v}'_j$ for any $i, j \in [k] \setminus G$, then the simulator aborts. This only happens with probability at most $\binom{k}{2}p^{-1}$, because there are only $\binom{k}{2}$ pairs (i, j) and for each pair, if $\mathbf{v}_i + \mathbf{v}'_i \neq \mathbf{v}_j + \mathbf{v}'_j$, then with probability $1 - 1/p$ we have $\langle \boldsymbol{\gamma}\|1, \mathbf{v}_i + \mathbf{v}'_i\rangle \neq \langle \boldsymbol{\gamma}\|1, \mathbf{v}_j + \mathbf{v}'_j\rangle$, which means at most one of these inner products can be equal to the value of $c_{\mathbf{v}_1}$ sent by the server, which means at least one of i or j lies in G. In the non-aborting case, let $\mathbf{v}^* = \mathbf{v}_i + \mathbf{v}'_i$ for all

$i \in [k] \setminus G$. Note that $\langle \boldsymbol{\gamma} \| 1, \mathbf{v}^* \rangle = c_{\mathbf{v}_1}$. The honest user would compute

$$\mathbf{o} \leftarrow \sum_{i=1}^{k} \lambda_i \left(\mathbf{u}_i + \Delta_i \left(\mathbf{v}_i + \mathbf{v}'_i \right) \right) + \mathbf{u}' + h\mathbf{v}' \,,$$

which can be rewritten as

$$\begin{aligned} \mathbf{o} &= \sum_{i \in [k]} \lambda_i \mathbf{u}_i + \sum_{i \in G} \lambda_i \Delta_i^* \left(\mathbf{v}_i + \mathbf{v}'_i - \mathbf{v}^* \right) + \sum_{i \in k} \lambda_i \Delta_i \mathbf{v}^* + \mathbf{u}' + h\mathbf{v}' \\ &= \underbrace{\mathbf{u}' - \Delta' \mathbf{v}^* + \sum_{i \in [k]} \lambda_i \mathbf{u}_i + \sum_{i \in G} \lambda_i \Delta_i^* \left(\mathbf{v}_i + \mathbf{v}'_i - \mathbf{v}^* \right)}_{:= \mathbf{u}^+} + h \underbrace{\left(\mathbf{v}^* + \mathbf{v}' \right)}_{:= \mathbf{v}^+} \,, \end{aligned} \tag{25}$$

where we used that $\Delta_i = \Delta_i^*$ for all $i \in G$, and that $\sum_{i \in [k]} \lambda_i \Delta_i = h - \Delta'$. Except for h, all the values in expression (25) are known to the simulator, so he can compute $\mathbf{u}^+, \mathbf{v}^+$ and parse them as $\mathbf{u} \| r_{\mathbf{u}}$, and $\mathbf{v} \| r_{\mathbf{v}}$ respectively, and send $(\mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}})$ to $\mathcal{F}_{\mathrm{VOLE+}}$. Then it is clear from (25) that the $\mathbf{o}$ output of the honest user agrees with the $\mathbf{o}$ output of $\mathcal{F}_{\mathrm{VOLE+}}$. Moreover, the honest user outputs $c_{\mathbf{v}} := c_{\mathbf{v}_1} + \langle \boldsymbol{\gamma} \| 1, \mathbf{v}' \rangle$, which is indeed equal to $\langle \boldsymbol{\gamma}, \mathbf{v} \rangle + r_{\mathbf{v}}$ because $\langle \boldsymbol{\gamma} \| 1, \mathbf{v}^* \rangle = c_{\mathbf{v}_1}$, so it matches the output in the ideal world. Finally, $c_{\mathbf{u}} = \langle \boldsymbol{\gamma}, \mathbf{o} \rangle - r_{\mathbf{u}} - h \cdot \langle \boldsymbol{\gamma}, \mathbf{v} \rangle$ in both the real and the ideal world.

We make the simplifying assumption that corrupted parties first send all inputs to $\mathcal{F}_{\text{sVOLE}}^{p,n,S_\Delta}$ before receiving any output from it. The simulator can easily be adjusted to other orders of executions, as all k sVOLE runs are independent.
On input ($\textsc{UserInput}, ssid, U$) or ($\textsc{ServerInput}, ssid, S$) from $\mathcal{F}_{\text{VOLE+}}$, forward the message to $\mathcal{A}$.

On ($\textsc{UserInput}, ssid_i, \mathbf{o}_i, \Delta_i)_{i\in[k]}$ from $\mathcal{A}$ on behalf of a corrupt user to $\mathcal{F}_{\text{sVOLE}}^{p,n,S_\Delta}$:
- Store a "user record" $(sid, ssid_1, \dots, ssid_k, \mathbf{o}_1, \dots, \mathbf{o}_k, \Delta_1, \dots, \Delta_k)$

On ($\textsc{ServerInput}, ssid_i, \mathbf{u}_i, \mathbf{v}_i$) from $\mathcal{A}$ on behalf of a corrupt server to $\mathcal{F}_{\text{sVOLE}}^{p,n,S_\Delta}$:
- Store a "server record" $(sid, ssid_1, \dots, ssid_k, \mathbf{u}_1, \dots, \mathbf{u}_k, \mathbf{v}_1, \dots, \mathbf{v}_k)$.

On ($\textsc{Output}, ssid$) from $\mathcal{A}$ to $\mathcal{F}_{\text{sVOLE}}^{p,n,S_\Delta}$:
- If the user is corrupt, retrieve $(sid, ssid_1, \dots, ssid_k, \mathbf{o}_1, \dots, \mathbf{o}_k, \Delta_1, \dots, \Delta_k)$ and send $\mathbf{o}_i, \Delta_i$ to $\mathcal{A}$ for $ssid_i = ssid$.
- If the server is corrupt, retrieve $(ssid, ssid_1, \dots, ssid_k, \mathbf{u}_1, \dots, \mathbf{u}_k, \mathbf{v}_1, \dots, \mathbf{v}_k)$ and send $\mathbf{u}_i, \mathbf{v}_i$ to $\mathcal{A}$ for $ssid_i = ssid$.
- If the user is corrupt and this is the k-th $\textsc{Output}$ message from $\mathcal{A}$, then do:
  - For each $i \in [k]$ sample $\mathbf{v}_i \xleftarrow{\$} \mathbb{F}_p^n$, set $\mathbf{u}_i \leftarrow \mathbf{o}_i - \Delta_i \mathbf{v}_i$
  - For each $i \in [2, k]$ set $\mathbf{v}'_i \leftarrow \mathbf{v}_1 - \mathbf{v}_i$
  - Append $\{\mathbf{v}'_i\}_{i\in[2,k]}$ to the user record
  - Send $\{\mathbf{v}'_i\}_{i\in[2,k]}$ to $\mathcal{A}$ as message of the simulated server.

On $\{\mathbf{v}'_i\}_{i\in[2,k]}$ from $\mathcal{A}$ as message from a corrupt server:
- Choose $\boldsymbol{\gamma} \xleftarrow{\$} \mathbb{F}_p^l$, append $\boldsymbol{\gamma}, \{\mathbf{v}'_i\}_{i\in[2,k]}$ to the server record, and send $\boldsymbol{\gamma}$ to $\mathcal{A}$.

On $\boldsymbol{\gamma}$ from $\mathcal{A}$ as message from a corrupt user:
- Compute $\{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1}$ as the honest server would do and send them back to $\mathcal{A}$.
- Append $\boldsymbol{\gamma}, \{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1}$ to the user record.

On $\{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1}$ from $\mathcal{A}$ as message from a corrupt server:
- Retrieve the server record $(sid, \{ssid_i\}_{i\in[k]}, \{\mathbf{u}_i\}_{i\in[k]}, \{\mathbf{v}_i\}_{i\in[k]})$
- For every $i \in [k]$ let s_i denote the number of solutions $\Delta_i \in S_\Delta$ of equation $\langle \boldsymbol{\gamma} \| 1, \mathbf{u}_i + \Delta_i(\mathbf{v}_i + \mathbf{v}'_i)\rangle = c_{\mathbf{u}_i} + \Delta_i c_{\mathbf{v}_1}$
- If $s_i = 0$ for any $i \in [k]$ then abort the user simulation.
- Let $G := \{i | s_i = 1\}$ and Δ_i^* denote the corresponding unique solution. Abort the user simulation with probability $1 - |S_\Delta|^{-|G|}$.
- Abort the user simulation if $\mathbf{v}_i + \mathbf{v}'_i \neq \mathbf{v}_j + \mathbf{v}'_j$ for any $i, j \in [k] \setminus G$.
- Choose $\boldsymbol{\lambda} \xleftarrow{\$} \mathbb{F}_p^k$, $\Delta' \xleftarrow{\$} \mathbb{F}_p$, append these values, G, $\{\Delta_i^*\}_{i\in G}$ and the received values to the server record and send $\boldsymbol{\lambda}, \Delta'$ to $\mathcal{A}$.

On $\boldsymbol{\lambda}, \Delta'$ from $\mathcal{A}$ as message from a corrupt user:
- Retrieve the user record $(sid, \{ssid_i\}_{i\in[k]}, \{\mathbf{o}_i\}_{i\in[k]}, \{\Delta_i\}_{i\in[k]}, \{\mathbf{v}'_i\}_{i\in[2,k]}, \boldsymbol{\gamma}, \{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1})$
- Compute $h \leftarrow \Delta' + \sum_{i=1}^k \lambda_i \Delta_i$ and send ($\textsc{UserInput}, sid, h, \boldsymbol{\gamma}$) to $\mathcal{F}_{\text{VOLE+}}$ //Extract corrupt user's input
- Send ($\textsc{Output}, sid$) to $\mathcal{F}_{\text{VOLE+}}$ and retrieve ($\textsc{Output}, sid, \mathbf{o}, \boldsymbol{\gamma}, c_{\mathbf{u}}, c_{\mathbf{v}}$).
- Sample $\mathbf{r} \xleftarrow{\$} \mathbb{F}_p^\ell$ and set $\mathbf{v}'_{\ell+1} := c_{\mathbf{v}} - \langle \boldsymbol{\gamma}, \mathbf{r}\rangle - c_{\mathbf{v}_1}$. Set $\mathbf{v}' := \mathbf{r} || \mathbf{v}'_{\ell+1}$
- Set $\mathbf{o}_{\ell+1} := c_{\mathbf{u}} + hc_{\mathbf{v}} - \langle \boldsymbol{\gamma}, \mathbf{o}\rangle$ and set $\mathbf{u}' := \mathbf{o} || \mathbf{o}_{\ell+1} - h\mathbf{v}' - \sum_{i=1}^k \lambda_i (\mathbf{u}_i + \Delta_i \mathbf{v}_1)$
- Send $\mathbf{u}', \mathbf{v}'$ to $\mathcal{A}$.
- Send ($\textsc{Output}, sid, S$) to $\mathcal{F}_{\text{VOLE+}}$.

On $\mathbf{u}', \mathbf{v}'$ from $\mathcal{A}$ as message from a corrupt server:
- Retrieve the server record $(sid, \{ssid_i\}_{i\in[k]}, \{\mathbf{u}_i\}_{i\in[k]}, \{\mathbf{v}_i\}_{i\in[k]}, \{c_{\mathbf{u}_i}\}_{i\in[k]}, c_{\mathbf{v}_1}, \boldsymbol{\gamma}, \{\mathbf{v}'_i\}_{i\in[2,k]}, \boldsymbol{\lambda}, \Delta', G, \{\Delta_i^*\}_{i\in G})$
- Choose an arbitrary $j \in [k] \setminus G$.
- Set $\mathbf{u} || r_{\mathbf{u}} := \mathbf{u}' - \Delta'(\mathbf{v}_j + \mathbf{v}'_j) + \sum_{i\in[k]} \lambda_i \mathbf{u}_i + \sum_{i\in G} \lambda_i \Delta_i^* \left(\mathbf{v}_i + \mathbf{v}'_i - (\mathbf{v}_j + \mathbf{v}'_j)\right)$.
- Set $\mathbf{v} || r_{\mathbf{v}} := \mathbf{v}_j + \mathbf{v}'_j + \mathbf{v}'$.
- Send ($\textsc{ServerInput}, sid, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}$) to $\mathcal{F}_{\text{VOLE+}}$.
- Send ($\textsc{Output}, sid, U$) to $\mathcal{F}_{\text{VOLE+}}$.

Fig. 14: Simulator of Theorem 2.