

The 2Hash OPRF Framework and Efficient Post-Quantum Instantiations

Ward Beullens¹[0000-0003-0888-283X], Lucas Dodgson², Sebastian Faller^{1,2}[0009-0005-4126-3098], and Julia Hesse¹[0000-0002-2875-6198]*

¹ IBM Research Europe – Zurich, Switzerland
wbe@zurich.ibm.com,sebastian.faller@ibm.com,juliahesse2@gmail.com

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 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. A preliminary implementation shows that an execution of our OPRF protocol, instantiated for 128 bits of security, runs in only 185 ms if both parties are running in separate threads on the same machine, with a total communication cost of approximately 748 KB.

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 [47] presented the first

* The author was supported by the Swiss National Science Foundation (SNSF) under the AMBIZIONE grant “Cryptographic Protocols for Human Authentication and the IoT.”

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 $\text{PRF}_K(x)$, 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 [31,38,5,30,34,42]. In particular, OPRFs can be used to amplify the entropy of user passwords pw by computing $\text{PRF}_K(pw)$ with the help of a server, making them a core component of modern password-based protocols [36,9,39,22,21,46]. Billions of users rely on OPRF-based protocols today, e.g., through WhatsApp’s password-protected chat history backups [53,22] or PrivacyPass [51], a Chrome/Firefox extension to replace Captchas. All these real-world systems deploy a discrete-log-based OPRF called *2-Hash Diffie Hellman* (2HashDH) [33]. 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 [33] 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 [34] in the Universal Composability (UC) framework [15], under the one-more Diffie Hellman assumption in the group G , in the random oracle model [34].

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 [17] 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 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. [14,6] 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 [16] 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. [14], who presented two isogeny-based OPRFs. The first was susceptible to an attack [7] and additionally relies on a hardness assumption which has since been shown to be insecure (SIDH) [43]. An updated version of this OPRF was introduced by Basso [6] with countermeasures against the attacks of [7,43] and the recent attacks on SIDH. It is round-optimal, requiring only two rounds of communication for an evaluation. It requires 3 MB of communication 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 [12]. An alternative would be to use the SCALLOP(-HD) group action, for which the acting class group and its relation lattice can be computed more efficiently [23,18], however, the evaluation of this group action is much slower than CSIDH, in part because of the use of higher-dimensional isogenies. Another CSIDH-based OPRF, called OPUS, was introduced in a recent work [32]. 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 [14] it is still only secure in the semi-honest server setting.

A lattice-based OPRF 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. Very recently, two more lattice-based constructions appeared. Albrecht and Gur [4] improve the construction from [3] and achieve 315KB online and 222KB online communication costs. Esgin et al. [25] propose an OPRF from new interactive lattice assumption, proposed in the same work. Their OPRF requires around 136KB of communication. Both papers use a game-based model of OPRF security and not the UC formulation that we use in this work.

Multiple works have investigated using the Dark Matter weak PRF [13] in an MPC setting as an OPRF. Dinur et al. proposed doing so using secret-sharing [24], resulting in an OPRF secure in the semi-honest model which requires pre-processing. Albrecht et al. instead used torus fully homomorphic encryption [2], resulting in a scheme that for 100 bits of security requires 2.5 MB of communication for an evaluation, of which only amortized 10 KB happens during 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. [29] 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. [50] 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 [41,10,40]. Faller et al. [27] 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 $f : \mathcal{I} \times \mathcal{K} \rightarrow \mathcal{Y}$, 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 . Our first contribution is to turn this appealingly simple “MPC” approach into an OPRF with composable security.

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 $f(H_1(x), K)$ through a second random oracle $H_2 : \mathcal{Y} \rightarrow \{0, 1\}^\lambda$.³ We then arrive at the following generic blueprint for building OPRFs with composable security:

- 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 $H_1(x)$,
- 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 $H_2(x, f(H_1(x), K))$.

Because it is inspired by 2HashDH, 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 $\mathcal{F}_{\text{SFE}}^f$, also called the $\mathcal{F}_{\text{SFE}}^f$ -*hybrid model*), what properties f should satisfy for a 2Hash OPRF to be a secure OPRF. 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 achieves “full” OPRF security in the UC framework [33]. More precisely, we introduce a new ideal functionality $\mathcal{F}_{\text{2H-OPRF}}^f$, 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 $\mathcal{F}_{\text{2H-OPRF}}^f$ functionality in the $\mathcal{F}_{\text{SFE}}^f$ -hybrid model. We dare say that one-more unpredictability and weak-key collision resistance are almost sufficient conditions because the $\mathcal{F}_{\text{2H-OPRF}}^f$ functionality is only a slight relaxation of the $\mathcal{F}_{\text{OPRF}}$ functionality: E.g., *for any* f the

³ This is because if the user was to output $f(H_1(x), K)$ 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. [27, Claim 3] shows that even a non-programmable RO is not sufficient.

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.

$\mathcal{F}_{2\text{H-OPRF}}^f$ functionality can be securely plugged into the OPAQUE and PPSS protocols [36,34]. In fact, we are not aware of any application of $\mathcal{F}_{\text{OPRF}}$ which cannot use $\mathcal{F}_{2\text{H-OPRF}}^f$ instead, regardless of the function f . We give a heuristic explanation of why $\mathcal{F}_{2\text{H-OPRF}}^f$ should be “almost” $\mathcal{F}_{\text{OPRF}}$ 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.

Using sequences of Legendre symbols as PRG was already proposed in 1988 by Damgård [20] and since then, Legendre symbols (and higher order residues) have shown to lend themselves particularly well to MPC protocols [50,29,11]. Spurred on by a bounty challenge by the Ethereum foundation, there have been several works analyzing the pseudorandomness of the Legendre PRF or, equivalently, the Decisional Shifted Legendre Symbol (DSLS) assumption [19,41,40,10,45].

We use a function $f_{\text{LSeq}} : \mathbb{F}_p \times \mathbb{F}_p \rightarrow \{-1, 0, 1\}^{\ell_{\text{com}} + \ell_{\text{eval}}}$ defined as

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

where p is a large prime, and $\mathbf{l} \in \mathbb{F}_p^{\ell_{\text{com}}}, \mathbf{l}' \in \mathbb{F}_p^{\ell_{\text{eval}}}$ are randomly sampled vectors. For appropriately chosen parameters $p, \ell_{\text{com}}, \ell_{\text{eval}}$ we can show that this function is one-more unpredictable and key-collision resistant, which means we can use f 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 $\left(\frac{K+h+l'_i}{p}\right)$, we use only $\ell_{\text{eval}} = \lambda$ 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 believe that our construction –as it is– is even a verifiable OPRF, although we leave a formal proof of this to future work.

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 $\mathbf{a} \in (\mathbb{F}_p^\times)_{\text{eval}}^\ell$, and sends $\mathbf{u} = \{(K + l'_i) a_i^2\}_{i \in [\ell_{\text{eval}}]}$ and $\mathbf{v} = \{a_i^2\}_{i \in [\ell_{\text{eval}}]}$ to the VOLE protocol. The user

sends his input h and learns

$$\mathbf{o} := \mathbf{u} + h \cdot \mathbf{v} = \{(K + h + l'_i) a_i^2\}_{i \in [\ell_{\text{eval}}]}.$$

The idea behind masking $(K + h + l'_i)$ with a random nonzero square is that the user learns the Legendre symbol of $K + h + l'_i$ but nothing else. The server can further send 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 $(K, \mathbf{a}) \in \mathbb{F}_p \times (\mathbb{F}_p^\times)^{\ell_{\text{eval}}}$ 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 γ for a universal hash function H_γ as well as the hash values $c_{\mathbf{u}} = H_\gamma(\mathbf{u}) + r_{\mathbf{u}}$ and $c_{\mathbf{v}} = H_\gamma(\mathbf{v}) + r_{\mathbf{v}}$, where $r_{\mathbf{u}}$ and $r_{\mathbf{v}}$ are field elements chosen randomly by the server to avoid leaking information about \mathbf{u} and \mathbf{v} . This is all we need to do to tie the zero-knowledge proof to the VOLE input. The server first commits to $(K, \mathbf{a}, r_{\mathbf{u}}, r_{\mathbf{v}})$, then the VOLE+ protocol is run, and then the server proves that $(K, \mathbf{a}, r_{\mathbf{u}}, r_{\mathbf{v}})$ 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).

This results in a secure evaluation protocol for f_{LSeq} , and hence we obtain an OPRF protocol. The total communication cost of the protocol is approximately 748 KB, and with our preliminary implementation the protocol takes 185 ms to execute when the client and server are each represented by a single thread on a single machine, proving that the protocol is practical. Our implementation is available at <https://github.com/2HashFramework/LegendreOPRF>.

1.3 Organization

We start with preliminaries on Legendre symbols and secure function evaluation, i.e., specifying $\mathcal{F}_{\text{SFE}}^f$, in Section 2. Section 3 presents our 2Hash OPRF framework, including a compiler from $\mathcal{F}_{\text{SFE}}^f$ to 2Hash OPRFs for appropriate

⁵ This is a slight simplification. In our full protocol, we apply an optimization to avoid proving in zero knowledge that $a_i \neq 0$ for all $i \in [\ell_{\text{eval}}]$. Instead of multiplicatively masking $\{(K + l'_i)\}_{i \in [\ell_{\text{eval}}]}$, with $\{a_i^2\}_{i \in [\ell_{\text{eval}}]}$ we mask it with $\mathbf{a}^2 * \text{shift}(\mathbf{a}^2) = (a_1, a_1 a_2, \dots, a_{\ell_{\text{eval}}-1} a_{\ell_{\text{eval}}})$ instead, and we let the client abort if there are at least two zeroes in the masked output \mathbf{o} . The point is that, if the client does not abort, then a_1 up to $a_{\ell_{\text{eval}}-1}$ are guaranteed to be nonzero. Then, we only have to use the zero-knowledge proof to prove that the remaining entry $a_{\ell_{\text{eval}}}$ is nonzero, which we do by proving knowledge of $b \in \mathbb{F}_p$ such that $a_{\ell_{\text{eval}}} b = 1$.

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

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 E. 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 C), and the full proof of our secure function evaluation protocol for the Legendre-based function f_{LSeq} (Appendix D).

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 $\{1, \dots, k\}$. 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 $\mathbf{a} \in (\mathbb{F}_p^\times)^k$, we denote by $\text{shift}(\mathbf{a})$ the vector obtained by shifting the entries of \mathbf{a} one position to the right, shifting in a ‘1’ in the leftmost position, and dropping the rightmost entry of \mathbf{a} . I.e., $\text{shift}(\mathbf{a}) = (1, a_1, \dots, a_{k-1}) \in (\mathbb{F}_p^\times)^k$. We denote by $\mathbf{u} \parallel \mathbf{v}$ 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 $\text{dom}(F)$ 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 $y^2 = x \pmod p$. For a prime number p and natural number x , the *Legendre symbol* $\left(\frac{x}{p}\right)$ is defined as follows:

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

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 by Damgård [20]. Later, the Legendre pseudorandom function with a single-bit output was defined as $\text{PRF}_k(x) = \left(\frac{k+x}{p}\right)$. The security of this PRF is equivalent to the *Decision Shifted Legendre Symbol* (DSLS) problem [29].

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 is to distinguish between \mathcal{O}_{Leg} and \mathcal{O}_R .

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.

(Quantum) Cryptanalysis of DSLS. Classically, the best attacks run in time $\tilde{O}(p/q^2 + q^2)$, where q is the number of queries that the adversary is allowed to make to the \mathcal{O}_{Leg} oracle [10,40]. A polynomial time quantum attack is possible if the adversary is allowed to query \mathcal{O}_{Leg} in superposition [19]. In the more limited setting where queries to \mathcal{O}_{Leg} are classical, quantum attackers have a much smaller advantage since all known quantum attacks have a time complexity of at least $\tilde{O}(p^{1/3})$ [28]. In conclusion, the DSLS problem seems to resist attacks from both classical and quantum adversaries, and can therefore be used as a basis for quantum-safe cryptosystems, as long as the adversary cannot query \mathcal{O}_{Leg} in superposition, which is typically the case [11].

Secure function evaluation. We define a secure function evaluation functionality \mathcal{F}_{SFE}^f that allows leakage of a function of the server’s input. Namely, \mathcal{F}_{SFE}^f allows computation of some function $f := (f_{pub}, f_{sec})$ 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. We will later use \mathcal{F}_{SFE}^f to build OPRF protocols.⁷ The functionality is depicted in Fig. 1.

3 A framework for 2Hash OPRFs

We first recap the definition of OPRFs in the UC framework by Jarecki et al. [36,34], depicted in Figure 2 ignoring the gray parts of the code. To improve the readability, we deviate from the original version of \mathcal{F}_{OPRF} in [34] by dropping prefixes (which are irrelevant for our work), dropping evaluation tickets

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 $f_{pub}(K)$ 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 $f_{pub}(K)$ means that no two keys can have the same tag.

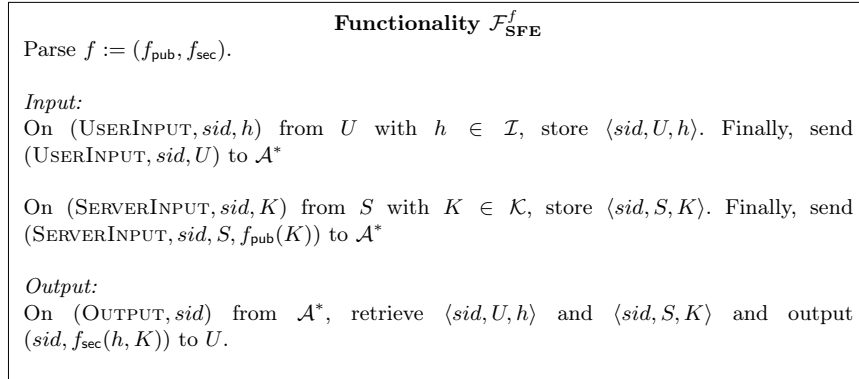


Fig. 1: Ideal secure function evaluation functionality $\mathcal{F}_{\text{SFE}}^f$, parameterized by a function $f = (f_{\text{pub}}, f_{\text{sec}}) : \mathcal{I} \times \mathcal{K} \rightarrow \mathcal{Y}$, where f_{pub} is modeling information leaked about the server’s input.

that dispense with the need for online extraction (also irrelevant for our work) and by splitting up the OFFLINEEVAL and RCVCOMPLETE interfaces into their honest and malicious versions. We start by describing how the functionality works. $\mathcal{F}_{\text{OPRF}}$ models the interaction of multiple users with a single server⁸. The functionality is initialized by a server S sending an INIT message. Then, S is the unique server in this session. The functionality stores a table $F_{\text{honest}}()$ of truly random values representing the OPRF output values of S . The server can always query his “own” PRF through OFFLINEEVAL, while the adversary can query all other function tables $F_{\text{malicious}}(\cdot, K^*)$ 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 COMPROMISE message and then arbitrarily query the server’s table $F_{\text{honest}}()$ through OFFLINEEVAL. We note that the model does capture client compromise, but opposed to server compromise where an OPRF key could leak, a malicious client does not affect the security guarantees of the OPRF. Hence, the code of $\mathcal{F}_{\text{OPRF}}$ does not depend on the corruption state of the client.

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 EVAL message with its input and the server to send a SNDR-COMplete message. The adversary can then send a RCVCOMPLETE message to cause the session to complete. The adversary learns when the parties send their messages and, in the RCVCOMPLETE message, can specify an alternative table $F_{\text{malicious}}(\cdot, K^*)$ from which the user will receive its output. This models the network adversary participating in the role of the server using its own K^* .

⁸ Working in the single-user setting would require every user to agree on a globally unique session id with the server before the protocol, e.g., for domain separation. Hence, we make weaker assumptions on the pre-shared knowledge.

Ideal OPRF functionality

The OPRF functionality is parameterized by a public PRF output length λ and a function $f : \mathcal{I} \times \mathcal{K} \rightarrow \mathcal{Y}$. 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, $\mathcal{F}_{2\text{H-OPRF}}^f$ chooses $r \leftarrow_{\$} \{0, 1\}^\lambda$ and sets $F_i(x) := r$, or $T_{\text{preview}}(x, i) := r$. Similarly, if $H_1(x)$ is referenced for the first time $\mathcal{F}_{2\text{H-OPRF}}^f$ chooses $h \leftarrow_{\$} \mathcal{I}$ and sets $H_1(x) := r$.

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

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 COMPROMISED as well.

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

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

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}^* , record $\langle S, ssid \rangle$.
- On (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 COMPROMISED, ignore this message if there is no record $\langle S, ssid \rangle$
 - Send (EVAL, $sid, ssid, F_{\text{honest}}(x)$) to P .
- On (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 CORRELATE(K^*) and send (EVAL, $sid, ssid, F_{\text{malicious}}(x, K^*)$) to P .

Random oracles:
 On (H_1, sid, x) from \mathcal{A}^* , reply with ($H_1, sid, H_1(x)$).
 On (H_2, sid, x, y^*) from \mathcal{A}^* :
 – For the oldest element K^* in \mathcal{K} that satisfies $f(H_1(x), K^*) = y^*$, 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 CORRELATE(K^*): //Assign previewed values to the $F_{K^*}(\cdot)$ function table

- Append K^* to \mathcal{K} .
- For all $(x, y) \in \text{dom}(T_{\text{preview}}) \setminus T_{\text{programmed}}$ with $f(H_1(x), K^*) = y$:
 - Set $F_{\text{malicious}}(K^*, x) := T_{\text{preview}}(x, y)$
 - Add (x, y) to $T_{\text{programmed}}$

Fig. 2: Ideal OPRF functionalities: $\mathcal{F}_{\text{OPRF}}$ [34] (without gray parts) and $\mathcal{F}_{2\text{H-OPRF}}^f$ (including gray parts).

In Figure 2, by including the gray parts, we define a functionality $\mathcal{F}_{2\text{H-OPRF}}^f$ that is slightly weaker than $\mathcal{F}_{\text{OPRF}}$ but more closely captures the guarantees of 2Hash OPRFs. We stress that $\mathcal{F}_{2\text{H-OPRF}}^f$ is a *dedicated tool* for analyzing 2Hash OPRFs, which we reflect in the functionality’s name. In a nutshell, $\mathcal{F}_{2\text{H-OPRF}}^f$ captures that 2Hash-PRF, i.e., functions of the form $\text{H}_2(x, f(\text{H}_1(x), K))$ can be computed not only from inputs x, K , but also from x, y where $y = f(\text{H}_1(x), K)$. $\mathcal{F}_{2\text{H-OPRF}}^f$ hence incorporates adversarial evaluation interfaces for such function evaluation from pairs x, y . To ensure consistency, $\mathcal{F}_{2\text{H-OPRF}}^f$ is parametrized with a function f and maintains the oracles H_1, H_2 , to automatically “correlate” adversarial function tables $F_{\text{malicious}}(\cdot, K^*)$ with evaluations x, y if $y = f(\text{H}_1(x), K^*)$. We describe these adversarial evaluation interfaces of $\mathcal{F}_{2\text{H-OPRF}}^f$, which are the only interfaces that are added on top of $\mathcal{F}_{\text{OPRF}}$, more formally below.

- $\mathcal{F}_{2\text{H-OPRF}}^f$ is parametrized by a function $f : \mathcal{I} \times \mathcal{K} \rightarrow \mathcal{Y}$ that takes as input a key and a PRF input. f is the core function of the 2Hash PRF.
- $\mathcal{F}_{2\text{H-OPRF}}^f$ maintains a random oracle H_1 , the “inner” random oracle of a 2Hash PRF. It is implemented as a truly random function.
- $\mathcal{F}_{2\text{H-OPRF}}^f$ 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., $f(\text{H}_1(x), K^*) = y$, then $\mathcal{F}_{2\text{H-OPRF}}^f$ “programs” $F_{\text{malicious}}(x, K^*)$ into $\text{H}_2(x, y)$. Otherwise, $\mathcal{F}_{2\text{H-OPRF}}^f$ 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`), $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $y = f(\text{H}_1(x), K^*)$, $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ *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.

In Appendix B we formalize the intuition that the security gap between $\mathcal{F}_{2\text{H-OPRF}}^f$ and $\mathcal{F}_{\text{OPRF}}$ is of no practical relevance. We show that, heuristically, any application of $\mathcal{F}_{\text{OPRF}}$ can be instantiated from $\mathcal{F}_{2\text{H-OPRF}}^f$ as well. The heuristic argument is backed up by our unawareness of any $\mathcal{F}_{\text{OPRF}}$ application in the literature that cannot be proven to be secure using $\mathcal{F}_{2\text{H-OPRF}}^f$. We now provide two prominent examples of how to instantiate $\mathcal{F}_{\text{OPRF}}$ -hybrid protocols from $\mathcal{F}_{2\text{H-OPRF}}^f$.

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

As explained above, $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $\mathcal{F}_{\text{OPRF}}$ that does not work with $\mathcal{F}_{2\text{H-OPRF}}^f$ instead.* To argue this more formally, we first state a lower bound on the security guarantees of $\mathcal{F}_{2\text{H-OPRF}}^f$. Jarecki, Krawczyk, and Xu [37] introduce the correlated OPRF functionality $\mathcal{F}_{\text{corOPRF}}$ as a relaxation of $\mathcal{F}_{\text{OPRF}}$. We render it in Appendix A.

Lemma 1. $\mathcal{F}_{2\text{H-OPRF}}^f$ UC-emulates $\mathcal{F}_{\text{corOPRF}}$.

The proof can also be found in Appendix A. By the UC composition theorem, we can hence replace $\mathcal{F}_{\text{corOPRF}}$ by $\mathcal{F}_{2\text{H-OPRF}}^f$ in applications. In particular, $\mathcal{F}_{2\text{H-OPRF}}^f$ (and its realizations provided in this work) can be plugged into OPAQUE⁹, which can be instantiated from $\mathcal{F}_{\text{corOPRF}}$ [37].

However, Jarecki et al. [37] 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ can be used to instantiate, e.g., the PPSS scheme of Jarecki et al. [34]. Intuitively, this is because $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 ([34], Figure 9) to export $\mathcal{F}_{2\text{H-OPRF}}^f$ ’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.

⁹ We note here that [36,37] consider OPRF functionalities that have a so-called prefix which plays a crucial role in the security proof of OPAQUE. However, we render all functionalities without the prefix. This is justified because one can turn any OPRF that realizes the functionality without the prefix into a functionality with a prefix by setting the prefix to be the full transcript of the protocol. Intuitively, this holds because the prefix is the part of the protocol after which the adversary can no longer “hijack” the protocol execution for a password guess. Clearly, after the last message was delivered the adversary cannot change the output of the protocol execution anymore.

A remark on “tickets”. Many applications of OPRFs in the literature, such as the two above, actually work with a version of $\mathcal{F}_{\text{OPRF}}$ that features “evaluation tickets” [34]. These tickets relax the requirement for online extraction of PRF inputs in the $\mathcal{F}_{\text{OPRF}}$ definition and allow to prove the UC security of efficient instantiations such as 2Hash Diffie Hellman OPRF. Because it is a relaxation, $\mathcal{F}_{\text{OPRF}}$ with tickets is weaker than the standard $\mathcal{F}_{\text{OPRF}}$. Our $\mathcal{F}_{2\text{H-OPRF}}^f$ does not have tickets since our protocol allows for online extraction of user inputs, and hence $\mathcal{F}_{2\text{H-OPRF}}^f$ is suitable to potentially replace both $\mathcal{F}_{\text{OPRF}}$ and “ticketed” $\mathcal{F}_{\text{OPRF}}$ in applications.

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

In this section, we have so far presented and discussed $\mathcal{F}_{2\text{H-OPRF}}^f$, a dedicated functionality for assessing the security of 2Hash OPRFs. We now move on to building such protocols. While the “insecurity” of $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $H_1(x)$
2. User (holding H_1) and server (holding K) engage in a 2-party secure computation of f , letting the user compute $y := f(H_1(x), K)$. Formally, we capture this step by functionality $\mathcal{F}_{\text{SFE}}^f$ (Figure 1).
3. The user computes and outputs $H_2(x, y)$.

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

- (n, q) -One-More Unpredictability. We say \mathcal{F} is (n, q) -one-more unpredictable, if for a random $f = (f_{\text{pub}}, f_{\text{sec}}) \leftarrow \mathcal{F}$ and a random key K , given f , auxiliary information $f_{\text{pub}}(K)$, n uniformly random inputs x_1, \dots, x_n , access to an evaluation oracle for $f_{\text{sec}}(\cdot, K)$, and access to a verification oracle $\mathcal{V}_K^f(i^*, y^*)$ which outputs 1 if $f_{\text{sec}}(x_{i^*}, K) = y^*$, it is computationally hard to find $f_{\text{sec}}(x_{i^*}, K)$ for $q + 1$ distinct $i^* \in [n]$, while making at most q queries to the evaluation oracle.

More precisely, we define the advantage of an adversary \mathcal{A} against the (n, q) -one-more unpredictability property of \mathcal{F} as

$$\text{Adv}_{\mathcal{F}, \mathcal{A}}^{\text{OMU}}(n, q) := \Pr \left[\begin{array}{l} \text{All } i_j \text{ are distinct,} \\ \forall j \in [q+1] : f_{\text{sec}}(x_{i_j}, K) = y_j, \\ \text{and } \mathcal{A} \text{ made at most} \\ q \text{ queries to } f_{\text{sec}}(\cdot, K) \end{array} \middle| \begin{array}{l} f = (f_{\text{pub}}, f_{\text{sec}}) \leftarrow \mathcal{F} \\ K \leftarrow \mathcal{K} \\ x_1, \dots, x_n \leftarrow \mathcal{I}^n \\ \{(i_j, y_j)\}_{j \in [q+1]} \leftarrow \mathcal{A}^{f_{\text{sec}}(\cdot, K), \mathcal{V}_K^f(\cdot, \cdot)}(f, f_{\text{pub}}(K), \mathbf{x}) \end{array} \right].$$

- *n-Weak Key-Collision Resistance.* For uniformly random $x_1, \dots, x_n \in \mathcal{I}$ 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

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

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

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

Public Parameters: function $f = (f_{\text{pub}}, f_{\text{sec}}) \leftarrow \mathcal{F}$, output length λ .

Initialization:
On input (INIT, sid): S picks $K \leftarrow \mathbb{Z}_p$ and stores $\langle sid, K \rangle$

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

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

Online Evaluation:

- On (EVAL, $sid, ssid, S', x$), U sends (USERINPUT, $ssid, \text{H}_1(x)$) to $\mathcal{F}_{\text{SFE}}^f$.
- On (SNDRCOMPLETE, $sid, ssid'$), S sends (SERVERINPUT, $ssid, K$) to $\mathcal{F}_{\text{SFE}}^f$.
- On receiving ($ssid, y$) from $\mathcal{F}_{\text{SFE}}^f$, U outputs (EVAL, $sid, ssid, \text{H}_2(x, y)$).

Fig. 3: The compiler $\Pi_{\text{OPRF}}^{\text{H}_1, \text{H}_2, f}$ in the $\mathcal{F}_{\text{SFE}}^f$ -hybrid model. H_1, H_2 are ROs.

Theorem 1. $\Pi_{\text{OPRF}}^{\text{H}_1, \text{H}_2, f}$ UC-realizes $\mathcal{F}_{2\text{H-OPRF}}^f$ in the $\mathcal{F}_{\text{SFE}}^f$ -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 H_1, H_2 are modeled as random oracles.

More precisely, for every efficient real-world adversary \mathcal{A}^* against $\Pi_{\text{OPRF}}^{\text{H}_1, \text{H}_2, f}$ the ideal-world execution with $\mathcal{F}_{2\text{H-OPRF}}^f$ and the simulator Sim from Fig. 10 is such that for every efficient environment \mathcal{Z} , making n_{H_1} queries to H_1 and giving q SNDRCOMPLETE inputs to the server, there exists an efficient adversary \mathcal{B} against the n_{H_1} -weak key-collision resistance of f and an efficient adversary \mathcal{B}' against the (n_{H_1}, q) -one-more unpredictability of f such that

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

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 $\mathcal{F}_{2\text{H-OPRF}}^f$ by an oblivious evaluation of $\text{PRF}_K(x) := H_2(x, f(H_1(x), K))$ through a 2-party computation of $f(\cdot)$, the client obviously learns *more* about K than just $\text{PRF}_K(x)$. In the proof we need to argue that the intermediate values $f(\cdot, K)$ do not help in breaking the strong pseudorandomness guarantees of $\mathcal{F}_{2\text{H-OPRF}}^f$, 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 $(x_1, f(H_1(x_1), K)), \dots, (x_q, f(H_1(x_q), K))$, where K is the simulated key of the honest server, the adversary is not able to come up with *another* tuple $(x_{q+1}, f(H_1(x_{q+1}), K))$.

Another imperfection of $\text{PRF}_K(x) := H_2(x, f(H_1(x), K))$ is that there could exist key collisions, i.e., two different keys K, K' such that for some input x it holds that $f(H_1(x), K) = f(H_1(x), K')$. With such a collision, a user evaluating 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ signal equality of user inputs, even when facing a malicious server that potentially found a key collision. To prove that an oblivious evaluation of $\text{PRF}_K(x) := H_2(x, f(H_1(x), K))$ UC-emulates $\mathcal{F}_{2\text{H-OPRF}}^f$, 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ is already tailor-made for evaluating the security of oblivious evaluations of functions of the form $\text{PRF}_K(x) := H_2(x, f(H_1(x), K))$, the work of the simulator is actually quite minimal: most queries are simply relayed between the adversary and $\mathcal{F}_{2\text{H-OPRF}}^f$. For example, answers to random oracles H_1, H_2 queries are not chosen by Sim but by $\mathcal{F}_{2\text{H-OPRF}}^f$, and the programming part of these oracles is already hard-coded in the corresponding $\mathcal{F}_{2\text{H-OPRF}}^f$ interfaces through the CORRELATE procedure. Furthermore, using \mathcal{F}_{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 $\mathcal{F}_{2\text{H-OPRF}}^f$ consistently reflects the real function tables $\text{PRF}_K(\cdot)$ for all adversarial keys K . We defer the proof to Appendix C.

4 Instantiation from block ciphers

To instantiate Theorem 1, we need to (a) choose a one-more unpredictable and key-collision resistant function family, and (b) instantiate $\mathcal{F}_{\text{SFE}}^f$ 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}_{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 \mathbf{E} is modeled as an ideal cipher. Together with *Theorem 1* and any maliciously secure multi-party computation for binary circuits, which realizes $\mathcal{F}_{\text{SFE}}^{f_{\mathbf{E}}}$ for $f_{\mathbf{E}} \in \mathcal{F}_{\mathbf{E}}$, this yields a $\mathcal{F}_{\text{2H-OPRF}}^f$ -secure OPRF. Let $\mathbf{E} : \{0, 1\}^{2\lambda} \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ be a block cipher, and let

$$f_{\mathbf{E}} : \{0, 1\}^{2\lambda} \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda} : (h, K) \mapsto (\mathbf{E}_K(h)).$$

Then we define $\mathcal{F}_{\mathbf{E}} := \{(f_{\text{pub}} = \perp, f_{\text{sec}} = f_{\mathbf{E}})\}$, where by setting $f_{\text{pub}} = \perp$ we denote that the corresponding protocol $\mathcal{F}_{\text{SFE}}^{f_{\mathbf{E}}}$ will have no leakage on the key.

Lemma 2. *If \mathbf{E} is modeled as a PRP then $\mathcal{F}_{\mathbf{E}}$ is one-more unpredictable. More concretely, if \mathcal{A}^* is an adversary for the (n, q_V) -one-more unpredictability game that makes q_V queries to the verification oracle and q_f queries to $f_{\mathbf{E}}(\cdot, K)$, then there exists an adversary against the PRP property of \mathbf{E} with roughly the same running time as \mathcal{A} such that*

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

Proof. Let \mathcal{A}^* be an adversary in the (n, q_V) -one-more unpredictability game. We construct an adversary \mathcal{B} for the PRP game that simulates the unpredictability game for \mathcal{A}^* and outputs 1 if \mathcal{A}^* wins. When \mathcal{A}^* sends a $f_{\mathbf{E}}(\cdot, K)$ query or a $\mathcal{V}_K^f(i, y)$ query to \mathcal{B} , \mathcal{B} uses the oracle provided by the PRP challenger to answer. If \mathcal{B} is interacting with the real cipher $\mathbf{E}_K(\cdot)$, then \mathcal{B} perfectly simulates the one-more unpredictability game for \mathcal{A}^* and thus, outputs 1 with probability $\text{Adv}_{\mathcal{F}, \mathcal{A}^*}^{\text{OMU}}(n, q_V)$.

Suppose the PRP game implements $\mathbf{E}_K(\cdot) = \mathcal{O}_R(\cdot)$ as a truly random permutation. We first consider the event that there is a repeated value in a list x_1, \dots, x_n of random inputs. 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 $f_{\mathbf{E}}(x, K)$ queries then the adversary needs to have guessed at least one $\mathbf{E}_K(x_i)$ value. Let q_V be the number of queries to the \mathcal{V}_K^f oracle and q_f the number of queries to $f_{\mathbf{E}}(\cdot, K)$ that \mathcal{A}^* makes. (Note that $q_V \geq q_f + 1$ as there might be queries on which \mathcal{V}_K^f outputs 0.) The probability of guessing the value of $\mathbf{E}_K(x_i)$ in one attempt is bounded by $1/(2^{2\lambda} - q_V - q_f)$, because $\mathbf{E}_K(x_i)$ could take any value in $\{0, 1\}^{2\lambda}$ except the q_V values that were already tried, and the q_f values that were output by the oracle $f_{\mathbf{E}}(\cdot, K)$ (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

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

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

□

Lemma 3. *If E is modeled as an ideal cipher then \mathcal{F}_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 generality (but increasing Q by 2) that before outputting a key collision x_i, K, K' with $E_K(x_i) = E_{K'}(x_i) = y$ the adversary has queried $E_K(x_i)$ and $E_{K'}(x_i)$.

Let $(K_i, x_i, y_i = E_{K_i}(x_i))$ 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 $E_{K_j}(x_j)$ query resulted in $y_j = E_{K_i}(x_i)$. 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 equally 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 $1 \leq i < j \leq Q+2$. \square

4.1 Estimates of concrete efficiency

We give an estimate for the concrete security of the obtained OPRF. One could instantiate the SFE protocol with an actively secure protocol for garbled circuits, e.g., Wang et al. [52]. Wang et al. report for evaluating AES-128 a running time of 16ms and network traffic of 3.4 MB. Our construction needs a block cipher with block length 2λ , where λ is the security parameter. One could use Rijndael with a block length of 256. We estimate that doubling the block size would roughly double the communication. Also, the 256-bit-block version of Rijndael has 14 rounds instead of 10, leading to an additional factor of 1.4. This amounts to an estimate of 9.5 MB of communication. Our instantiation from the DSLS assumption in Section 5 will achieve much better efficiency.

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 $\mathcal{F}_{\text{SFE}}^f$ 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 $\mathbf{1} \in \mathbb{F}_p^{\ell_{\text{com}}}$ and $\mathbf{1}' \in \mathbb{F}_p^{\ell_{\text{eval}}}$ and defined as follows:

$$f_{\text{LSeq}}^{\mathbf{1}, \mathbf{1}'} : \mathbb{F}_p \times \mathbb{F}_p \rightarrow \{-1, 0, 1\}^{\ell_{\text{com}} + \ell_{\text{eval}}}$$

$$(h, K) \mapsto \left(\left\{ \left(\frac{K + l_i}{p} \right) \right\}_{i \in [\ell_{\text{com}}]}, \left\{ \left(\frac{h + K + l'_i}{p} \right) \right\}_{i \in [\ell_{\text{eval}}]} \right)$$

We then set $f_{\text{pub}}(K)^{\mathbf{1}} := \left(\left(\frac{K + l_1}{p} \right), \dots, \left(\frac{K + l_{\ell_{\text{com}}}}{p} \right) \right)$. We define a family of functions as

$$\mathcal{F}_{\text{LSeq}} := \{(f_{\text{pub}}, f_{\text{sec}} = f_{\text{LSeq}}^{\mathbf{1}, \mathbf{1}'} \mid \mathbf{1} \in \mathbb{F}_p^{\ell_{\text{com}}}, \mathbf{1}' \in \mathbb{F}_p^{\ell_{\text{eval}}}\}.$$

For convenience we will usually write f_{LSeq} instead of $f_{\text{LSeq}}^{\mathbf{1}, \mathbf{1}'}$.¹⁰

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 $f_{\text{LSeq}}(x, K)$ is the same Legendre symbol as the t -th bit of $f_{\text{LSeq}}(x', K)$. Therefore, we must ensure that the adversary can not learn too much about the bits of the sequences $f_{\text{LSeq}}(x_i, K)$ by querying its oracle $f_{\text{LSeq}}(\cdot, K)$ on other inputs. Concretely, we bound in the next lemma the probability that with one query to $f_{\text{LSeq}}(\cdot, K)$ the adversary can learn information about the non-public part of more than one sequence $f_{\text{LSeq}}(x_i, K)$.

Lemma 4. *Let $(x_1, \dots, x_n) \leftarrow_{\$} \mathbb{Z}_p$ and $(l_1, \dots, l_{\ell_{\text{eval}}}) \leftarrow_{\$} \mathbb{Z}_p$. Let $p_{2\text{SeqOverlap}}$ be the probability that there exists an $x^* \in \mathbb{F}_p$ and indices $i \neq j$ such that $\{x^* + l_1, \dots, x^* + l_{\ell_{\text{eval}}}\}$ has a nonempty intersection with both $\{x_i + l_1, \dots, x_i + l_{\ell_{\text{eval}}}\}$ and with $\{x_j + l_1, \dots, x_j + l_{\ell_{\text{eval}}}\}$. We have*

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

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

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

holds. We get

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

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

¹⁰ Note that $\mathbf{1}, \mathbf{1}'$ are uniformly random values. So, one could sample $f_{\text{LSeq}} \leftarrow_{\$} \mathcal{F}_{\text{LSeq}}$ in practice by hashing a fixed string, e.g., the session identifier.

– $(x^* - l_t + l_u, i, j, s, u, t, v)$ 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

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

So, the union bound yields $p_{2\text{SeqOverlap}} \leq (n^2 \cdot (\ell_{\text{eval}} + 1)^4)/8p$. \square

Lemma 5. *The family of Legendre symbol sequences $\mathcal{F}_{\text{LSeq}}$ 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 $\mathcal{F}_{\text{LSeq}}$ 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*

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

Proof. \mathcal{B} chooses inputs $x_1, \dots, x_n \leftarrow_{\$} \mathbb{Z}_p$ and public parameters $\mathbf{1} \leftarrow_{\$} \mathbb{F}_p^{\ell_{\text{com}}}, \mathbf{1}' \leftarrow_{\$} \mathbb{F}_p^{\ell_{\text{eval}}}$. Next, \mathcal{B} queries $l_1, \dots, l_{\ell_{\text{com}}}$ to its real-or-random oracle \mathcal{O}_R to obtain $f_{\text{pub}}(K)$. Then, \mathcal{B} internally runs \mathcal{A} on these inputs and answers all queries that \mathcal{A} does to its oracles $f_{\text{LSeq}}(\cdot, K)$ and $\mathcal{V}_K^f(\cdot, \cdot)$ by forwarding the individual Legendre symbols to its oracle \mathcal{O}_R . More precisely, on a $f_{\text{LSeq}}(\cdot, K)$ -query x by \mathcal{A} , \mathcal{B} sends the queries $x + l'_1, \dots, x + l'_{\ell_{\text{eval}}}$ to \mathcal{O}_R . On receiving the outputs $o_1, \dots, o_{\ell_{\text{eval}}}$ from \mathcal{O}_R , \mathcal{B} gives $y := (o_1, \dots, o_{\ell_{\text{eval}}})$ to \mathcal{A} . Similarly, when \mathcal{A} queries its \mathcal{V}_K^f on input (i, y) then \mathcal{B} sends the queries $x_i + l'_1, \dots, x_i + l'_{\ell_{\text{eval}}}$ to \mathcal{O}_R . On receiving the outputs $o_1, \dots, o_{\ell_{\text{eval}}}$ from \mathcal{O}_R , \mathcal{B} checks if $y = (o_1, \dots, o_{\ell_{\text{eval}}})$ and answers accordingly to \mathcal{A} . Finally, if \mathcal{A} makes $q + 1$ distinct queries where the check $y = (o_1, \dots, o_{\ell_{\text{eval}}})$ holds, \mathcal{B} outputs 1. When \mathcal{A} terminates and made less than $q + 1$ verifying queries then \mathcal{B} outputs 0.

Lemma 4 says that the adversary can learn information about at most one sequence $f_{\text{LSeq}}(x_i, K)$ for each query to $f_{\text{LSeq}}(\cdot, K)$ except with probability $(n^2 \cdot \ell_{\text{eval}}^4)/8p$. That means, with overwhelming probability, from the q queries that the adversary makes it can obtain bits of at most q Legendre sequences. But then, to make $q + 1$ successful guesses, the adversary must correctly guess at least one Legendre sequence, say $f_{\text{LSeq}}(x_{i^*}, K)$ about which \mathcal{A} had no information. Note that if \mathcal{O}_R is implemented as a truly random function then \mathcal{A} can guess the input-dependent part of $f_{\text{LSeq}}(x_{i^*}, K)$, i.e., $(\mathcal{O}_R(x_{i^*} + l'_1), \dots, \mathcal{O}_R(x_{i^*} + l'_{\ell_{\text{eval}}}))$ with probability at most $2^{-\ell_{\text{eval}}}$ per guess, so \mathcal{A} wins the unpredictability game with probability at most $q_v 2^{-\ell_{\text{eval}}}$.

In contrast, if \mathcal{O}_R is implemented as $(\frac{\pm K}{p})$ for some uniformly random $K \in \mathbb{Z}_p$ then the view of \mathcal{A} is distributed exactly as in the unpredictability game. So \mathcal{A} succeeds with probability $\mathbf{Adv}_{\mathcal{F}_{\text{LSeq}}, \mathcal{A}}^{\text{OMU}}(n, q)$. The inequality from the theorem statement follows. \square

Before we argue that $\mathcal{F}_{\text{LSeq}}$ has key-collision resistance in Lemma 7, we first show a supporting lemma about Legendre symbols.

Lemma 6. *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 $(p-3)/2$. \square

Lemma 7. *The family of Legendre symbol sequences $\mathcal{F}_{\text{LSeq}}$ as defined in Section 2 is weakly key-collision resistant. More precisely, for any adversary \mathcal{A} we have $\text{Adv}_{\mathcal{F}_{\text{LSeq}}, \mathcal{A}}^{\text{WKCR}}(n) \leq np^2 2^{-\ell_{\text{com}} - \ell_{\text{eval}} - 1}$.*

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

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

Because of Lemma 6, for every tuple (i, K, K') with $K \neq K'$ the probability over the random choice of the offsets $\mathbf{l} \in \mathbb{F}_p^{\ell_{\text{com}}}$ and $\mathbf{l}' \in \mathbb{F}_p^{\ell_{\text{eval}}}$ that (i, K, K') corresponds to a collision is at most

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

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 $np^2 2^{-\ell_{\text{com}} - \ell_{\text{eval}} - 1}$. \square

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

We now instantiate the \mathcal{F}_{SFE} functionality from Figure 1 for $f = f_{\text{LSeq}}$.

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 γ and two check values $c_{\mathbf{u}} = \langle \gamma, \mathbf{u} \rangle + r_{\mathbf{u}}$ and $c_{\mathbf{v}} = \langle \gamma, \mathbf{v} \rangle + r_{\mathbf{v}}$, where $r_{\mathbf{u}}$ and $r_{\mathbf{v}}$ are inputs from the server.

We give an overview over the protocol that realizes the $\mathcal{F}_{\text{VOLE}^+}$ functionality in the $\mathcal{F}_{\text{sVOLE}}$ -hybrid model. We refer to Appendix E for the details of the protocol

and the security proof. The protocol uses the $\mathcal{F}_{\text{sVOLE}}$ (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 [44]. We describe the functionality in Fig. 12. The OT-based protocol of [49] instantiates this functionality with a constant number of rounds and an amount of communication that is independent of ℓ 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 $\mathcal{F}_{\text{VOLE}^+}$ functionality is similar to the protocol of [49,8]. 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 $\lambda \in \mathbb{F}_p^k$ 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 $\gamma \in \mathbb{F}_p^\ell$ and asks the sender to send $c_{\mathbf{u}_i} = \langle \gamma, \mathbf{u}_i \rangle$ for all $i \in [k]$ and $c_{\mathbf{v}_1} = \langle \gamma, \mathbf{v}_1 \rangle$. The receiver checks if $\langle \gamma, \mathbf{o}_i \rangle = c_{\mathbf{u}_i} + \Delta_i \cdot c_{\mathbf{v}_1}$ for all $i \in [k]$. In our protocol, we let the receiver output γ 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 $\langle \gamma, \mathbf{o}_i \rangle = c_{\mathbf{u}_i} + \Delta_i \cdot c_{\mathbf{v}_1}$ 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 [49,8] 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 E we prove the following theorem.

Theorem 2. *The protocol $\Pi_{\text{VOLE}^+}^{p,\ell,k}$ of Fig. 13 UC-realizes $\mathcal{F}_{\text{VOLE}^+}$ in the $\mathcal{F}_{\text{sVOLE}}$ -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 an ideal zero-knowledge functionality in Fig. 5. The functionality $\mathcal{F}_{\text{ZK}}^{p,n}$ 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 $f_1, \dots, f_t \in \mathbb{F}_p[x_1, \dots, x_n]_{\leq d}$, 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 [54] 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 [54] allows the witness to be arbitrarily long, and it can be extended dynamically. Since our functionality is more restrictive, the Quicksilver protocol of [54] securely realizes our \mathcal{F}_{ZK} functionality.

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

Theorem 3. *Let $f_{\text{LSeq}}^{1,1'} \stackrel{\$}{\leftarrow} \mathcal{F}_{\text{LSeq}}$. The protocol $\Pi_{\text{SFE}}^{f_{\text{LSeq}}}$ from Fig. 6 UC-realizes $\mathcal{F}_{\text{SFE}}^{f_{\text{LSeq}}}$ in the $\mathcal{F}_{\text{VOLE+}}, \mathcal{F}_{\text{ZK}}$ -hybrid model.*

The full proof is in Appendix D, and the simulator is given in Figure 11.

Proof sketch. *Malicious Server:* The simulator extracts the client’s input h from its input to $\mathcal{F}_{\text{VOLE+}}$. The simulator then submits h to \mathcal{F}_{SFE} on behalf of the corrupt client, to obtain $(\mathbf{e}, \mathbf{e}') = f_{\text{LSeq}}(h, K)$ for K being the input of the honest server (unknown to the simulator). In addition to \mathbf{e} , the only other values seen by the corrupt client are $\mathbf{o}, c_{\mathbf{u}}$, and $c_{\mathbf{v}}$. Because of the multiplicative masking by $\mathbf{a}^2 * \text{shift}(\mathbf{a}^2)$, the entries of \mathbf{o} are uniformly random field elements with the correct Legendre symbols $\left(\frac{o_i}{p}\right) = e'_i$. Since \mathbf{e}' is known to the simulator, this is easy to simulate. The values $c_{\mathbf{u}}, c_{\mathbf{v}}$ are independent and uniformly random, and therefore they are trivial to simulate.

Malicious Server: During an execution of the real protocol, the environment \mathcal{Z} sees only input confirmation notifications from \mathcal{F}_{ZK} and $\mathcal{F}_{\text{VOLE+}}$ and the $\mathcal{F}_{\text{VOLE+}}$ leakage which is a uniformly random vector $\gamma \in \mathbb{F}_p^{\ell_{\text{eval}}}$. 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. The simulator does this as follows: if the corrupt server sends $\mathbf{w} = (K, \mathbf{a}, \mathbf{s}, \mathbf{u}, r_{\mathbf{u}}, r_{\mathbf{v}}, b)$ to \mathcal{F}_{ZK} , and $(\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}})$ to $\mathcal{F}_{\text{VOLE+}}$, the simulator sends $(\text{OUTPUT}, \text{ssid})$ to $\mathcal{F}_{\text{SFE}}^{f_{\text{LSeq}}}$ if and only if the following hold:

Functionality $\mathcal{F}_{\text{VOLE}^+}^{p,\ell}$
<p><u>Honest user input:</u> On input $(\text{USERINPUT}, \text{sid}, h)$ from U with $h \in \mathbb{F}_p$, sample $\gamma \leftarrow \mathbb{F}_p^\ell$ and store $\langle \text{USERINPUT}, \text{sid}, h, \gamma \rangle$ and send $(\text{USERINPUT}, \text{sid}, \gamma, U)$ to \mathcal{A}^*.</p>
<p><u>Malicious user input:</u> On input $(\text{USERINPUT}, \text{sid}, h, \gamma)$ from \mathcal{A}^* with $h \in \mathbb{F}_p$ and $\gamma \in \mathbb{F}_p^\ell$, ignore if U is not corrupt. Otherwise, store $\langle \text{USERINPUT}, \text{sid}, h, \gamma \rangle$.</p>
<p><u>Server input:</u> On input $(\text{SERVERINPUT}, \text{sid}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}})$ from S with $\mathbf{u}, \mathbf{v} \in \mathbb{F}_p^\ell$ and $r_{\mathbf{u}}, r_{\mathbf{v}} \in \mathbb{F}_p$ store $\langle \text{SERVERINPUT}, \text{sid}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}} \rangle$ and send $(\text{SERVERINPUT}, \text{sid}, U)$ to \mathcal{A}^*.</p>
<p><u>Output:</u> On input $(\text{OUTPUT}, \text{sid}, P)$ from \mathcal{A}^* with $P \in \{\mathcal{A}^*, U, S\}$, retrieve $\langle \text{USERINPUT}, \text{sid}, h, \gamma \rangle$, $\langle \text{SERVERINPUT}, \text{sid}, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}} \rangle$ and do:</p> <ul style="list-style-type: none"> – Compute $\mathbf{o} \leftarrow \mathbf{u} + h\mathbf{v}$, $c_{\mathbf{u}} = \langle \gamma, \mathbf{u} \rangle + r_{\mathbf{u}}$, and $c_{\mathbf{v}} = \langle \gamma, \mathbf{v} \rangle + r_{\mathbf{v}}$ – If $P = \mathcal{A}^*$, output $(\text{OUTPUT}, \text{sid}, \gamma, c_{\mathbf{u}}, c_{\mathbf{v}})$ to \mathcal{A}^* – If $P = U$, output $(\text{OUTPUT}, \text{sid}, \mathbf{o}, \gamma, c_{\mathbf{u}}, c_{\mathbf{v}})$ to U – If $P = S$, output $(\text{OUTPUT}, \text{sid}, \gamma)$ to S.

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

Functionality $\mathcal{F}_{\text{ZK}}^{p,n}$
<p><u>Input witness:</u> On input $(\text{VFINPUT}, \text{sid})$ from \mathcal{V} store $\langle \text{VFINPUT}, \text{sid}, \mathcal{V} \rangle$ and send $(\text{VFINPUT}, \text{sid}, \mathcal{V})$ to \mathcal{A}^*.</p>
<p>On input $(\text{PRVINPUT}, \text{sid}, \mathbf{w})$ from \mathcal{P} with $\mathbf{w} \in \mathbb{F}_p^n$, store $\langle \text{PRVINPUT}, \text{sid}, \mathcal{P}, \mathbf{w} \rangle$ and send $(\text{PRVINPUT}, \text{sid}, \mathcal{P})$ to \mathcal{A}^*.</p>
<p>On input $(\text{COMMITTED}, \text{sid})$ from \mathcal{A}^*, if there are records $\langle \text{VFINPUT}, \text{sid}, \mathcal{V} \rangle$ and $\langle \text{PRVINPUT}, \text{sid}, \mathcal{P}, \mathbf{w} \rangle$, send $(\text{COMMITTED}, \text{sid})$ to \mathcal{V} and \mathcal{P}.</p>
<p><u>Prove polynomial constraints:</u> On input $(\text{PROVE}, \text{sid}, \{f_i\}_{i \in [t]})$ from \mathcal{V} store $\langle \text{PROVE}, \text{sid}, \mathcal{V}, \{f_i\}_{i \in [t]} \rangle$ and send $(\text{PROVE}, \text{sid}, \{f_i\}_{i \in [t]})$ to \mathcal{A}^*.</p>
<p>On input $(\text{PROVE}, \text{sid}, \{f'_i\}_{i \in [t']})$ from \mathcal{P} store $\langle \text{PROVE}, \text{sid}, \mathcal{P}, \{f'_i\}_{i \in [t']} \rangle$ and send $(\text{PROVE}, \text{sid}, \{f'_i\}_{i \in [t']})$ to \mathcal{A}^*.</p>
<p>On input $(\text{OUTPUT}, \text{sid})$ from \mathcal{A}^*, retrieve $\langle \text{PRVINPUT}, \text{sid}, \mathcal{P}, \mathbf{w} \rangle$, $\langle \text{PROVE}, \text{sid}, \mathcal{V}, \{f_i\}_{i \in [t]} \rangle$, and $\langle \text{PROVE}, \text{sid}, \mathcal{P}, \{f'_i\}_{i \in [t']} \rangle$. 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.</p>

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

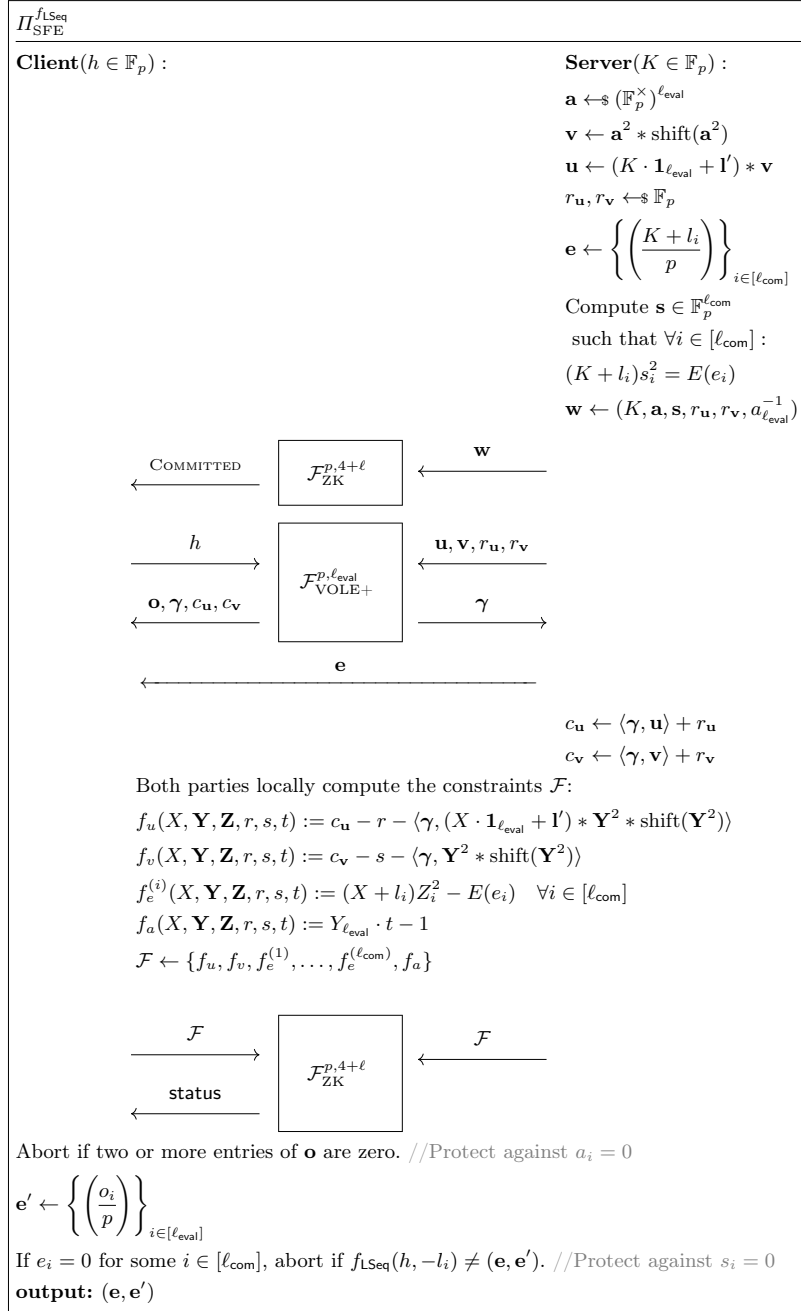


Fig. 6: Secure evaluation of f_{LSeq} for parameters $p, \ell_{\text{com}}, \ell_{\text{eval}} > 2$ and public offsets $\mathbf{l} \in \mathbb{F}_p^{\ell_{\text{com}}}, \mathbf{l}' \in \mathbb{F}_p^{\ell_{\text{eval}}}$. For readability, we simplify the messages to and from the \mathcal{F}_{ZK} and $\mathcal{F}_{\text{VOLE}+}$ 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}_{ZK} .

$$\begin{aligned}
a_i &\neq 0 \text{ for all } i \in [\ell_{\text{eval}}], & \mathbf{v} &= \mathbf{a}^2 * \text{shift}(\mathbf{a}^2), & (4) \\
e_i &= \left(\frac{K + l_i}{p}\right) \text{ for all } i \in [\ell_{\text{com}}], & \mathbf{u} &= (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{l}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2), & (5) \\
(K + l_i)s_i^2 &= E(e_i) \text{ for all } i \in [\ell_{\text{com}}], & r_{\mathbf{u}} &= r'_{\mathbf{u}} \text{ and } r_{\mathbf{v}} = r'_{\mathbf{v}}, & (6) \\
a_{\ell_{\text{eval}}-1}b &= 1, & \mathcal{F} &= \{f_u, f_v, f_e^{(1)}, \dots, f_e^{(\ell_{\text{com}})}, f_a\}. & (7)
\end{aligned}$$

Now we argue why this is a good simulator. We show in the proof in Appendix D that, conditioned on the adversary sending $\mathbf{w} = (K, \mathbf{a}, \mathbf{s}, \mathbf{u}, r_{\mathbf{u}}, r_{\mathbf{v}}, b)$ and $(\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}})$, and the simulator or verifier choosing γ , the output of $\mathcal{F}_{\text{SFE}}^{\text{fLSeq}}$ is the same in the real world and the ideal world, except if (4), (5) or (6) is not satisfied but still

$$\begin{aligned}
\langle \gamma, \mathbf{u} - (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{l}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle &= r_{\mathbf{u}} - r'_{\mathbf{u}}, \text{ and} & (8) \\
\langle \gamma, \mathbf{v} - \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle &= r_{\mathbf{v}} - r'_{\mathbf{v}}. & (9)
\end{aligned}$$

Because in this case, in the ideal world the simulator will abort, but in the real world the zero-knowledge proof will be accepted and so the honest verifier will not abort. Luckily, this case occurs with probability at most $1/p$ because γ 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 (4) or (5) does not hold then one of the left-hand sides is uniformly random, which means the equation holds with probability at most $1/p$. Otherwise, if (4) and (5) hold but (6) 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. Either way, the distinguishing advantage of \mathcal{Z} for telling apart the ideal and the real world is bounded by $1/p$. \square

Combining Theorem 1, Theorem 3, Lemma 5 and Lemma 7, using the composition theorem of the UC framework, we get the following result.

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

Let $n_{\mathbf{H}_1}$ be the number of \mathbf{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 $\Pi_{\text{OPRF}}^{\mathbf{H}_1, \mathbf{H}_2, \text{fLSeq}}$ the ideal-world execution with $\mathcal{F}_{2\text{H-OPRF}}^f$ there exists a simulator Sim such that for every efficient environment \mathcal{Z} there exists*

an efficient DSLS adversary \mathcal{B} such that

$$\begin{aligned} \mathbf{Dist}_{\mathcal{Z}}^{\Pi_{OPRF}^{H_1, H_2, f_{LSeq}}, \{\mathcal{F}_{2H-OPRF}^f, \text{Sim}\}}(\lambda) &\leq \frac{n_{H_1} p^2}{2^{-\ell_{\text{com}} - \ell_{\text{eval}} - 1}} + \mathbf{Adv}_{p, \mathcal{B}}^{\text{DSLS}}(1^\lambda) \\ &\quad + \frac{n_{H_1}^2 \cdot \ell_{\text{eval}}^4}{8p} + \frac{n_{H_2}}{2^{\ell_{\text{eval}}}} + \frac{n_{H_1}}{p}. \end{aligned}$$

To obtain a concrete OPRF protocol from the above corollary, we instantiate the \mathcal{F}_{ZK} functionality with the Quicksilver protocol [54], and we instantiate the $\mathcal{F}_{\text{VOLE}^+}^{p, \ell_{\text{eval}}}$ functionality with our Π_{VOLE^+} protocol from Appendix E, which in turn relies on the subset VOLE functionality $\mathcal{F}_{\text{sVOLE}}$, of which there exist efficient post-quantum realizations from oblivious transfer [49]. 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}_{LSeq} aiming for 128 bits of security and 64 bits of statistical security. We take the following parameters:

- We set $\ell_{\text{eval}} = 128$, since that is the minimum required to achieve unpredictability of \mathcal{F}_{LSeq} .
- The bitsize of the prime p needs to be large enough to ensure the (n, q) -one-more unpredictability of \mathcal{F}_{LSeq} . Lemma 5 reduces the unpredictability of \mathcal{F}_{LSeq} to the DSLS assumption, with an additive loss of $n^2 \ell_{\text{eval}}^4 / 8p$, where n_{H_1} corresponds to the number of queries that the adversary makes to H_1 . We instantiate H_1 with a hash function that is a factor 2^{16} slower than a “usual” cryptographic hash function¹¹. Therefore, to reach 128 bits of security, we need to protect against adversaries that make up to $2^{128-16} = 2^{112}$ calls to H_1 . This allows us to use a 255-bit prime, since $(2^{112})^2 128^4 / (8 \cdot 2^{255}) < 1$. Concretely, we use the prime $p = 2^{255} - 19$. 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 $\tilde{\Omega}(p / (q \ell_{\text{eval}})^2 + q^2)$ [10,40] where q is the number of queries that the adversary makes to the OPRF.
- The ℓ_{com} parameter needs to be large enough to ensure weak collision resistance. Lemma 7 bounds the probability that an adversary finds a key-collision by $p^2 \cdot q_{H_1} \cdot 2^{-\ell_{\text{com}} - \ell_{\text{eval}} - 1}$, so we set $\ell_{\text{com}} = 2 \log p + 112 - \ell_{\text{eval}} - 1 = 493$.
- The parameter $t = \log |S_\Delta|$ used by the sVOLE protocol of [49]. This t controls a trade-off between communication and computational efficiency. Our preliminary implementation uses $t = 8$, as recommended by the author of [49] in a WAN setting.

¹¹ For convenience, we instantiated H_1 by iterating BLAKE2 a total of 2^{16} times, adding a counter for domain separation, although using a memory-hard function might be more appropriate.

Message complexity. If we instantiate the $\mathcal{F}_{\text{VOLE}^+}$ functionality by the protocol of Appendix E, and the \mathcal{F}_{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 $\mathcal{F}_{\text{sVOLE}}$ functionality and the consistency checks used by both the commitment phase of the zero-knowledge proof and the input-independent setup of the VOLE^+ protocol. (In our implementation, we have used OT extension to generate the subset voles from 128 base OTs). Then, four more messages are used to finish the VOLE^+ protocol. The second phase of the zero-knowledge proof can be run in parallel with the last two messages of the VOLE^+ protocol, and so it does not increase the message complexity.

Communication. The main contributions to the communication cost are as follows:

- The cost of the base OT’s. We use OT extension to generate all the OT correlations required by the subset VOLE protocol from 128 base OTs. For the base OTs, our implementation uses the Masny-Rindal OT protocol instantiated with the Kyber KEM [44]. In this protocol, the server sends 2 Kyber public keys, and the client sends 2 Kyber ciphertext, for a total cost of $128 \cdot 2(800 + 768) \text{ B} \approx 392 \text{ KB}$.
- The $1 + \lceil (\log p + 2s)/t \rceil = 49$ vectors of length $\ell_{\text{eval}} + 1 = 129$ over \mathbb{F}_p sent during the execution of the VOLE^+ protocol. With 32 bytes per field element, these vectors have a combined size of 198 KB.
- The $\lceil 64/t \rceil = 8$ vectors of length $\ell_{\text{eval}} + \ell_{\text{com}} + 9 = 630$ sent during the commitment phase of the Quicksilver zero-knowledge proof system. These vectors have a combined size of 158 KB.

This makes for a combined communication cost of approximately 748 KB.

Implementation. We implemented our protocol ¹² using the libOTe library of Rindal and Roy [48]. In particular, we used libOTe’s implementation of the Masny-Rindal OT protocol instantiated with the Kyber KEM [44], the implementation of $(N-1)$ -out-of- N OT (Punctured PRF), and the networking sockets provided by the library. In addition to the protocols developed in this paper, we also implemented the subset VOLE from SoftspokenOT and FEAST [49,8], since no implementation of it modulo large primes was available, and a version of the Quicksilver proof system [54], specialized to the polynomial constraint required by our OPRF protocol. The implementation of our protocol takes 185 ms when the client and server are each represented by a single thread on a single machine with an intel i9-10885H CPU, with a clock speed fixed at 2.4 GHz. This running time should be seen as a lower bound for the running time of the protocol, since in practice, network bandwidth and latency are expected to have a significant effect on the overall running time of the protocol. However, the t parameter

¹² The implementation is available at github.com/2HashFramework/LegendreOPRF.

gives a tradeoff between communication and computation. For example, if we decrease t to 6, or increase it to 10, then the amount of communication becomes 871 KB or 691 KB respectively, while the running time of the protocol without networking delays becomes 178 ms and 313 ms respectively. Finally, we note that the vast majority of the communication and computation is in generating the subset VOLE correlations, which is independent of the OPRF input and key. This can help mitigate the cost of the OPRF protocol. For example, when using OPAQUE to login to an online service, the protocol can start before the user has entered their username and password. After entering the username, the commitment phase of the ZKP can be completed, and after entering the password, the protocol can be finish with only more round trip of communication and minimal computation. So the OPRF should not hurt the user experience.

Conclusion and future work. Our OPRF protocol is 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 $\mathcal{F}_{\text{LSeq}}$ 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}_{SFE} functionality, which would require us to strengthen our main theorem to prove that the weaker \mathcal{F}_{SFE} functionality is still enough to achieve $\mathcal{F}_{\text{2H-OPRF}}^f$. This would be similar to the 2Hash DH OPRF, which uses a very efficient evaluation protocol for a weaker (but still sufficiently strong) version of the \mathcal{F}_{SFE} functionality. We leave the task of exploring these optimizations for future work.

As noted already in Section 2, we believe our Legendre-based OPRF is *verifiable*, i.e., a server can commit to a key by publishing the input-independent part of $f(\text{H}_1(x), K)$, and the client can then efficiently decide whether evaluations are performed with this key, by checking if the \mathcal{F}_{SFE} output agrees with the commitment. The 2HashDH OPRF lends itself not only to verifiability [33] but also to other properties of OPRFs such as, e.g., partially oblivious evaluation [26,9] or threshold evaluation [35]. We leave it for future work to extend our framework to capture such properties of OPRFs, e.g., to add verifiability to $\mathcal{F}_{\text{2H-OPRF}}^f$ and formally prove our compiler to realize it given a key-collision resistant f_{pub} , or to give threshold versions of $\mathcal{F}_{\text{2H-OPRF}}^f$ and $\mathcal{F}_{\text{SFE}}^f$ that allow to lift our compiler to the threshold setting. Our security proofs are in the (Classical) Random Oracle Model, which is not completely satisfactory since we aim to construct quantum-safe OPRFs. Therefore, we consider the security analysis of the 2HashOPRF framework in the Quantum Random Oracle Model as an important open problem.

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, Cham (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 TFHE. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 447–476. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_16
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, Cham (May 2021). https://doi.org/10.1007/978-3-030-75248-4_10
4. Albrecht, M.R., Gür, K.D.: Verifiable oblivious pseudorandom functions from lattices: Practical-ish and thresholdisable. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part IV. LNCS, vol. 15487, pp. 205–237. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0894-2_7
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.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 160–184. Springer, Cham (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, Cham (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., Delpech de Saint Guilhem, C.: LegRoast: Efficient post-quantum signatures from the Legendre PRF. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 130–150. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_8
12. 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, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
13. 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, Cham (Nov 2018). https://doi.org/10.1007/978-3-030-03810-6_25

14. 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, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_18
15. 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>
16. 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>
17. 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, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
18. Chen, M., Leroux, A., Panny, L.: SCALLOP-HD: group action from 2-dimensional isogenies. In: Tang, Q., Teague, V. (eds.) Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14603, pp. 190–216. Springer (2024). https://doi.org/10.1007/978-3-031-57725-3_7, https://doi.org/10.1007/978-3-031-57725-3_7
19. van Dam, W., Hallgren, S.: Efficient quantum algorithms for shifted quadratic character problems. arXiv preprint quant-ph/0011067 (2000)
20. Damgård, I.: On the randomness of Legendre and Jacobi sequences. In: Goldwasser, S. (ed.) CRYPTO’88. LNCS, vol. 403, pp. 163–172. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34799-2_13
21. 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>
22. 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, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_11
23. De Feo, L., Fouotsa, T.B., Kutas, P., Leroux, A., Merz, S.P., Panny, L., Wesolowski, B.: SCALLOP: Scaling the CSI-FiSh. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 345–375. Springer, Cham (May 2023). https://doi.org/10.1007/978-3-031-31368-4_13
24. 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, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_18
25. Esgin, M.F., Steinfeld, R., Tairi, E., Xu, J.: LeOPaRd: Towards practical post-quantum oblivious PRFs via interactive lattice problems. Cryptology ePrint Archive, Paper 2024/1615 (2024), <https://eprint.iacr.org/2024/1615>

26. 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)
27. 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
28. Frixons, P., Schrottenloher, A.: Quantum security of the legendre prf. *Mathematical Cryptology* **1**(2), 52–69 (2021)
29. 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>
30. 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, Berlin, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46497-7_4
31. 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, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_10
32. Heimberger, L., Hennerbichler, T., Meisingseth, F., Ramacher, S., Rechberger, C.: OPRFs from isogenies: Designs and analysis. *Cryptology ePrint Archive*, Report 2023/639 (2023), <https://eprint.iacr.org/2023/639>
33. 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, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_13
34. 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). <https://doi.org/10.1109/EuroSP.2016.30>
35. 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 International Conference on Applied Cryptography and Network Security*. LNCS, vol. 10355, pp. 39–58. Springer, Cham (Jul 2017). https://doi.org/10.1007/978-3-319-61204-1_3
36. 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, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15
37. 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, Cham (May 2021). https://doi.org/10.1007/978-3-030-75248-4_14
38. 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, Berlin, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_34

39. 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)
40. Kaluderović, N., Kleinjung, T., Kostić, D.: Cryptanalysis of the generalised legendre pseudorandom function. *Open Book Series* **4**(1), 267–282 (2020)
41. 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>
42. 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>
43. 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>
44. 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>
45. May, A., Zweydinger, F.: Legendre PRF (multiple) key attacks and the power of preprocessing. In: *CSF 2022 Computer Security Foundations Symposium*. pp. 428–438. IEEE Computer Society Press (Aug 2022). <https://doi.org/10.1109/CSF54842.2022.9919640>
46. 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>
47. 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>
48. Peter Rindal, L.R.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. <https://github.com/osu-crypto/libOTe>
49. 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, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_23
50. 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)
51. Team, F.P.P.: Silk - Privacy Pass Client (accessed Jan 28 2024), <https://addons.mozilla.org/en-US/firefox/addon/privacy-pass/>
52. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) *ACM CCS 2017*. pp. 21–37. ACM Press (Oct / Nov 2017). <https://doi.org/10.1145/3133956.3134053>
53. WhatsApp: Security of End-To-End Encrypted Backups (September 10, 2021), https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf
54. 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 $\mathcal{F}_{2\text{H-OPRF}}^f$

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 [37]. While this is a weaker notion, it still suffices for various applications, such as OPAQUE [36]. 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 1 Proof idea: $\mathcal{F}_{2\text{H-OPRF}}^f$ is $\mathcal{F}_{\text{OPRF}}$ plus a preview interface. All queries to the “ $\mathcal{F}_{\text{OPRF}}$ -part” of $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ (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 $\mathcal{F}_{2\text{H-OPRF}}^f$ is simulated. First, H_1 is simulated with the same code as in $\mathcal{F}_{2\text{H-OPRF}}^f$, 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 INIT message 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 INIT message for the session identifier sid . (Ignore all subsequent INIT messages 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 COMPROMISED as 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 COMPROMISED and $(S^* = S \mid [(S^*, S, x) \in \mathcal{E} \text{ and } P = \mathcal{A}^*])$:
 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.

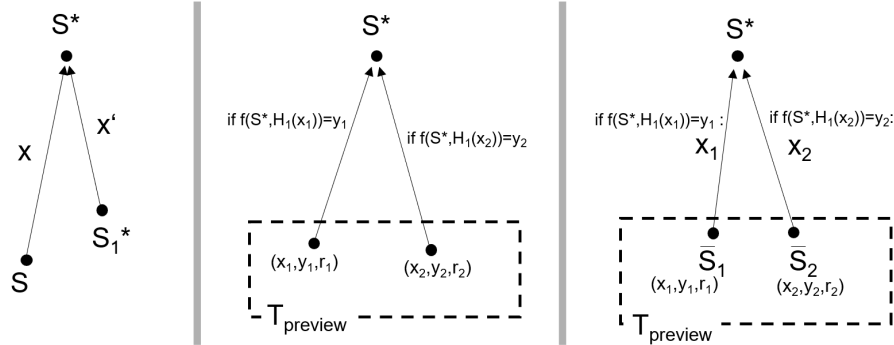


Fig. 8: Illustration of different correlation techniques related to Lemma 1. **Left:** $\mathcal{F}_{\text{corOPRF}}$ allows to correlate each new adversarial key (S^*) with all previous keys (S, S_1^*) on adversarially-chosen inputs (x, x'), including the honest key S . **Middle:** $\mathcal{F}_{2\text{H-OPRF}}^f$ *automatically* correlates each new adversarial key (S^*) by checking consistency with f . It never correlates with the honest key. **Right:** Our simulator of Lemma 1 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 $\mathcal{F}_{2\text{H-OPRF}}^f$. 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 $\mathcal{F}_{2\text{H-OPRF}}^f$, 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ and $\mathcal{F}_{\text{corOPRF}}$ match.

B Relation between $\mathcal{F}_{2\text{H-OPRF}}^f$ and $\mathcal{F}_{\text{OPRF}}$

We formalize the intuition about $\mathcal{F}_{2\text{H-OPRF}}^f$ being almost as secure as $\mathcal{F}_{\text{OPRF}}$: we show that $\mathcal{F}_{2\text{H-OPRF}}^f$ 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.

The simulator maintains functions $H_1, T_{\text{preview}}(\cdot, \cdot)$ initially undefined everywhere, and initially empty sets $\mathcal{K}, T_{\text{programmed}}$. $\text{dom}(T_{\text{preview}})$ the set of tuples these functions are already defined on, i.e., upon initialization we have $\text{dom}(T_{\text{preview}}) = \emptyset$. 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 $\mathcal{F}_{\text{corOPRF}}$:

- On (INIT, sid) from $\mathcal{F}_{\text{corOPRF}}$, forward to \mathcal{A}^* .
- On (EVAL, $sid, ssid, P, S'$) from $\mathcal{F}_{\text{corOPRF}}$, forward to \mathcal{A}^* .
- On (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 (COMPROMISE, sid) from \mathcal{A}^* , forward to $\mathcal{F}_{\text{corOPRF}}$.

On (OFFLINEEVAL, $sid, ssid, K^*, x$) from \mathcal{A}^* :

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

On (RCVCOMPLETE, $sid, ssid, P, K^*$) from \mathcal{A}^*

- Add K^* to \mathcal{K} and run $L \leftarrow \text{getL}(K^*)$
- Send (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 (OFFLINEEVAL, $sid, ssid, \bar{S}, x$) to $\mathcal{F}_{\text{corOPRF}}$ and receive back (OFFLINEEVAL, $sid, ssid, r$)
- Send (H_2, sid, r) to \mathcal{A}^* .

Procedure $\text{getL}(K^*)$:

- Set $L = \emptyset$
- For all $(y^*, x) \in \text{dom}(T_{\text{preview}}) \setminus T_{\text{programmed}}$:
- 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 1.

Lemma 8. *Relative to a random oracle H_1^* and an associated oracle $\mathcal{O}_{\text{key-eq}}$ that decides whether for two pairs $(x, y), (x', y')$, there exists a key K such that $f(H_1^*(x), K) = y$ and $f(H_1^*(x'), K) = y'$, $\mathcal{F}_{2H\text{-OPRF}}^f$ UC-emulates $\mathcal{F}_{\text{OPRF}}$.*

Proof. We give a simulator Sim interacting with $\mathcal{F}_{\text{OPRF}}$ that emulates the adversarial evaluation interfaces of $\mathcal{F}_{2H\text{-OPRF}}^f$. 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 $f(H_1(x^*), K^*) = y^*$, 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 $\mathcal{F}_{2H\text{-OPRF}}^f$: 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 $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, K^*, x)$ from \mathcal{A}^* , Sim does:
 - If $\exists x||y \in \mathcal{I}$ such that $f(H_1(x), K^*) = y$, send $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, x||y, x)$ 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 $(\text{RCVCOMPLETEMALICIOUS}, \text{sid}, \text{ssid}, P, K^*)$ from \mathcal{A}^* , Sim does:
 - If $\exists x||y \in \mathcal{I}$ s.t. $f(H_1(x), K^*) = y$, send $(\text{RCVCOMPLETEMALICIOUS}, \text{sid}, \text{ssid}, P, x||y)$ 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 $(H_2, \text{sid}, x^*, y^*)$ from \mathcal{A}^* , Sim does:
 - If $\exists x||y \in \mathcal{I}$ such that $\mathcal{O}_{\text{key-eq}}(x, y, x^*, y^*) = 1$, send $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, x||y, x^*)$ 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 $f(H_1(x^*), K^*) = y^*$, send $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, K^*, x^*)$ 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 $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, x^*||y^*, x^*)$ 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 $\mathcal{F}_{2H\text{-OPRF}}^f$ 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, $\mathcal{F}_{2H\text{-OPRF}}^f$ replies with a fresh random value to this query. The same happens in the ideal world, where

(OFFLINEEVAL, $sid, ssid, x^*, x^*||y^*$) and (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 CORRELATE function programs a preview $T_{\text{preview}}(x, y)$ into table $F_{\text{malicious}}(\cdot, K^*)$, query (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 $f(H_1(x^*), K^*) = y^*$. This holds because, upon (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 8 implies that $\mathcal{F}_{2\text{H-OPRF}}^f$ is just as strong as $\mathcal{F}_{\text{OPRF}}$, as shown in the following corollary.

Corollary 2. *Let G be a prime-order group and let f be the keyed function*

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

used by 2HashDH. Then $\mathcal{F}_{2\text{H-OPRF}}^f$ 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 $\mathcal{O}_{\text{key-eq}}$: The oracle $H_1^* : \{0, 1\}^* \rightarrow G \setminus \{1_G\}$ 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 $\mathcal{O}_{\text{key-eq}}$ 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 8 that, if f is weakly key-collision resistant, then (non-pathological) protocols that use $\mathcal{F}_{\text{OPRF}}$ can be securely instantiated from $\mathcal{F}_{2\text{H-OPRF}}^f$ instead. Heuristically, the oracles H_1^* and $\mathcal{O}_{\text{key-eq}}$ 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 $\mathcal{O}_{\text{key-eq}}$ oracle would be useless: Either (x, y) or (x', y') is an invalid commitment and then the $\mathcal{O}_{\text{key-eq}}$ -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 $\mathcal{F}_{\text{OPRF}}, H_1^*, \mathcal{O}_{\text{key-eq}}$ -hybrid model. Lemma 8 says that it is then also secure in the $\mathcal{F}_{2\text{H-OPRF}}^f, H_1^*, \mathcal{O}_{\text{key-eq}}$ -hybrid model (and hence also in the $\mathcal{F}_{2\text{H-OPRF}}^f$ -hybrid model).

C Proof of Theorem 1

Proof. Without loss of generality, we assume that the adversary \mathcal{A}^* is the dummy adversary [15], 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 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 $\mathbf{G}_0, \dots, \mathbf{G}_{10}$ 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 $f = (f_{\text{pub}}, f_{\text{sec}})$.

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

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 $\Pi_{\text{OPRF}}^{\mathbf{H}_1, \mathbf{H}_2, f}$ 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 $\Pi_{\text{OPRF}}^{\mathbf{H}_1, \mathbf{H}_2, f}$.

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 : \mathbf{H}_1 from \mathcal{F} . In this game we add the \mathbf{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 sent to $\mathcal{F}_{\text{SFE}}^f$. That means, on a message $(\text{USERINPUT}, \text{ssid}, h)$ to $\mathcal{F}_{\text{SFE}}^f$ the simulator records $\langle \mathcal{F}_{\text{SFE}}^f, \text{ssid}, U, h \rangle$. Then, on a new $\mathbf{H}_1(x, \text{sid})$ query Sim forwards x to $\mathcal{F}_{2\text{H-OPRF}}^f$'s \mathbf{H}_1 interface. When $\mathcal{F}_{2\text{H-OPRF}}^f$ responds with $(\mathbf{H}_1, \text{sid}, h)$ Sim checks if there is a record $\langle \mathcal{F}_{\text{SFE}}^f, \text{ssid}, U, h \rangle$. If there exists such a record then Sim aborts. We call this event E_{guess} . Else Sim stores $\langle \mathbf{H}_1, \text{sid}, x, h \rangle$.

Note that $\mathcal{F}_{2\text{H-OPRF}}^f$ answers the \mathbf{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 $\mathcal{F}_{\text{SFE}}^f$ and afterwards submitted a query $\mathbf{H}_1(x) = h$. But $\mathbf{H}_1(x)$ is chosen by $\mathcal{F}_{2\text{H-OPRF}}^f$ uniformly at random. Let $n_{\mathbf{H}_1}$ be the number of \mathbf{H}_1 queries. Then, E_{guess} happens at most with probability

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

Game \mathbf{G}_3 : Add OFFLINEEVAL and \mathbf{H}_2 interfaces. We now augment the functionality \mathcal{F} with tables $F_{\text{honest}}(\cdot)$ and $F_{\text{malicious}}(\cdot)$, which are initially

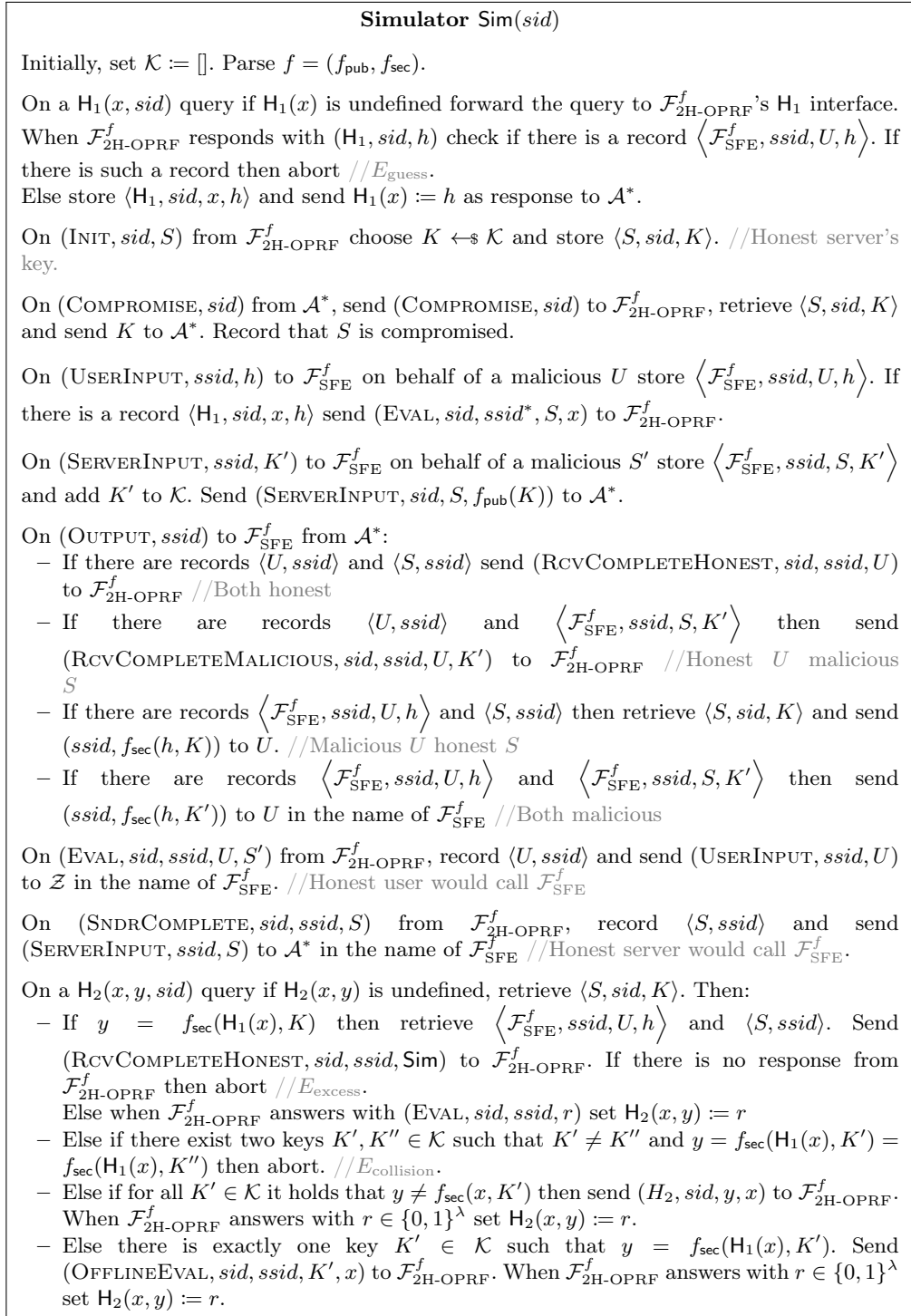


Fig. 10: The simulator that shows that $\Pi_{\text{OPRF}}^{\text{H}_1, \text{H}_2, f}$ UC-realizes $\mathcal{F}_{2\text{H-OPRF}}^f$. We assume that Sim ignores a message when it tries to retrieve a record that does not exist.

uninitialized and on the first reference, are set to a random element in $\{0, 1\}^\lambda$. Furthermore, we add the OFFLINEEVAL interfaces. Like in $\mathcal{F}_{2\text{H-OPRF}}^f$, the OFFLINEEVAL interface for adversarial functions, given a message $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, K^*, x)$ from \mathcal{A}^* , responds with $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, F_{\text{malicious}}(K^*, x))$ and $\text{CORRELATE}(K^*)$ is called. For now, the OFFLINEEVAL interface for the honest function only responds with $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, F_{\text{honest}}(x))$ if the query comes from \mathcal{A}^* . In contrast to the honest function OFFLINEEVAL interface of $\mathcal{F}_{2\text{H-OPRF}}^f$ requests from $P = S$ are forwarded to Sim and answered by Sim with $H_2(x, f_{\text{sec}}(H_1(x), K))$. Finally, we add the H_2 interface to \mathcal{F} exactly as it is in $\mathcal{F}_{2\text{H-OPRF}}^f$.

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 $\mathcal{F}_{\text{SFE}}^f$ then Sim stores this key in a list \mathcal{K} . Next, Sim checks on every $H_2(x, y)$ query if there exist two keys $K', K'' \in \mathcal{K}$ such that $K' \neq K''$ and $y = f_{\text{sec}}(H_1(x), K') = f_{\text{sec}}(H_1(x), K'')$. 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 $h_1, \dots, h_{n_{H_1}}$ as challenge and answers the i -th H_1 query x_i with $H_1(x_i) = h_i$ 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 $y = f_{\text{sec}}(H_1(x^*), K) = f_{\text{sec}}(H_1(x^*), K')$ then \mathcal{B} retrieves the record $(H_1(x^*), i^*)$ and outputs (K, K', i^*) , which is a key collision.

$$\Pr[\mathbf{G}_4] = \Pr[\mathbf{G}_3] \leq \text{Adv}_{\mathcal{F}, \mathcal{B}}^{\text{WKCR}}(n_{H_1}).$$

Game \mathbf{G}_5 : Programming H_2 for malicious keys. Next, we modify how a hash function request $H_2(x, y)$ is answered for some $y = f_{\text{sec}}(H_1(x), K')$ 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 $y \neq f_{\text{sec}}(H_1(x), K')$ then Sim sends (H_2, sid, y, x) to \mathcal{F} . When \mathcal{F} answers with $r \in \{0, 1\}^\lambda$ the simulator sets $H_2(x, y) := r$.
2. Else, send $(\text{OFFLINEEVAL}, \text{sid}, \text{ssid}, K', x)$ to \mathcal{F} . When \mathcal{F} answers with $r \in \{0, 1\}^\lambda$ the simulator sets $H_2(x, y) := r$.

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 $y = f_{\text{sec}}(x, K^*)$ 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 $\mathcal{F}_{2\text{H-OPRF}}^f$ to set $H_2(x, y)$ 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 $y = f_{\text{sec}}(H_1(x), K^*)$ 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 $H_2(x', y')$ with $y' = f_{\text{sec}}(H_1(x'), K')$ 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 $H_2(x, f_{\text{sec}}(H_1(x), K'))$. 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 $y = f_{\text{sec}}(H_1(x), K)$ 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 $\mathcal{F}_{2\text{H-OPRF}}^f$, 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 $\mathcal{F}_{\text{SFE}}^f$ the simulator now checks if there is a record $\langle H_1, ssid, x, h \rangle$. If that is the case then Sim sends (EVAL, $ssid, ssid^*, S, x$) to \mathcal{F} .
- To answer a hash query $H_2(x, y)$ where $y = f_{\text{sec}}(H_1(x), K)$ the simulator checks if there is a corresponding record $\langle \mathcal{F}_{\text{SFE}}^f, ssid, U, h \rangle$ with $h = H_1(x)$ 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, $ssid, ssid, \text{Sim}$) message to \mathcal{F} . When Sim receives a response (EVAL, $ssid, ssid, r$) it sets $H_2(x, y) := r$. As a

result, $H_2(x, y)$ 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 $(\text{RCVCOMPLETEHONEST}, \text{sid}, \text{ssid}, \text{Sim})$ 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 $(\text{EVAL}, \text{SNDRCOMPLETE})$ 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 $(h_1, \dots, h_n) \in \mathcal{I}^n$ and $f_{\text{pub}}(K)$ 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 $H_1(x_i) = h_i$ and that Sim uses its $f_{\text{sec}}(\cdot, K)$ -oracle instead of choosing a key for the honest server. That means, whenever Sim receives a message $(\text{SNDRCOMPLETE}, \text{sid}, \text{ssid}, S)$ it stores $\langle S, \text{ssid} \rangle$ and sends $(\text{SERVERINPUT}, \text{sid}, S, f_{\text{pub}}(K))$ to \mathcal{Z} . When Sim receives an $\mathcal{F}_{\text{SFE}}^f$ -messages $(\text{USERINPUT}, \text{ssid}, h)$, and $(\text{OUTPUT}, \text{ssid})$ for a ssid where Sim has a corresponding record $\langle S, \text{ssid} \rangle$ then Sim forwards $h \in \mathcal{I}$ as a query to its $f_{\text{sec}}(\cdot, K)$ oracle. When the oracle returns $f_{\text{sec}}(h, K)$ then Sim gives $(\text{ssid}, f_{\text{sec}}(h, K))$ 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 $(\text{USERINPUT}, \text{ssid}, h)$ message to $\mathcal{F}_{\text{SFE}}^f$. Also, the honest user computes its output still as $H_2(x, f_{\text{sec}}(H_1(x), K))$). Now, whenever \mathcal{B}' receives a query $H_2(x, y)$ it uses its $\mathcal{V}_K^f(i, y)$ oracle to check if $y = f_{\text{sec}}(h_i = H_1(x), K)$ holds.

Note that Sim queries its $f_{\text{sec}}(\cdot, K)$ oracle if two conditions are satisfied. (1) Sim received messages $(\text{USERINPUT}, \text{ssid}, h)$, and $(\text{OUTPUT}, \text{ssid})$. In this case Sim sends an EVAL message to \mathcal{F} . (2) Sim stored $\langle S, \text{ssid} \rangle$. This means that Sim received a message $(\text{SNDRCOMPLETE}, \text{sid}, \text{ssid}, S)$ from \mathcal{F} . Therefore, we can upper-bound the number of $f_{\text{sec}}(\cdot, K)$ oracle queries by q . E_{excess} implies that there was at least one query $H_2(x, y)$ with $y = f_{\text{sec}}(H_1(x), K)$ but without a matching record $\langle \mathcal{F}_{\text{SFE}}^f, \text{ssid}, U, h \rangle$ or a matching record $\langle S, \text{ssid} \rangle$. Consequently, Sim did either not send an EVAL message for this ssid or there was no 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 $f_{\text{sec}}(\cdot, K)$ oracle and $q + 1$ distinct queries to its $\mathcal{V}_K^f(\cdot, \cdot)$ oracle, such that \mathcal{V}_K^f outputs 1.

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

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

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

In \mathbf{G}_6 we ensured that $\mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K)) = F_{\text{honest}}(x)$, 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 $\text{OFFLINEEVAL}(x) = \mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K))$.

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 (EVAL, $sid, ssid, U, S, x$) from \mathcal{F} , instead of running the user's code on x according to $\Pi_{\text{OPRF}}^{\mathbf{H}_1, \mathbf{H}_2, f}$, Sim now disregards x and sends the message (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 (SERVERINPUT, $ssid, K'$) from \mathcal{A}^* in the name of some malicious server and a $\mathcal{F}_{\text{SFE}}^f$ -message (OUTPUT, $ssid$) from \mathcal{A}^* then Sim sends (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 $\mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K')) = F_{\text{malicious}}(K', x)$. But in \mathbf{G}_5 we programmed $\mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K'))$ exactly to this value.

- If Sim receives a message (SNDRCOMPLETE, $sid, ssid, S$) from \mathcal{F} then Sim sends the message (SERVERINPUT, $ssid, S, f_{\text{pub}}(K)$) to \mathcal{A}^* in the name of $\mathcal{F}_{\text{SFE}}^f$. Note that Sim can compute $f_{\text{pub}}(K)$ as it only depends on K . On a $\mathcal{F}_{\text{SFE}}^f$ -message (OUTPUT, $ssid$) from \mathcal{A}^* the simulator sends (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 $\mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K)) = F_{\text{honest}}(x)$. But in \mathbf{G}_6 $\mathbf{H}_2(x, f_{\text{sec}}(\mathbf{H}_1(x), K))$ was programmed exactly to this value. Also note that the number of (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 $H_2(x, f_{\text{sec}}(H_1(x), K))$ 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 $\mathcal{F} = \mathcal{F}_{2\text{H-OPRF}}^f$ and the described simulator works as described in Fig. 10.

□

D 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 $\mathcal{F}_{\text{SFE}}^{f_{\text{Seq}}}$ 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}_{ZK} and $\mathcal{F}_{\text{VOLE}+}$. $\mathcal{F}_{\text{SFE}}^{f_{\text{Seq}}}$ also relays protocol outputs of the simulator to the respective parties. That is, the functionality resembles already $\mathcal{F}_{\text{SFE}}^{f_{\text{Seq}}}$, 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 G₂: Abort upon collision in offset vectors. We let the simulator abort whenever two entries of \mathbf{I} or \mathbf{I}' are the same.

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

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

Game G₃: Simulate \mathcal{F}_{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}_{ZK} anymore but instead just sends COMMITTED to the user. The simulated \mathcal{F}_{ZK} hence doesn't check if the witness \mathbf{w} satisfies the polynomial constraints \mathcal{F} . Instead, Sim lets \mathcal{F}_{ZK} output $(\text{OUTPUT}, \text{ssid}, \top)$ 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 G₄: Extract from a corrupt user. We change the code of the simulator in case of a corrupt user. Sim takes h , input by the corrupt user into $\mathcal{F}_{\text{Vole}+}$ and sends it to $\mathcal{F}_{\text{SFE}}^{f_{\text{LSeq}}}$, which replies with $(\mathbf{e}, \mathbf{e}') = f_{\text{LSeq}}(h, K)$. 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 G₅: Simulate $\mathcal{F}_{\text{Vole}+}$ in case of a corrupt user. In case of a corrupt user, instead of simulating $\mathcal{F}_{\text{Vole}+}$, Sim picks $\mathbf{o} \in \mathbb{F}_p^{\ell_{\text{eval}}}$ uniformly at random subject to $\binom{\mathbf{o}_i}{p} = e'_i$ for all $i \in [\ell_{\text{eval}}]$, and picks $c_{\mathbf{u}}, c_{\mathbf{v}} \in \mathbb{F}_p$ uniformly at random. Sim sends $(\text{OUTPUT}, \mathbf{o}, \gamma, c_{\mathbf{u}}, c_{\mathbf{v}})$ to \mathcal{A}^* . This does not change the distribution of $\mathbf{o}, \gamma, c_{\mathbf{u}}, c_{\mathbf{v}}$ so the view of \mathcal{Z} does not change and we have

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

Game G₆: 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 G₇: Abort upon $c_{\mathbf{u}}, c_{\mathbf{v}}$ collision. We let the simulator abort if a corrupt server submits $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b)$ to \mathcal{F}_{ZK} and $\mathbf{u}', \mathbf{v}', r'_{\mathbf{u}}, r'_{\mathbf{v}}$ to $\mathcal{F}_{\text{VOLE}^+}$ such that $(r_{\mathbf{u}}, r_{\mathbf{v}}) \neq (r'_{\mathbf{u}}, r'_{\mathbf{v}})$ and the following two equations hold:

$$\begin{aligned}\langle \gamma, \mathbf{u}' \rangle + r'_{\mathbf{u}} &= \langle \gamma, (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{l}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{u}} \\ \langle \gamma, \mathbf{v}' \rangle + r'_{\mathbf{v}} &= \langle \gamma, \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}\end{aligned}$$

Since γ 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 G₈: Functionality produces user output for honest sessions.

We change the simulator to send $(\text{OUTPUT}, ssid)$ to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ upon \mathcal{A}^* sending $(\text{OUTPUT}, ssid)$ to \mathcal{F}_{ZK} for an honest session, i.e., both U and S are honest. Note that this results in $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ sending output $f_{\text{LSeq}}(h, K)$ 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 G₉: No output for mismatching \mathcal{F} . We terminate the simulation in case of a corrupt server sending \mathcal{F} to \mathcal{F}_{ZK} such that not all of the following hold, where $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \gamma$ denote the input and output of the server to $\mathcal{F}_{\text{VOLE}^+}$:

$$\mathcal{F} = \{f_u, f_v, f_e^{(1)}, \dots, f_e^{(\ell_{\text{com}})}, f_a\} \text{ where} \quad (10)$$

$$f_u(X, \mathbf{Y}, \mathbf{Z}, r, s, t) := c_{\mathbf{u}} - r - \langle \gamma, (X \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{l}') * \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle, \quad (11)$$

$$f_v(X, \mathbf{Y}, \mathbf{Z}, r, s, t) := c_{\mathbf{v}} - s - \langle \gamma, \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle, \quad (12)$$

$$f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s, t) := (X + l_i)Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\text{com}}], \quad (13)$$

$$f_a(X, \mathbf{Y}, \mathbf{Z}, r, s, t) := Y_{\ell_{\text{eval}}}t - 1 \quad (14)$$

$$c_{\mathbf{u}} = \langle \gamma, \mathbf{u} \rangle + r'_{\mathbf{u}}, c_{\mathbf{v}} = \langle \gamma, \mathbf{v} \rangle + r'_{\mathbf{v}}. \quad (15)$$

In \mathbf{G}_8 , the honest user computed $c_{\mathbf{u}}, c_{\mathbf{v}}$ as in equation (13) and received γ as output. Because \mathcal{F}_{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 (10)-(15). 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}_{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

$$(r_{\mathbf{u}}, r_{\mathbf{v}}) = (r'_{\mathbf{u}}, r'_{\mathbf{v}}). \quad (16)$$

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

$$\begin{aligned} \langle \gamma, \mathbf{u}' \rangle + r'_{\mathbf{u}} &\neq \langle \gamma, (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{I}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{u}}, \text{ or} \\ \langle \gamma, \mathbf{v}' \rangle + r'_{\mathbf{v}} &\neq \langle \gamma, \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}. \end{aligned}$$

Consequently, the user in \mathbf{G}_9 will abort because either the first (f_u) or second (f_v) polynomial relation of \mathcal{F}_{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 $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \mathbf{u}, \mathbf{v}, \gamma, \mathbf{e})$ (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:

$$(K + l_i)s_i^2 = E(e_i) \text{ for all } i \in [\ell_{\text{com}}], \quad (17)$$

$$\mathbf{v} = \mathbf{a}^2 * \text{shift}(\mathbf{a}^2), \quad (18)$$

$$\mathbf{u} = (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{I}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2). \quad (19)$$

$$a_{\ell_{\text{eval}}} b = 1. \quad (20)$$

In \mathbf{G}_{10} , if any of the above equations does not verify, \mathcal{F}_{ZK} does not output **status** and hence if the simulation terminates 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}_{ZK} does not output **status**, which happens either because a corrupt server sends polynomials that do not match his $\mathcal{F}_{\text{VOLE}^+}$ 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}_{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}_{ZK} with a 0 component, i.e., we only continue if

$$a_i \neq 0 \text{ for all } i \in [\ell_{\text{eval}}]. \quad (21)$$

As of \mathbf{G}_{11} , we know that $\mathbf{o} = \mathbf{u} + h\mathbf{v} = (K \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{1}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) + h \cdot \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) = ((K+h) \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{1}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2)$. Suppose \mathcal{F}_{ZK} outputs **status**. Since the l'_i are pairwise different, we have that at most one of the $(K+l'_i)$ is zero, so if two entries of \mathbf{o} are zero, then at least one entry of \mathbf{a} is zero. Conversely, if any of the entries of \mathbf{a} is zero, then it must be a_i with $i < \ell_{\text{eval}}$, since $a_{\ell_{\text{eval}}}$ has an inverse. Therefore o_i and o_{i+1} are both zero. We have shown that \mathbf{a} has one or more zero entries if and only if \mathbf{o} has two or more zero entries. Consequently, checking for zeros in \mathbf{a} is equivalent to checking for two or more zeros in \mathbf{o} , and the simulator terminates in this game if and only if the user aborts in \mathbf{G}_{11} , and we have

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

Game \mathbf{G}_{13} : No output if public key \mathbf{e} is wrong. We skip the recomputation 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}_{ZK} in case a corrupt server has sent and seen values $(K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \mathbf{u}, \mathbf{v}, \gamma, \mathbf{e})$ where \mathbf{e} is not honestly generated from K . I.e., the simulator only continues if

$$\mathbf{e} = \left\{ \left(\frac{K+l_i}{p} \right) \right\}_{i \in [\ell_{\text{com}}]}. \quad (22)$$

We show that the check in \mathbf{G}_{12} passes if and only if equation 22 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 $i \in [\ell_{\text{com}}]$. If $s_i \neq 0$ for all $i \in [\ell_{\text{com}}]$ we hence have $\left(\frac{K+l_i}{p}\right) = e_i$ for all $i \in [\ell_{\text{com}}]$. Hence, the above equation can only be violated by a corrupt server choosing $K \neq -l_i, s_i = 0$ for some $i \in [\ell_{\text{com}}]$, 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 $f_{\text{LSeq}}(h, -l_i) \neq (\mathbf{e}, \mathbf{e}')$, and aborts.

We now show that if equation (22) holds, the check in \mathbf{G}_{12} passes. Because the l_i are all different, we have $e_i = 0$ for at most one $i \in [\ell_{\text{com}}]$ and, if so, $K = -l_i$. As of \mathbf{G}_{11} we have $\mathbf{o} = \mathbf{u} + h\mathbf{v} = ((K+h) \cdot \mathbf{1}_{\ell_{\text{eval}}} + \mathbf{1}') * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2)$ and hence $f_{\text{LSeq}}(h, -l_i) = (\{(\frac{-l_i+l_j}{p})\}_{j \in [\ell_{\text{com}}]}, \{(\frac{e_i}{p})\}_{j \in [\ell_{\text{eval}}]})$, 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 $(\text{OUTPUT}, \text{sid})$ to the functionality when simulating an honest user facing a corrupt server, where the user has received **status** from \mathcal{F}_{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 if 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 $\mathcal{F}_{\text{VOLE}^+}$ which, because of equations (18),(19),(16) is equal to $(K+h+l'_i)(\mathbf{a}^2 * \text{shift}(\mathbf{a}^2))_i$ for all $i \in [\ell_{\text{eval}}]$. Because (21) $a_i \neq 0$ for all $i \in [\ell_{\text{eval}}]$ we have $\left(\frac{a_i}{p}\right) = \left(\frac{K+h+l'_i}{p}\right)$ for all $i \in [\ell_{\text{eval}}]$, i.e., if the user generates output in this game and the previous one, it is the same.

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 $\mathcal{F}_{\text{SFE}}^{\text{flSeq}}$, 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.

E VOLE^+

We state the ideal functionality $\mathcal{F}_{\text{sVOLE}}$ that our protocol for VOLE^+ relies on in Figure 12. The functionality is a special case of the functionality from Figure 2 of [8] (which is adapted from [49]), restricted to $p = q$, $\mathcal{L} = \{2^{S_\Delta}\}$ (no leakage), and where $\mathcal{C} = \mathbb{F}_p$. Our protocol Π_{VOLE^+} for realizing the VOLE^+ functionality is described in Fig. 13.

Theorem 4. *The protocol $\Pi_{\text{VOLE}^+}^{p,\ell,k}$ of Fig. 13 UC-realizes $\mathcal{F}_{\text{VOLE}^+}$ in the $\mathcal{F}_{\text{sVOLE}}$ -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

On (USERINPUT, sid, U) from $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$, send (VFINPUT, sid, U).

On (SERVERINPUT, sid, S, \bar{e}) from $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$, store (sid, \bar{e}) and send (PRVINPUT, sid, S) to \mathcal{A}^* .

On \mathcal{A}^* sending \mathbf{w} to \mathcal{F}_{ZK} on behalf of a corrupt server:

- Parse $\mathbf{w} := (K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b)$, send (SERVERINPUT, sid, K) to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$, retrieve (SERVERINPUT, sid, S). Store ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b$).
- Send (PRVINPUT, sid, S) to \mathcal{A}^* .

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

On (COMMITTED, sid) from \mathcal{A}^* to \mathcal{F}_{ZK} :

- If U is honest, choose $\gamma \xleftarrow{\$} \mathbb{F}_p^n$. If S is corrupt, append γ to the record (K, \dots) stored previously, otherwise append it to record (sid, \mathbf{e}).
- If U is honest, send (USERINPUT, sid, γ, U) to \mathcal{A}^* .
- If S is honest, send (SERVERINPUT, sid, S) to \mathcal{A}^* .

On \mathcal{A}^* sending $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$ to $\mathcal{F}_{\text{VOLE}^+}$ on behalf of a corrupt server:

- Retrieve ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \gamma$) and append $\mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$ to the record.
- Abort if $(r_{\mathbf{u}}, r_{\mathbf{v}}) \neq (r'_{\mathbf{u}}, r'_{\mathbf{v}})$, $\langle \gamma, \mathbf{u} \rangle + r'_{\mathbf{u}} = \langle \gamma, (K \cdot \mathbf{1}_{\text{eval}} + \mathbf{I}) * \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{u}} \gamma$ and $\langle \gamma, \mathbf{v} \rangle + r'_{\mathbf{v}} = \langle \gamma, \mathbf{a}^2 * \text{shift}(\mathbf{a}^2) \rangle + r_{\mathbf{v}}$.
- Send (SERVERINPUT, sid, S) to \mathcal{A}^* .

On \mathcal{A}^* sending (USERINPUT, sid, h, γ) to $\mathcal{F}_{\text{VOLE}^+}$ on behalf of a corrupt user:

- Send (USERINPUT, sid, h) to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ and receive back (USERINPUT, sid, U).
- Send (OUTPUT, sid) to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ and receive back (sid, y). Parse $y := (\mathbf{e}, \mathbf{e}') \in \{-1, 0, 1\}^{\ell_{\text{com}}} \times \{-1, 0, 1\}^{\ell_{\text{eval}}}$. Store ($h, \gamma, \mathbf{e}, \mathbf{e}'$).

On (OUTPUT, sid, P) from \mathcal{A}^* to $\mathcal{F}_{\text{VOLE}^+}$:

- 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 $\binom{2\mathbf{o}_i}{p} = \mathbf{e}'_i$, e.g., set $\mathbf{o}_i \leftarrow E(e_i)r_i^2$ 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 ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \gamma, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$) and set $c_{\mathbf{u}} := \langle \gamma, \mathbf{u} \rangle + r'_{\mathbf{u}}$, $c_{\mathbf{v}} := \langle \gamma, \mathbf{v} \rangle + r'_{\mathbf{v}}$.
- Else (both parties are honest) retrieve ($ssid, \bar{\mathbf{e}}, \gamma$) and sample $c_{\mathbf{u}} \xleftarrow{\$} \mathbb{F}_p, c_{\mathbf{v}} \xleftarrow{\$} \mathbb{F}_p$.
- If $P = \mathcal{A}^*$, send $(\gamma, c_{\mathbf{u}}, c_{\mathbf{v}})$ to \mathcal{A}^* .
- If $P = U$ and U is corrupt, send (OUTPUT, $sid, \mathbf{o}, \gamma, c_{\mathbf{u}}, c_{\mathbf{v}}$) to \mathcal{A}^* .
- If $P = S$, do:
 - If S is corrupt, send (OUTPUT, sid, γ) 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 ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \gamma, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}$) 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}_{ZK} .

On \mathcal{A}^* sending \mathcal{F} to \mathcal{F}_{ZK} on behalf of a corrupt server:

- Retrieve ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \gamma, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \mathbf{e}$) and set $c_{\mathbf{u}} \leftarrow \langle \gamma, \mathbf{u} \rangle + r_{\mathbf{u}}$, $c_{\mathbf{v}} \leftarrow \langle \gamma, \mathbf{v} \rangle + r_{\mathbf{v}}$
- Set $f_{\mathbf{u}}(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \gamma, (X \cdot \mathbf{1}_{\text{eval}} + \mathbf{I}) * \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle$
- Set $f_{\mathbf{v}}(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \gamma, \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle$
- Set $f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i)Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\text{com}}]$
- Set $\mathcal{F}_U \leftarrow \{f_{\mathbf{u}}, f_{\mathbf{v}}, f_e^{(1)}, \dots, f_e^{(\ell_{\text{com}})}\}$. If $\mathcal{F}_U = \mathcal{F}$, append bit = 1 to the stored data, otherwise append bit = 0.
- Send (PROVE, sid, \mathcal{F}_U) to \mathcal{A}^* .

On \mathcal{A}^* sending \mathcal{F} to \mathcal{F}_{ZK} on behalf of a corrupt user:

- Retrieve record ($h, \gamma, \mathbf{e}, \mathbf{e}', c_{\mathbf{u}}, c_{\mathbf{v}}, \mathbf{o}$)
- Set $f_{\mathbf{u}}(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{u}} - r - \langle \gamma, (X \cdot \mathbf{1}_{\text{eval}} + \mathbf{I}) * \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle$
- Set $f_{\mathbf{v}}(X, \mathbf{Y}, \mathbf{Z}, r, s) := c_{\mathbf{v}} - s - \langle \gamma, \mathbf{Y}^2 * \text{shift}(\mathbf{Y}^2) \rangle$
- Set $f_e^{(i)}(X, \mathbf{Y}, \mathbf{Z}, r, s) := (X + l_i)Z_i^2 - E(e_i) \quad \forall i \in [\ell_{\text{com}}]$
- Set $\mathcal{F}_S \leftarrow \{f_{\mathbf{u}}, f_{\mathbf{v}}, f_e^{(1)}, \dots, f_e^{(\ell_{\text{com}})}\}$. If $\mathcal{F}_S = \mathcal{F}$, append bit = 1 to the stored data, otherwise append bit = 0.
- Send (PROVE, sid, \mathcal{F}_S) to \mathcal{A}^* .

On (OUTPUT, sid) from \mathcal{A}^* to \mathcal{F}_{ZK} :

- If U corrupt, retrieve record ($h, \gamma, \mathbf{e}, \mathbf{e}', c_{\mathbf{u}}, c_{\mathbf{v}}, \mathbf{o}, \text{bit}$) and if bit = 1, send ($ssid, \text{T}$) to \mathcal{A}^*
- If both parties are honest, send (OUTPUT, sid) to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$
- If S is corrupt, retrieve ($K, \mathbf{a}, \mathbf{s}, r_{\mathbf{u}}, r_{\mathbf{v}}, b, \gamma, \mathbf{u}, \mathbf{v}, r'_{\mathbf{u}}, r'_{\mathbf{v}}, \mathbf{e}, \text{bit}$) and terminate if bit = 0. Otherwise, Sim sends (OUTPUT, $ssid$) to $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ only if

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

Fig. 11: Simulator for Theorem 3 interacting with $\mathcal{F}_{\text{SFE}}^{\text{fSeq}}$ for simulating protocol $\Pi_{\text{SFE}}^{\text{fSeq}}$ from Fig. 6, including hybrid functionalities $\mathcal{F}_{\text{ZK}}, \mathcal{F}_{\text{VOLE}^+}$.

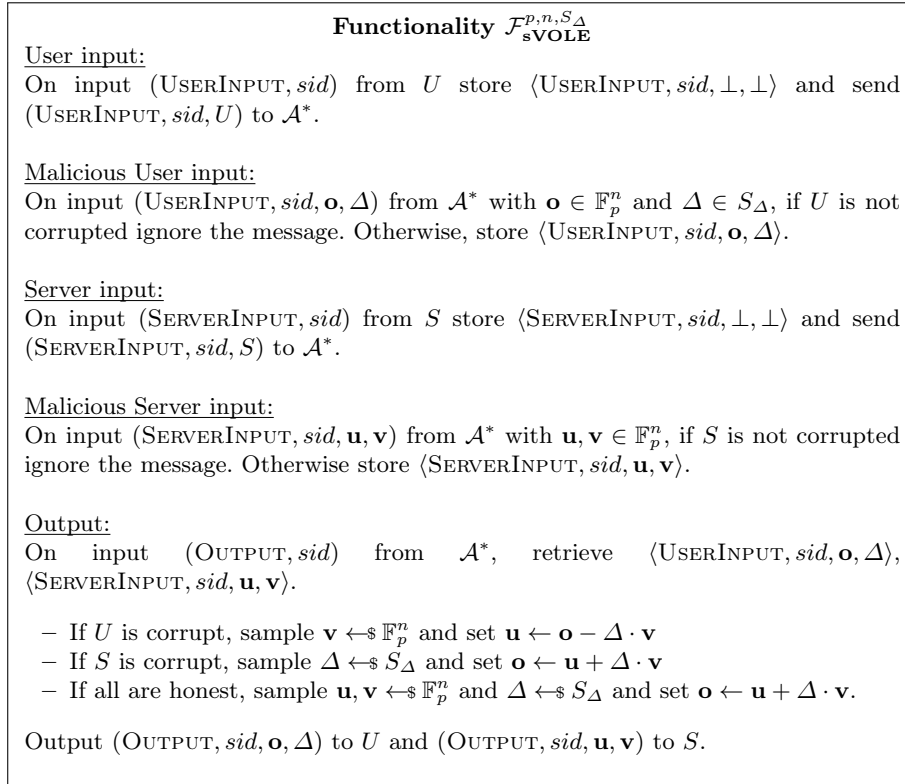


Fig. 12: Ideal functionality for endemic subset VOLE $\mathcal{F}_{s\text{VOLE}}^{p,n,S_\Delta}$, for a prime p , vector length n and a non-empty subset $S_\Delta \subset \mathbb{F}_p$.

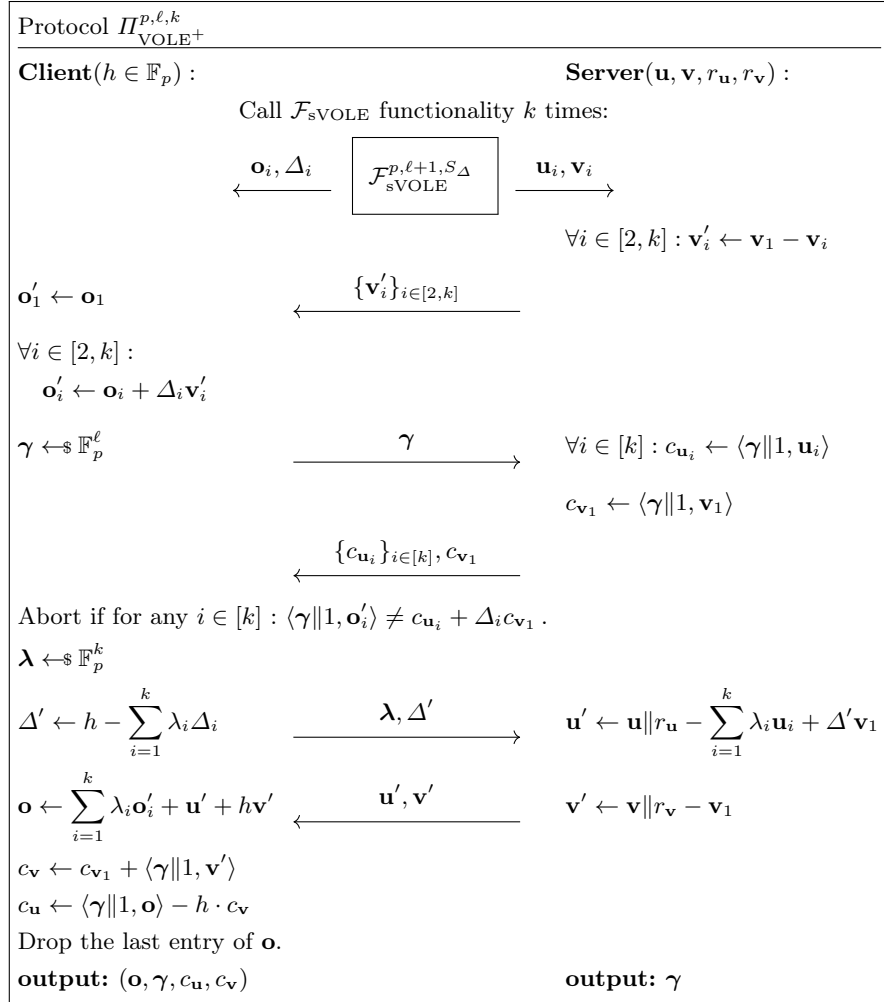


Fig. 13: Protocol $\Pi_{\text{VOLE}^+}^{p,\ell,k}$ for VOLE^+ for vectors over \mathbb{F}_p of length ℓ and with k executions of subset VOLE .

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 $\mathcal{F}_{\text{sVOLE}}$ protocol. The malicious user is allowed to pick his outputs \mathbf{o}_i, Δ_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 $\mathcal{F}_{\text{VOLE}+}$ functionality. (Here we used that a corrupted user is allowed to dictate the $\boldsymbol{\gamma}$ vector used by $\mathcal{F}_{\text{VOLE}+}$.) 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 $\mathcal{F}_{\text{VOLE}+}$, 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 $\mathbf{u}' := \mathbf{o} \| \mathbf{o}_{\ell+1} - h\mathbf{v}' - \sum_{i=1}^k \lambda_i (\mathbf{u}_i + \Delta_i \mathbf{v}_1)$.

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 $\langle \boldsymbol{\gamma} \| 1, \mathbf{u}_i + \Delta_i (\mathbf{v}_i + \mathbf{v}'_i) \rangle = c_{\mathbf{u}_i} + \Delta_i c_{\mathbf{v}_1}$ for every $i \in [k]$. If the server behaved honestly, all these equations would be satisfied for all values of $\Delta_i \in S_{\Delta}$, 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_{Δ} . 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 $\Delta_i^* \in S_{\Delta}$. Then the simulator aborts with probability $1 - |S_{\Delta}|^{-|G|}$ because the honest user would only continue if it happened to be the case that $\Delta_i = \Delta_i^*$ for all $i \in G$.

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

with a statistical distance of at most $2^{-s'}$ with

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

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

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 $\langle \gamma \| 1, \mathbf{v}_i + \mathbf{v}'_i \rangle \neq \langle \gamma \| 1, \mathbf{v}_j + \mathbf{v}'_j \rangle$, 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 $\langle \gamma \| 1, \mathbf{v}^* \rangle = c_{\mathbf{v}_1}$. The honest user would compute

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

which can be rewritten as

$$\begin{aligned} \mathbf{o} &= \sum_{i \in [k]} \lambda_i \mathbf{u}_i + \sum_{i \in G} \lambda_i \Delta_i^* (\mathbf{v}_i + \mathbf{v}'_i - \mathbf{v}^*) + \sum_{i \in [k]} \lambda_i \Delta_i \mathbf{v}^* + \mathbf{u}' + h \mathbf{v}' \\ &= \underbrace{\mathbf{u}' - \Delta' \mathbf{v}^* + \sum_{i \in [k]} \lambda_i \mathbf{u}_i + \sum_{i \in G} \lambda_i \Delta_i^* (\mathbf{v}_i + \mathbf{v}'_i - \mathbf{v}^*)}_{:= \mathbf{u}^+} + h \underbrace{(\mathbf{v}^* + \mathbf{v}')}_{:= \mathbf{v}^+}, \end{aligned} \quad (23)$$

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 (23) 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 $\mathcal{F}_{\text{VOLE}^+}$. Then it is clear from (23) that the \mathbf{o} output of the honest user agrees with the \mathbf{o} output of $\mathcal{F}_{\text{VOLE}^+}$. Moreover, the honest user outputs $c_{\mathbf{v}} := c_{\mathbf{v}_1} + \langle \gamma \| 1, \mathbf{v}' \rangle$, which is indeed equal to $\langle \gamma, \mathbf{v} \rangle + r_{\mathbf{v}}$ because $\langle \gamma \| 1, \mathbf{v}^* \rangle = c_{\mathbf{v}_1}$, so it matches the output in the ideal world. Finally, $c_{\mathbf{u}} = \langle \gamma, \mathbf{o} \rangle - r_{\mathbf{u}} - h \cdot \langle \gamma, \mathbf{v} \rangle$ 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 (USERINPUT, $ssid, U$) or (SERVERINPUT, $ssid, S$) from $\mathcal{F}_{\text{VOLE}^+}$, forward the message to \mathcal{A} .

On (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” ($ssid, ssid_1, \dots, ssid_k, \mathbf{o}_1, \dots, \mathbf{o}_k, \Delta_1, \dots, \Delta_k$)

On (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” ($ssid, ssid_1, \dots, ssid_k, \mathbf{u}_1, \dots, \mathbf{u}_k, \mathbf{v}_1, \dots, \mathbf{v}_k$).

On (OUTPUT, $ssid$) from \mathcal{A} to $\mathcal{F}_{\text{sVOLE}}^{p,n,S_\Delta}$:

- If the user is corrupt, retrieve ($ssid, ssid_1, \dots, ssid_k, \mathbf{o}_1, \dots, \mathbf{o}_k, \Delta_1, \dots, \Delta_k$) and send \mathbf{o}_i, Δ_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 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 $\gamma \xleftarrow{\$} \mathbb{F}_p^l$, append $\gamma, \{\mathbf{v}'_i\}_{i \in [2, k]}$ to the server record, and send γ to \mathcal{A} .

On γ 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 $\gamma, \{c_{\mathbf{u}_i}\}_{i \in [k]}, c_{\mathbf{v}_1}$ 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 ($ssid, \{ssid_i\}_{i \in [k]}, \{\mathbf{u}_i\}_{i \in [k]}, \{\mathbf{v}_i\}_{i \in [k]}$)
- For every $i \in [k]$ let s_i denote the number of solutions $\Delta_i \in S_\Delta$ of equation $\langle \gamma | 1, \mathbf{u}_i + \Delta_i (\mathbf{v}_i + \mathbf{v}'_i) \rangle = c_{\mathbf{u}_i} + \Delta_i c_{\mathbf{v}_1}$
- 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 $\lambda \xleftarrow{\$} \mathbb{F}_p^k, \Delta' \xleftarrow{\$} \mathbb{F}_p$, append these values, $G, \{\Delta_i^*\}_{i \in G}$ and the received values to the server record and send λ, Δ' to \mathcal{A} .

On λ, Δ' from \mathcal{A} as message from a corrupt user:

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

On \mathbf{u}', \mathbf{v}' from \mathcal{A} as message from a corrupt server:

- Retrieve the server record ($ssid, \{ssid_i\}_{i \in [k]}, \{\mathbf{u}_i\}_{i \in [k]}, \{\mathbf{v}_i\}_{i \in [k]}, \{c_{\mathbf{u}_i}\}_{i \in [k]}, c_{\mathbf{v}_1}, \gamma, \{\mathbf{v}'_i\}_{i \in [2, k]}, \lambda, \Delta', G, \{\Delta_i^*\}_{i \in G}$)
- Choose an arbitrary $j \in [k] \setminus G$.
- Set $\mathbf{u} || r_{\mathbf{u}} := \mathbf{u}' - \Delta' (\mathbf{v}_j + \mathbf{v}'_j) + \sum_{i \in [k]} \lambda_i \mathbf{u}_i + \sum_{i \in G} \lambda_i \Delta_i^* (\mathbf{v}_i + \mathbf{v}'_i - (\mathbf{v}_j + \mathbf{v}'_j))$.
- Set $\mathbf{v} || r_{\mathbf{v}} := \mathbf{v}_j + \mathbf{v}'_j + \mathbf{v}'$.
- Send (SERVERINPUT, $ssid, \mathbf{u}, \mathbf{v}, r_{\mathbf{u}}, r_{\mathbf{v}}$) to $\mathcal{F}_{\text{VOLE}^+}$.
- Send (OUTPUT, $ssid, U$) to $\mathcal{F}_{\text{VOLE}^+}$.

Fig. 14: Simulator of Theorem 2.