Pool: A Practical OT-based OPRF from Learning with Rounding

Alex Davidson
LASIGE, Faculdade de Ciências,
Universidade de Lisboa
Lisbon, Portugal
aadavidson@fc.ul.pt

Amit Deo
Zama
Paris, France
amit.deo@zama.ai

Louis Tremblay Thibault
Zama
Paris, France
École de technologie supérieure
Montréal, Canada
louis.tremblay.thibault@zama.ai

ABSTRACT

We propose Pool: a conceptually simple post-quantum (PQ) oblivious pseudorandom function (OPRF) protocol, that is round-optimal (with input-independent preprocessing), practically efficient, and has security based on the well-understood hardness of the learning with rounding (LWR) problem. Specifically, our design permits oblivious computation of the LWR-based pseudorandom function $F_{\mathsf{sk}}(x) = \lceil H(x)^\top \cdot \mathsf{sk} \rfloor_{q,p}$, for random oracle $H : \{0,1\}^* \mapsto \mathbb{Z}_q^n$ and uniformly chosen $\mathsf{sk} \in \{0,1\}^n$. For 128-bits of semi-honest security, the Pool OPRF has an online communication cost of 11.9 kB, and a computational runtime of less than 2 ms on a single thread (via an open-source software implementation). This is more efficient (in either online communication cost or runtime) than constructions from well-known PQ PRFs, and is competitive even with constructions that only *conjecture* PQ security on lesser-known assumptions. As a result, our design gives high-performance, post-quantum variants of established OPRF applications in multi-party computation and private set operation protocols.

1 INTRODUCTION

Oblivious pseudorandom function (OPRF) protocols [51, 60] permit a client to receive PRF evaluations $F_k(x)$ on inputs $x \in \{0,1\}^\ell$ of their choice, from a server holding the PRF key $k \leftarrow\!\!\$\ \{0,1\}^\lambda$. The key security property of such protocols is that the server learns nothing of the input x, and that the client learns $F_k(x)$ without compromising the pseudorandomness of the function $F_k(\cdot)$, from the client's perspective.

Over the last decade or so, OPRFs have come to form the bedrock of a large swathe of interesting cryptographic applications. In particular, they are a core primitive in protocols for computing private set operations (PSOs, e.g. [61, 62, 72]), private aggregation procedures (e.g. [46, 63]), and compromised credential checking mechanisms [64]. In terms of real-world deployments, the IETF standardization [44] of the (group-based) 2HashDH-based OPRF [57] has been instrumental in permitting further standardization efforts for protocols that handle privacy-preserving access token issuance [37, 45] and password-authenticated key exchange (PAKE) [28, 59] in the Internet setting. In particular, such efforts have seen such protocols developed and run at global-scale, for enhancing the privacy of Internet users (for instance, the company Cloudflare handles millions of OPRF evaluations per minute [67]). In summary, OPRFs are a core cryptographic tool that underpin the privacy of today's communication infrastructure.

With the success of the classical 2HashDH construction,[1] which can perform evaluations in less than a millisecond and in less than a hundred bytes, questions have arisen around the feasibility of developing post-quantum (PQ) variants of such protocols. In particular, while 2HashDH involves a client request that is perfectly private, the server key is subject to quantum adversaries that could invert discrete-log relations, that would ultimately violate the pseudorandomness property of said construction. Various constructions have arisen from lattice-based assumptions [5, 6, 50, 55], isogenies [15, 17, 26], and more heuristic constructions based on low-depth [2, 4, 49] and Legendre-based PRFs [21, 75].

Considerable effort has been poured into developing PQ (and plausibly PQ) OPRF constructions that attempt to match the practicality of 2HashDH, but a sizable gap remains. Table 1 summarizes the state-of-the-art, reporting the performance characteristics of existing constructions in a *semi-honest* setting, where adversaries attempt to subvert privacy within the confines of the protocol description. We elaborate more on related work in Section 1.2. As a synopsis, current constructions either: (i) are based on novel assumptions or are subject to active cryptanalysis, (ii) have stringent parameter requirements leading to large bandwidth costs, (iii) do not have concrete implementations exhibiting practical computational performance; or (iv) are not round optimal (i.e. more than two messages are transferred between client and server for OPRF evaluation). This situation naturally asks whether it is possible to construct round-optimal OPRF protocols from well-understood PRFs, while achieving credible performance for real-world applications.

Our work. We construct Pool, a round-optimal[2] OPRF, based on the security of the well-understood Learning With Rounding (LWR) assumption [14]. This assumption, and natural algebraic variants of it, were prominently used by constructions in the NIST PQC standardization call [10, 11, 20, 39, 43]. The underlying form of the PRF considers a secret key $\mathbf{k} \leftarrow\!\!\$\ \{0,1\}^n$ and a random oracle H, and the function itself is defined as $F_{\mathbf{k}}(x) = \lceil \frac{p}{q} \langle \mathbf{a}, \mathbf{k} \rangle \rfloor$, where $\mathbf{a} = H(x) \in \mathbb{Z}_q^n$. This form of PRF was first explicitly introduced in [27] and recently used in [48], with security following from the binary-secret LWR assumption. We discuss existing security reductions and the hardness of LWR further in Section 2.2.

Pool is very simple (see the technical overview in Section 1.1) and supports adding metadata (partial obliviousness), which is important for several applications [44]. In the preprocessing phase, we require n 1-out-of-2 oblivious transfer (OT) protocols, and a single 1-out-of-Δ OT (where $\Delta = q/p$) — both of which must satisfy PQ security. This can be done by using a PQ base OT construction, such as KyberOT [66]. In the online phase, all operations are essentially modular operations over $\mathbb{Z}_q, \mathbb{Z}_p$ and $\mathbb{Z}_\Delta$. The security proof of Pool

[1] As well as related constructions, such as [74].

[2] After input-independent preprocessing.

requires formally defining *preprocessing* adaptations (Section 3) to the POPRIV and PPOPRF security games that have been used in prior work [4, 50, 74]. Such models had previously gone undefined, despite several constructions and various applications (e.g., in the design of PSOs and PAKEs) aiming for said functionality. Our abstraction allows the formal usage of preprocessing-based OPRFs as black-boxes in higher-level primitives.

To measure efficiency, we implement Pool using libOTe to instantiate the OT functionalities [71] and compare it against state-of-the-art designs, such as Leap [55] (see Section 6.2). For a 128-bit security parameter and output length (LWR parameters of $q = 2^{12}$, $p = 2^8$, and $n = 482$), we achieve online bandwidth of 11.9 kB, server runtimes of 0.6 ms, and client runtimes of 1 ms; either outperforming or competitive with prior works. This includes those based on newer, less well-understood assumptions (where recent cryptanalysis applies). Compared with constructions based on well-known lattice assumptions (e.g. [5, 6]), we are the first to provide any concrete measure of efficiency. As a result, we believe Pool is a strong candidate for building secure and practical PQ OPRFs, with relevance to various Internet-based applications and standardization efforts.

Contributions. The formal contributions of this work follow.

- A simple, online round-optimal OPRF protocol, named Pool (Section 4), for evaluating the PRF $F_{\mathbf{k}}(\mathbf{s})$ defined above, where $F : \mathbb{Z}_q^n \times \mathbb{Z}_2^n \mapsto \mathbb{Z}_p$, for $p|q$, and n, q chosen such that the LWR assumption holds.
- Formal security analysis of Pool (Section 5), in a newly adapted variant of the game-based security model of [74], for handling semi-honest adversaries and preprocessing. Our definition (Section 3) separates the preprocessing phase explicitly from the online phase, leading to a clear online/offline separation in the construction and proof.
- An implementation[3] and experimental analysis (Section 6) demonstrating the practical efficiency of Pool.

1.1 Technical Overview

We start by giving an overview of a basic construction that can be instantiated directly from OT, and OT extension [56]. Briefly, 1-out-of-N OT allows a *receiver* to obtain 1 out of N strings picked by a *sender*. Crucially, the sender does not learn which string the receiver obtained, while the receiver does not gain any information on the remaining $N - 1$ strings. We explain the protocol in full, and then explain various optimizations that allow us to split the exchange into a very efficient online phase (the most expensive operations being $\mathbb{Z}_q$, $\mathbb{Z}_p$ and $\mathbb{Z}_\Delta$ arithmetic), and a slower offline preprocessing phase that includes Random OT (i.e. OT where the sent strings and receiver choice are uniformly distributed). We note throughout that we only target security against semi-honest adversaries that corrupt either the client or the server, and only compare with other solutions under this condition. In particular, semi-honest adversaries respect the protocol description but attempt to subvert the privacy of honest parties. We discuss protocol modifications that would achieve malicious security in Section 4.

Notation. Let λ be the security parameters, respectively. We use $\binom{N}{1}$-OT to refer to 1-out-of-N OT. We write $z \leftarrow\!\$\, \mathbb{Z}_m$ to say that z is sampled uniformly from the integers modulo m, $\mathbb{Z}_m$. Further, for any strictly positive integer n, we define $[n] := \{1, 2, \ldots, n-1, n\}$ and for integers $n_2 > n_1$, we define $[n_1, n_2] = \{n_1, n_1+1, \ldots, n_2-1, n_2\}$. For $x \in \mathbb{Z}_q$, we write $\lceil x \rfloor_{q,p}$ to indicate rounding $(p/q) \cdot x$ to the nearest integer (rounding down in the case of a tie). When we refer to a number of "rounds" in a protocol, we mean the total number of messages in a transcript.

Initial construction. Denote $(a_1, \ldots, a_n) = H(x)$, where $H : \{0,1\}^\ell \mapsto \mathbb{Z}_q^n$ is modeled as a random oracle, and $x \in \{0,1\}^\ell$ is the client input in the OPRF. Further, let $\mathbf{s} = (s_1, \ldots, s_n) \in \{0,1\}^n$ be a binary LWR secret key owned by the server.

Step 1: For $i = 1 \ldots n$, the client plays the role of $\binom{2}{1}$-OT sender with $(r_i, a_i + r_i)$, for some random mask $r_i \leftarrow\!\$\, \mathbb{Z}_q$, and the server plays the role of $\binom{2}{1}$-OT receiver, with choice bit s_i. As a result, the server receives $a_i \cdot s + r_i$.

Step 2: The server adds all of the vectors it received to obtain $c = \mathbf{a}^\top \mathbf{s} + r_\Sigma$ where $r_\Sigma = \sum_{i \in [n]} r_i$.

Step 3: Client plays a $\binom{q}{1}$-OT receiver with choice $r_\Sigma \in \mathbb{Z}_q$ and server plays the sender with elements

$$(\lceil c - 0 \rfloor_{q,p}, \lceil c - 1 \rfloor_{q,p}, \ldots, \lceil c - (q-1) \rfloor_{q,p}).$$

The client, therefore, receives $\lceil c - r_\Sigma \rfloor_{q,p} = \lceil \mathbf{a}^\top \mathbf{s} \rfloor_{q,p}$, which is the correct PRF output. We note that the techniques used above bear resemblance to those used in the Leap OPRF [55]. Client privacy is maintained via the functionality of OT and the uniformity of the blinding factors r_i. For server security, the OT in step 3 guarantees that the client learns the PRF output, and nothing more. To instantiate $\binom{q}{1}$-OT, we can use [61] or [69] and use IKNP [56] or Silent OT [29] for the base OT extension of the $\binom{2}{1}$-OT, where the PQ base OT is implemented using KyberOT [66]. Concretely, we can set n and (p, q) to ensure that the LWR PRF is secure using the well-known Lattice Estimator [7][4].

Number of Rounds. Steps 1 and 3 both use OT and may be aided by preprocessing. Using OT preprocessing, a single *online* OT interaction usually takes two rounds of communication [19]. However, if the choice bits of the receiver are known during the preprocessing phase, only one round of communication is required. Luckily, the server can know its choice bits (i.e. its secret key) during the preprocessing phase, so step 1 is a single round of communication from client to server. On the other hand, step 3 costs two online rounds as the client's choice bits cannot be known during the preprocessing phase in a normal application. Therefore step 3 consists of a message from client to server (a few correction bits), and a response. However, the message from client to server can be sent with the message from step 1, reducing communication to just two rounds in the online phase.

Parameters. Suppose we want 128 bits of output. This requires generating $128/\log(p)$ output elements in parallel. The estimated communication cost for 128 bits of output is then

$$([2n\log(q) + \log(q)] + [q * \log(p)]) \cdot \frac{128}{\log(p)} \text{ bits.} \qquad (1)$$

where the first square bracket is the client's message and the second square bracket is the server message. Example parameters for 100

[3] https://github.com/tremblaythibaultl/ot-pq-oprf

[4] https://github.com/malb/lattice-estimator

Construction	Assumption	Online Rounds	Preprocessing	Online comms.	Output bits
[49]	Alternating Moduli [25]	2	✓	80B†	128
[2]	Alternating Moduli [2]	2	✓	114.5B†	128
Leap [55]	Spring-BCH PRF [12]	3	✓	23KB	64
Pool	Binary LWR (100 bits sec.)	2	✓	9.2KB	128
Pool	Binary LWR (128 bits sec.)	2	✓	11.9KB	128

Table 1: Brief comparison of Pool with existing *semi-honest* OPRF constructions believed to resist quantum attacks, and for security levels of 100 bits, or higher. †Communication value corresponds to a batched evaluation setting, where costs of a single query are amortized over the total batch size. We note that while we can offer well-grounded security estimates for Pool, prior constructions are based on assumptions/parameters that have not been well-studied, and thus security guarantees are not stable (see Section 1.2 for more details). We provide a comparison with maliciously-secure protocols in Appendix A.

bits of security from the lattice estimator are $n = 370, p = 2^8, q = 2^{12}$, leading to an estimated 83 kB online cost.

Reducing the size of Step 3. As an optimization, we can exploit the highly structured form of the length-q vector computed by the server in step 3. This vector is simply a cyclic shift of the vector

$$(\lceil 0 \rfloor_{q,p}, \ldots, \lceil q-1 \rfloor_{q,p}) = (\overbrace{0, \ldots, 0}^{\Delta \text{ times}}, \overbrace{1, \ldots, 1}^{\Delta \text{ times}}, \ldots, \overbrace{p-1, \ldots, p-1}^{\Delta \text{ times}}),$$

where each block of repeated elements has length $\Delta := q/p$. It follows that the vector computed by the server is fully determined by its first Δ elements. Therefore, it is enough for the client to learn the index $r_\Sigma \bmod \Delta$, and then infer the value at index r_Σ. This can be done by using the fact that entries of the length-q vector that are $k \cdot \Delta$ apart differ in value by $k \bmod p$. Note that learning the value of the database at index $r_\Sigma \bmod \Delta$ leaks no more information that directly learning the value at index r_Σ due to the aforementioned structure. With this in mind, we can amend Step 3 to take the form:

> **Step 3:** The client plays $\binom{\Delta}{1}$-OT receiver with choice element $r_\Sigma \bmod \Delta$ and the server plays the sender with elements $(\lceil c-0 \rfloor_{q,p}, \lceil c-1 \rfloor_{q,p}, \ldots, \lceil c-(\Delta-1) \rfloor_{q,p})$.

Writing $r_\Sigma = \Delta \cdot k + (r_\Sigma \bmod \Delta)$, this results in the client receiving $y = \lceil c - (r_\Sigma - \Delta \cdot k) \rfloor_{q,p} = \lceil \mathbf{a}^\top \mathbf{s} \rfloor_{q,p} + k$. Therefore, the client can output $y - k \bmod p$ as its final output. With this change, the estimated communication cost for 128 bits of output decreases to:

$$([2n \log(q) + \log(\Delta)] + [\Delta \cdot \log(p)]) \cdot \frac{128}{\log(p)} \text{ bits.}$$

by replacing q by Δ in step 3. Using the same example parameters as previously, we therefore obtain an online communication cost of around 18 kB.

Reducing client online message. We now discuss the preprocessing/online phases of the protocol (as described up to this point) in slightly more detail. Overall, the client and server perform *random* n $\binom{2}{1}$-OTs (client as sender) and one $\binom{\Delta}{1}$-OT (server as sender) in the preprocessing phase to learn a sequence of masks. The server then sends their OT inputs for the n $\binom{2}{1}$-OTs but shifted by their secret key, i.e. $\bar{b}_i = b_i \oplus \mathsf{sk}_i$. In the online phase, the client sends the message $(r_i + r_{0,i}, a_i + r_i + r_{1,i})$, where r_i are ephemeral masks sampled in the online phase, and the value of $r_{s_i,i}$ is learned by the server in the preprocessing phase (whereas $r_{1-s_i,i}$ remains hidden). The server can then either unmask the first value (if $\mathsf{sk}_i = 0$), or the second (if $\mathsf{sk}_i = 1$) to obtain $\tilde{a}_i = a_i s_i + r_i$.

Instead of sending $(r_i + r_{0,i}, a_i + r_i + r_{1,i})$, the client can just send $a_i + r_{0,i} + r_{1,i}$. If $\mathsf{sk}_i = 0$, the server knows $r_{0,i}$, whereas if $\mathsf{sk}_i = 1$ the server can unmask to learn $a_i + r_{0,i}$. In essence, $r_{0,i}$ takes the place of the mask r_i. The cost then reduces considerably to:

$$([n \log(q) + \log(\Delta)] + [\Delta \cdot \log(p)]) \cdot \frac{128}{\log(p)} \text{ bits.} \qquad (2)$$

Using the aforementioned parameters, the concrete online cost therefore decreases to just over 9 kB. Our full protocol, including each of the tweaks and optimizations detailed here, is described in Figure 4.

Proving security. From a security perspective, our OPRF construction is split between a preprocessing step containing Random OT invocations, and an online step involving various masked messages. A natural, and seemingly benign, question is, what security model should we use? Unfortunately, models that support a preprocessing phase are not formally defined in previous works. In particular, OPRF constructions normally use security notions that are either game-based ([4, 5, 45, 74]), presented in the universal composability model ([21, 55, 57]), or are left formally undefined ([2, 49]).

We address this gap by formally specifying a game-based security notion that allows embedding a preprocessing phase into the exchange, based on the PPOPRF and POPRIV1 security notions defined in [74]. We also introduce a parameter τ that dictates how many OPRF evaluations a single preprocessing run should facilitate. In the case of PPOPRF, we define SH-PPOPRF (Figure 1) requiring the existence of a simulator that simulates the view of a semi-honest client interacting with both the preprocessing and online phases. Importantly, the simulator does not possess any server secret key, and only uses an appropriate number of PRF values during the simulation. This number is limited to one per client online OPRF request, and τ per individual (simulated) preprocessing output state. In the case of POPRIV1, we also amend the original definition to also provide explicit oracle access to a preprocessing oracle PreProc. Ultimately, we obtain an easy-to-use game-based definition for preprocessing OPRFs that are optimal in the online phase.

With the given model, we describe our scheme and proof in an OT-hybrid model, where client and server has access to random

idealized OT functionalities (see Appendix C). The proof against a corrupted server (Theorem 5.3) holds unconditionally in the OT-hybrid model, and the proof against a corrupted client (Theorem 5.2) additionally relies on the binary LWR assumption. We realize the OT functionality and report on concrete performance in Section 6.

Limitations. Our construction is only proven secure in the semi-honest security setting where clients and servers respect the protocol description. To upgrade to malicious client security, one could use maliciously secure OT primitives and add zero-knowledge proofs and extractable commitments to the preprocessing and online phases of the OPRF. This is in line with previous maliciously-secure constructions based on lattices [4, 5, 50, 55]. We discuss such changes more specifically in Section 4.

1.2 Related Work

We now summarize the various kinds of post-quantum OPRF constructions. Note that classical/algebraic OPRFs were introduced and built upon in [51, 60] for applications associated with private set operations. Subsequently, various constructions have arisen (e.g. [57, 58, 74]) giving rise to applications for anonymous credentials [45], password-authenticated key exchange [59], privacy-preserving data aggregation [46], private matching protocols [68], amongst many other applications. With many such applications having been deployed across the Internet, developing *practical* PQ alternatives has considerable importance. We encourage the reader to refer to the excellent survey of [34] for more details.

Lattice-based. The work of [5] gave the first (round-optimal) PQ OPRF construction from ideal lattices. Depending on the RLWE and Ring-SIS problems, it computes an OPRF of the form $\lceil a_x \cdot k \rfloor_{q,p}$, for $a_x \in R$ derived from the client input $x \in \{0,1\}^\ell$, the (short) server key $k \in R$. The client request blinds a_x and the response allows the client to compute $a_x \cdot k + e$ before rounding to achieve the PRF. In the end, the parameters are impractical due to using super-polynomial noise flooding, and the usage of expensive zero-knowledge proofs. Subsequent work [6] greatly improves on this design by adapting the PRF design and using Renyi divergence. An estimated online bandwidth cost of 70 kB + 110 kB can be derived from [6, Tab.2] (where the second summand represents the costs of zero-knowledge proofs). The LeOPaRd construction takes an alternative route to improving [5] using new variants of the module LWE problem,[5] improving on performance relative to [6], but with less well-understood security. None of these constructions have implementations that evaluate their concrete runtimes.

The very recent Leap construction [55] uses the lattice-based Spring-BCH PRF [12] (a Naor-Reingold-like product $\mathbf{k}_0 \cdot \prod_{i=1}^{\ell} \mathbf{k}_i^{x_i}$, for key elements $\mathbf{k}_i$ and input $x \in \{0,1\}^\ell$), to build an OPRF. This construction is similar to Pool, utilizing Random OT and masking/unmasking to perform the evaluation. However, the preprocessing steps also utilize Oblivious Linear Evaluation (OLE) protocols. The overall protocol (that is proven secure) contains six rounds, though it is noted that some of these rounds can be moved into a preprocessing phase, to give just three online rounds. The OPRF has an open-source implementation[6] and is very efficient, requiring only 23 kB of communication, and milliseconds of evaluation time, to produce 64 bits of output. However, while the structural design of the Spring-BCH design does have a security proof, the parametrizations are purposefully chosen for efficiency and do not satisfy the requirements of the security proof. In order to justify parameter choices, bespoke direct cryptanalysis is used on the PRF, rather than on any single well-established lattice assumption. Given the efficiency of Leap and the similarities with our design, we provide a detailed comparison of performance in Section 6.2.

Isogeny-based. Since the initial construction of Boneh, Kogan, and Woo [26], various constructions have arisen from hardness assumptions constructed over isogenies [15, 17]. Furthermore, very recently, the CSIDH group has been used to build algebraic-like OPRFs [47]. Several attacks and insecurities have been found in some of these constructions, that have either completely broken existing schemes [16] or resulted in defining more conservative (and less efficient) parameter choices [18]. In general, while communication can match those of the most efficient lattice-based OPRFs, concrete implementations and runtimes are unknown.

From Legendre symbols. The work of [21] constructs a generic template for establishing 2HashDH-like [57] OPRF protocols, where the individual components can be instantiated with PQ components, to achieve post-quantum security. 2HashDH protocols construct outputs of the form: $H_2(x, H_1(x)^k)$, where the client sends $H_1(x)$ blinded to the server (e.g. $X = H_1(x)^r$) and the server computes $Y = X^k$ for their key k. The client then computes $Z = Y^{-r}$, which, assuming commutativity in the exponent, allows them to compute the final output of $H_2(x, Z)$. The original instantiation of 2HashDH is presented in a cyclic prime-order group, with H_1 and H_2 modeled as random oracles, but [21] presents a generalized framework based on a secure function evaluation (SFE) protocol for evaluating $f(H_1(x), k)$. The work demonstrates two potential constructions, one based on Legendre symbols and oblivious transfer, and another based on block ciphers. The construction is based on Legendre symbols, which have been used as a candidate PQ construction of pseudorandom generators since an initial design of Damgård [42]. The SFE protocol for the Legendre symbol PRF can be built from PQ instantiations of VOLE. An open-source implementation is provided, that shows that the base online OPRF protocol (without network delays etc.) evaluations takes just under 200 ms to execute, and with a communication cost of around 200 kB. The Gold OPRF construction [75] also uses Legendre symbols as a base OPRF, and demonstrates a mechanism to perform evaluations based directly from OT. Their construction requires around 774 kB for a single evaluation, but this can be amortized considerably (e.g. to only 100 B) in a batched evaluation setting.

From low-depth weak PRFs. Recent studies determine that low-dimensional linear operations, followed by lifting and reducing values to different moduli, may give enough non-linearity to establish weak PRFs. For example, one of the "Dark Matter" PRF candidate constructions [25] evaluates $(\mathbf{A} \cdot \mathbf{x} \bmod 2) \cdot 1^n \bmod 3$ for $\mathbf{A} \in \mathbb{Z}_2^{n \times m}$ a secret key, and $\mathbf{x} \in \{0,1\}^m$ a randomly sampled input. Various other constructions have suggested paths forward for constructing constant-depth PRFs from simple binary operations [1, 30]. However, cryptanalysis has broken some constructions [24], or

[5] Enjoying reductions to MLWE-like problems by making idealized assumptions.
[6] While Leap itself is proven secure in a malicious setting, the implementation is only secure against semi-honest adversaries.

demonstrated weaknesses that imply revising parameters to be more conservative [38], but the example above remains unbroken.

With this in mind, various works have taken to using the Dark Matter PRF, and slight variants of it (so-called "Alternating Moduli" PRFs), as the basis for constructing *plausibly* PQ OPRFs. The work of [4] constructs an OPRF protocol for evaluating the PRF above, that involves encrypting the client's input x with an FHE scheme, and then having the server evaluate the circuit $F_{\mathbf{A}}(\cdot)$ over this input, and return the result to the client for decryption. Using TFHE [41], the protocol incurs communication of less than 10 kB, but the requirement for bootstrapping enforces an expensive one-time cost of tens of megabytes, plus expensive runtimes. The works of [2, 49] propose mechanisms based on OT for evaluating specific alternating moduli PRFs. Both constructions are shown to be very efficient for batched evaluations, incurring only around 100 B of (amortized) communication, and microseconds in runtime. However, recent cryptanalysis appears to show that the concrete security of these constructions is relatively unknown [9]. At the least, it is required to set parameters more conservatively than the original work suggests, and the possibility of future, more systematic attacks remains. Furthermore, neither of [2, 49] provide explicit security proofs for their OPRF constructions. Finally, implementations are not provided open-source, and efficiency over non-batched evaluations is not reported.

Preprocessing vs one-time setup. Various OPRFs require a one-time setup, such as the publishing of a server public key (e.g. [57]), or likewise for the client to register themselves with the server (e.g. [4]). On the other hand, preprocessing OPRFs require the client and server perform a potentially expensive offline protocol, which can then be amortized across various online phases (e.g. [55]). In the applications we consider, such as PSOs and MPC, preprocessing phases fit into the API of the higher-level protocols, so we do not consider them a hindrance. A full summary of the existing semi-honest OPRF protocols believed to resist quantum attacks (including Pool) is given in Table 1, where we note that all such known constructions utilize preprocessing. In addition, we provide a performance comparison with malicious OPRF protocols in Appendix A, but where zero-knowledge proof and commitment costs are removed to ensure a fair comparison.

2 BACKGROUND

Notation. We continue to use the notation defined in Section 1.1. For integers p and q we write $p \mid q$ to denote that p divides q exactly. A probabilistic polynomial time (PPT) algorithm is a probabilistic algorithm that runs in time bounded by some polynomial function of λ. We say that an algorithm $\mathcal{A}$ has oracle access to another algorithm $\mathcal{B}$ if it can submit inputs to $\mathcal{B}$ and obtain the corresponding outputs. In particular, $\mathcal{A}$ has no knowledge of the inner workings of $\mathcal{B}$ and only has input-output access. To make explicit that $\mathcal{A}$ has oracle access to $\mathcal{B}$, we write $\mathcal{A}^{\mathcal{B}}$. Furthermore, we will often assume that algorithms are stateful. For such algorithms, it will be assumed that the state may be modified whenever the algorithm is called. For example, take $\mathcal{A}$ to be an algorithm with state st and input x. When we write $y \leftarrow \mathcal{A}(x; \mathrm{st})$, we implicitly assume that $\mathcal{A}$ has the ability to modify the state st when it is run.

For a distribution X, $x \leftarrow\!\$\ X$ denotes sampling x according to X. Similarly, writing $x \leftarrow\!\$\ S$ for a finite set S denotes uniformly sampling of x from S. We use the notation that $U(S)$ denotes the uniform distribution over the set S. We say two distribution ensembles $(X(\lambda))_\lambda$ and $(Y(\lambda))_\lambda$ are computationally indistinguishable if for all PPT algorithms $\mathcal{A}$, $|\Pr_{x\leftarrow\!\$ X}[\mathcal{A}(x) \Rightarrow 1] - \Pr_{x\leftarrow\!\$ Y}[\mathcal{A}(x) \Rightarrow 1]| \leq \nu(\lambda)$ for some negligible function in λ, $\nu(\lambda)$. We recall that the usual definition of a negligible function is any function that is $\lambda^{-\omega(1)}$ where $\omega(\cdot)$ is standard asymptotic notation. The notation $\mathrm{negl}(\lambda)$ denotes an unspecified negligible function.

We will be using the well-known random oracle model (ROM) where random oracles will be realized using a suitable hash function. Denoting an idealized random oracle RO, we have that on any input x, $\mathrm{RO}(x)$ is distributed uniformly in the range of RO.

2.1 Oblivious Transfer

Random oblivious transfer (rOT) protocols over message-space $\mathcal{M}$ and choice-space $\mathbb{Z}_\gamma$ can be informally understood as protocols that securely realize the functionality $f(\perp, \perp) = ((b, r_b), (r_i)_{i=0}^{\Delta-1})$ where $b \in \mathbb{Z}_\gamma$, $r_i \in \mathcal{M}$ are all uniformly distributed. That is, on input $\perp$ from a receiver R and input $\perp$ from a sender S, an rOT protocol aims to provide S with uniform output $(r_i)_{i=0}^{\gamma-1}$ and R with output (b, r_b) for a uniformly chosen $b \in \mathbb{Z}_\gamma$. A secure rOT protocol ensures that R and S learn nothing beyond their outputs. When $\gamma = 2$, we get the usual 1-out-of-2 Random OT or "$\binom{2}{1}$-Random OT". It is well known that random-OT can be used to build the more standard notion of OT where R and S choose b and (r_i) respectively [19]. However, we choose to work with random-OT as it leads to a more natural "preprocessing" construction.

Our construction uses a Random OT functionality with two different parametrizations. In Figure 5 of Appendix C, we formally define the two Random OT functionalities (using the formulation of [33]) in the universally composable (UC) model [31]. The advantage of using UC secure formulations of Random OT functionalities is that we may arbitrarily compose realizations of these idealized functionalities without compromising security. The functionalities we use are denoted $\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ and $\mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}$. The former has $(\gamma, \mathcal{M}) = (\mathbb{Z}_\Delta, \mathbb{Z}_p)$ whereas the latter has $(\gamma, \mathcal{M}) = (\mathbb{Z}_2, \mathbb{Z}_q^\tau)$. In both cases, the parameter ℓ represents the number of individual OT correlations output by the functionality in both cases.

We may design constructions that have access to the two idealized Random OT functionalities $\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ and $\mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}$. Any security properties proven to hold for such constructions are said to hold in the "idealized $\left\{\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}, \mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}\right\}$-hybrid model". To move from this model to a construction in the real world, one simply needs to replace the functionalities with a UC-secure realization.

2.2 Learning With Rounding

The Learning With Rounding (LWR) assumption was introduced by Banerjee, Peikert and Rosen [14]. LWR can be seen as a variant of the Learning With Errors (LWE) problem [73] with a deterministic error which makes it a particularly useful assumption in the

design of PRFs. The LWR assumption and its natural algebraic variants were used by numerous candidates in the recent NIST PQC standardization call [10, 11, 20, 39, 43].

Definition 2.1. For integers $m(\lambda), n(\lambda), p(\lambda), q(\lambda)$ and distribution $\mathcal{D}$ over $\mathbb{Z}^n$, the $\mathsf{LWR}^{\mathcal{D}}_{m,n,p,q}$ assumption states that the distribution of $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} \rfloor_{q,p})_{i=1}^m$ is computationally indistinguishable from uniform over $(\mathbb{Z}_q^n \times \mathbb{Z}_p)^m$ for $\mathbf{a}_i \leftarrow\$ \mathbb{Z}_q^n, \mathbf{s} \leftarrow\$ \mathcal{D}$.

Equivalently, if X denotes the the distribution of $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} \rfloor_{q,p})_{i=1}^m$ and U denotes the aforementioned uniform distribution, then for all PPT adversaries $\mathcal{A}$, the following advantage is $\mathsf{negl}(\lambda)$:

$$\mathsf{Adv}^{\mathsf{LWR_{n,p,q}}}_{\mathcal{A}}(\lambda) := |\Pr_{x \leftarrow\$ X}[\mathcal{A}(x) \Rightarrow 1] - \Pr_{u \leftarrow\$ U}[\mathcal{A}(u) \Rightarrow 1]|.$$

The parameter m is referred to as the number of samples, n is referred to as the dimension and (p, q) are moduli. $\mathsf{LWR}^{\mathcal{D}}_{n,p,q}$ with m omitted is the LWR assumption for any polynomial number of samples m. We will use the LWR problem where secrets are sampled uniformly over $\{0,1\}^n$. As short-hand notation, we will write this as $\mathsf{LWR}^{\{0,1\}}_{m,n,p,q}$ and refer to this problem as binary secret LWR.

Although there are reductions from standard LWE to LWR, they either require (i) a super-polynomial modulus and modulus-to-error ratio [14] or (ii) that the number of samples m to be known ahead of time [8]. These reductions provide very useful theoretical evidence that the LWR problem is hard. On a less positive note, Newton and Richelson [70] showed that there cannot exist a "point-wise" reduction from a hard LWE problem to LWR if $q > p^{2/3+c}$ for some small constant c and prime q. However, we stress that only reductions lying in a particular model are considered here. In particular, the reductions considered are not allowed to reject samples and must map each individual LWE sample to an individual LWR sample. By utilizing rejection, one can obtain a reduction from the LWE problem with uniform error distribution in $[-q/(2p), +q/(2p))$ to LWR that holds for *any* secret distribution [23] whenever p divides q. This particular reduction relates closely to concrete cryptanalysis. In particular, state-of-the-art parameter selection for LWR considers the hardness of the LWE problem with uniform error in the aforementioned interval [3]. In practice, one may find secure parameters using the well-known lattice-estimator tool[7] [7]. We note that binary LWE keys are practically prominent, and are a key feature of the TFHE fully homomorphic encryption scheme [40, 77].

3 ADAPTING THE (P)OPRF SECURITY MODEL

We now present the definition of a (round optimal) preprocessing, partial OPRF or (PPOPRF) following [74]. Previous work using these definitions exclude OPRF candidates that function with an offline "preprocessing" phase. Consequently, we adapt these definitions to include a PreProc algorithm into the syntax/definitions. We will assume here that the adversaries are semi-honest i.e. that they follow the protocol honestly. In reality, the preprocessing phase will be an interactive protocol between a client and server. However, since we are working in the semi-honest model, the preprocessing phase will be modeled as an efficient probabilistic algorithm PreProc that honestly emulates a real world preprocessing interaction. The output of PreProc is split into two distinct parts – the client state and the server state. The client/server state contains the client/server's view of the preprocessing interaction emulated by the preprocessing algorithm. In order to have a realizable construction, one should ultimately describe the preprocessing algorithm as an interaction between a client and server.

Importantly, the PreProc algorithm takes only the server key as input. This models the common case where the server chooses its key before OPRF clients are enrolled (e.g. in the case of Privacy Pass [37]). After the preprocessing phase, the client is able to use its state to run the OPRF on any input of its choice. We recall that RO in a superscript denotes oracle access to a hash function that will be modeled as a random oracle. In order to recover the definition of a preprocessing OPRF where there is no public tag/input, the value of t in the following definition can be fixed, e.g. $t = \perp$.

Definition 3.1 (Adapted from [74]). A preprocessing partial oblivious PRF (PPOPRF) is a tuple of PPT algorithms $\mathcal{F}$ = (Setup, KeyGen, PreProc, Request, BlindEval, Finalize, Eval).
$\mathcal{F}$.Setup generates public parameters pp and initializes the initial global server state $\mathsf{Gst_S}$ to $\perp$. $\mathcal{F}$.KeyGen samples a public/secret key pair (pk, sk). The preprocessing algorithm $\mathcal{F}$.PreProc uses pp with input sk to output a client state $\mathsf{st_C}$ and a local server state $\mathsf{st_S}$ that is automatically appended to the global server state $\mathsf{Gst_S}$.
Oblivious evaluation is an interactive protocol between C and S, here presented as algorithms $\mathcal{F}$.Request, $\mathcal{F}$.BlindEval, $\mathcal{F}$.Finalize working as follows:

(1) First, C runs the algorithm $\mathcal{F}.\mathsf{Request}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{pk}, t, x; \mathsf{st_C})$ using state $\mathsf{st_C}$, taking a public key pk, a "tag" or public input t and a private input x. It updates its state $\mathsf{st_C}$ and outputs a request message req or an abort symbol $\perp$, which is sent to the server.
(2) S runs $\mathcal{F}.\mathsf{BlindEval}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{sk}, t, \mathsf{req}; \mathsf{Gst_S})$ taking as input a secret key sk, a tag t and the request message req using global state $\mathsf{Gst_S}$. It produces a response message rep or an abort symbol $\perp$ sent back to C and updates its global state $\mathsf{Gst_S}$.
(3) Finally, C runs $\mathcal{F}.\mathsf{Finalize}(\mathsf{rep}; \mathsf{st_C})$ which takes the response message and its previously constructed state $\mathsf{st_C}$ and either outputs a PRF evaluation or $\perp$ if rep is rejected.

The unblinded evaluation algorithm $\mathcal{F}$.Eval is deterministic and takes as input a secret key sk, an input pair (t, x) and outputs a PRF evaluation z.

We also define sets $\mathcal{F}$.SK, $\mathcal{F}$.PK, $\mathcal{F}$.T, $\mathcal{F}$.X and $\mathcal{F}$.Out representing the secret key, public key, tag, private input, and output space, respectively. We define the input space $\mathcal{F}.\mathsf{In} = \mathcal{F}.\mathsf{T} \times \mathcal{F}.\mathsf{X}$. We assume efficient algorithms for sampling and membership queries on these sets. □

The correctness definition below captures the fact that a single run of the preprocessing algorithm may be used to obtain multiple evaluations. The parameter τ denotes the number of evaluations possible.

Definition 3.2 (PPOPRF Correctness). For an integer $\tau \geq 1$, a PPOPRF

$\mathcal{F}$ = (Setup, KeyGen, PreProcRequest, BlindEval, Finalize, Eval)

[7] Available at https://github.com/malb/lattice-estimator

is τ-correct if for any server global state $\mathsf{Gst_S}$, and any $(t_j, x_j)_{j\in[\tau]}$,

$$\Pr\left[\begin{array}{l} z_1{=}\mathcal{F}.\mathsf{Eval}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{sk},t_1,x_1) \\ \wedge\ z_2{=}\mathcal{F}.\mathsf{Eval}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{sk},t_2,x_2) \\ \wedge\ \ldots \\ \wedge\ z_\tau{=}\mathcal{F}.\mathsf{Eval}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{sk},t_\tau,x_\tau) \end{array} \middle| \begin{array}{l} \mathsf{pp}\leftarrow\$\mathcal{F}.\mathsf{Setup}(1^\lambda) \\ (\mathsf{pk},\mathsf{sk})\leftarrow\$\mathcal{F}.\mathsf{KeyGen}^{\mathsf{RO}}_{\mathsf{pp}}(1^\lambda) \\ \mathsf{st_C},\mathsf{st_S}\leftarrow\$\mathcal{F}.\mathsf{PreProc}_{\mathsf{pp}}(\mathsf{sk}) \\ \mathsf{Gst_S}\leftarrow\mathsf{Gst_S}\|\mathsf{st_S} \\ \textbf{for } j\in[\tau]\text{:} \\ \quad \mathsf{req}_j\leftarrow\$\mathcal{F}.\mathsf{Request}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{pk},t_j,x_j;\mathsf{st_C}) \\ \quad \mathsf{rep}_j\leftarrow\$\mathcal{F}.\mathsf{BlindEval}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{sk},t_j,\mathsf{req}_j;\mathsf{Gst_S}\|\mathsf{st_S}) \\ \quad z_j\leftarrow\$\mathcal{F}.\mathsf{Finalize}^{\mathsf{RO}}_{\mathsf{pp}}(\mathsf{rep}_j,\mathsf{st_C}) \end{array}\right] = 1.$$

We next define pseudorandomness of a PPOPRF [74] against semi-honest clients adapted to a preprocessing model. In order to obtain a definition for semi-honest adversaries, we alter the BlindEval oracle from [74] so that it takes a PRF input rather than an arbitrary PPOPRF request. The oracle then computes an honestly generated request message and returns the server response to the adversary. $\mathsf{Gst_{Sim}}, \mathsf{Gst}_S, \mathsf{Gst}_C$ are the global states of a simulator, honest server and semi-honest client respectively.

Intuitively, the definition captures the notion that a semi-honest adversarial client $\mathcal{A}$ cannot distinguish between the real world (with $b = 1$) where it interacts with a true implementation of $\mathcal{F}$, or an ideal world (with $b = 0$) where preprocessing and blind evaluation are simulated without using a secret key and PRF evaluation is replaced by an idealized truly random function (denoted $\mathsf{RO_{Fn}}$). A key point is that the simulation only uses values known to the adversarial client throughout its execution and a limited number of $\mathsf{RO_{Fn}}$ outputs that it obtains via a LimitEval oracle. Specifically, the number of $\mathsf{RO_{Fn}}$ calls available to the simulator is bound by the number of times preprocessing (multiplied by the parameter τ) has been called and the number of blind evaluation queries made by the adversary. This implies that the client essentially learns nothing about the secret key during its real-world interaction (beyond the PRF evaluations corresponding to its queries).

In the security game (fig. 1), the oracle $\textsc{Prim}(x)$ provides oracle access to a random oracle that may be used in the PPOPRF construction. This oracle appears in the definition from [74]. For the case $b = 0$ the simulator is allowed to use only a limited number of $\mathsf{RO_{Fn}}$ queries to simulate the random oracle accessed via $\textsc{Prim}(x)$. This is particularly useful for OPRF designs that make extensive use of hash functions to obtain PRF output such as 2HashDH [57] and related designs. Again, the bounds in LimitEval ensure the simulator only uses an appropriate number of random function ($\mathsf{RO_{Fn}}$) calls to simulate BlindEval/Prim responses (i.e., at most one per client request and no more than τ multiplied by the current number of preprocessing calls).

Definition 3.3 (Pseudorandomness). We say a PPOPRF $\mathcal{F}$ is pseudorandom against semi-honest clients with τ-reusable preprocessing if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator Sim such that the following advantage is $\mathsf{negl}(\lambda)$:

$$\mathsf{Adv}^{\text{sh-ppoprf}}_{\mathcal{F},\mathsf{Sim},\mathsf{RO},\mathcal{A}}(\lambda) := \left|\Pr\left[\text{SH-PPOPRF}^{\mathcal{A},1}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda) \Rightarrow 1\right] - \Pr\left[\text{SH-PPOPRF}^{\mathcal{A},0}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda) \Rightarrow 1\right]\right|,$$

where $\text{SH-PPOPRF}^{\mathcal{A},b}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$ is defined as in Figure 1.

The remaining security notion in the semi-honest model considers an adversarial server that wishes to passively learn the private inputs of clients. The definition is again heavily inspired by [74] with the addition of an honestly performed preprocessing algorithm. The security game and oracles are presented in fig. 2. The definition gives a semi-honest server access to a PreProc oracle and an oracle Run that takes as input two client inputs (x_0, x_1) and two "pointers" to preprocessing states (i_0, i_1). For challenge bit $b \in \{0, 1\}$, the Run oracle first runs $\mathcal{F}$.Request() on input x_b using the state pointed to by i_0 and then produces an honest response and finalized evaluation. We call the (request, response, finalize) triple $(\mathsf{req}_0, \mathsf{rep}_0, z_0)$. It then proceeds to do the same for input x_{1-b} using the state pointed to by i_1 yielding the triple $(\mathsf{req}_1, \mathsf{rep}_1, z_1)$. If $b = 0$, then Run returns the two triples as they are whereas if $b = 1$ the triples with z_0 and z_1 swapped are returned. The definition requires that the adversarial server $\mathcal{A}$ cannot effectively distinguish between the two cases, implying that a semi-honest server cannot distinguish between transcripts corresponding to inputs of its choice.

We note that this "triple" formulation follows the blueprint of [74] in such a way that the triple T_0 in fig. 2 always uses the state pointed to by i_0 and T_1 always uses the state pointed to by i_1. Further, the z-entry in T_j always contains a finalize run for a response/requests on x_j for $j \in \{0, 1\}$. A caveat is that if Run encounters $\perp$ at any point, it returns $\perp$. This prevents an adversary from distinguishing the cases $b = 0$ and $b = 1$ by using state pointers where one pointer yields a valid request/response and the other yields a request/response that equals $\perp$.

Definition 3.4 (Request Privacy (POPRIV1)). We say a PPOPRF $\mathcal{F}$ has request privacy against a semi-honest server if for all PPT adversary $\mathcal{A}$ the following advantage is $\mathsf{negl}(\lambda)$:

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

where $\text{POPRIV1}^{\mathcal{A},b}_{\mathcal{F},\mathsf{RO}}(\lambda)$ is defined in Figure 2.

Malicious Security. We now informally discuss how to derive maliciously secure definitions. In order to define PPOPRFs with malicious security, one could replace the PreProc algorithm by an idealized functionality $\mathcal{F}_{\mathsf{PreProc}}$ that interfaces with malicious clients and servers. In this case, access to PreProc in the definitions would be replaced by malicious access to the idealized $\mathcal{F}_{\mathsf{PreProc}}$ functionality. The remaining oracles for malicious client/server security that do not depend on PreProc could then be replaced by their counterparts in the malicious-security definitions from [74] (with the caveat that an extra input that points to a particular preprocessing state may be required). One would then write definitions and proofs in a $\mathcal{F}_{\mathsf{PreProc}}$-hybrid model where a UC secure realization of the preprocessing functionality would be required for a concrete instantiation. We leave formalizing these notions as future work since we focus on semi-honest security.

Game $\text{SH-PPOPRF}^{\mathcal{A},b}_{\mathcal{F},\text{Sim},\text{RO}}(\lambda)$

$q_{s,t}, q_t, q_p \leftarrow 0, 0, 0$
$\text{st}_{\text{Fn}} \leftarrow\$ \text{RO}_{\text{Fn}}.\text{Init}(1^\lambda)$ / $\mathcal{F}.\text{In} \to \mathcal{F}.\text{Out}$
$\text{st}_{\text{RO}} \leftarrow\$ \text{RO}.\text{Init}(1^\lambda)$
$\text{pp}_1 \leftarrow\$ \mathcal{F}.\text{Setup}(1^\lambda)$
$(\text{Gst}_{\text{Sim}}, \text{pk}_0, \text{pp}_0) \leftarrow\$ \text{Sim}.\text{Init}(\text{pp}_1)$
$(\text{sk}, \text{pk}_1) \leftarrow\$ \mathcal{F}.\text{KeyGen}^{\text{RO}}_{\text{pp}_1}(1^\lambda)$
$\text{Gst}_S, \text{Gst}_C \leftarrow \perp, \perp$
$b' \leftarrow\$ \mathcal{A}^{\text{PreProc},\text{Eval},\text{BlindEval},\text{Prim}}(\text{pp}_b, \text{pk}_b)$
return b'

Oracle $\text{Eval}(t, x)$

$z_0 \leftarrow \text{RO}_{\text{Fn}}.\text{Eval}((t, x); \text{st}_{\text{Fn}})$
$z_1 \leftarrow \mathcal{F}.\text{Eval}^{\text{RO}}_{\text{pp}_1}(\text{sk}, t, x)$
return z_b

Oracle $\text{PreProc}()$

$q_p \leftarrow q_p + 1$
$\text{st}^0_C \leftarrow \text{Sim}.\text{PreProc}(\text{pp}_0; \text{Gst}_{\text{Sim}})$
$(\text{st}^1_C, \text{st}^1_S) \leftarrow \mathcal{F}.\text{PreProc}_{\text{pp}_1}(\text{sk})$
$\text{Gst}_S \leftarrow \text{Gst}_S \parallel \text{st}^1_S$
$\text{Gst}_C \leftarrow \text{Gst}_C \parallel \text{st}^b_C$
return st^b_C

Oracle $\text{LimitEval}(t, x)$

$q_{t,s} \leftarrow q_{t,s} + 1$
if $q_{t,s} \le q_t \wedge q_{t,s} \le \tau q_p$ **then**
  return $\text{Eval}(t, x)$
return $\perp$

Oracle $\text{BlindEval}(t, x, i)$

Parse $\text{Gst}_C = (\text{st}_{C,1} \parallel \ldots \parallel \text{st}_{C,q_p})$
if $i > q_p$ **then return** $\perp$
$\text{req} \leftarrow \mathcal{F}.\text{Request}^{\text{RO}}_{\text{pp}_b}(\text{pk}_b, t, x; \text{st}_{C,i})$
if $\text{req} = \perp$ **then return** $\perp$
if $\text{req} \neq \perp$ **then** $q_t \leftarrow q_t + 1$
$\text{rep}_0 \leftarrow\$ \text{Sim}.\text{BlindEval}^{\text{LimitEval}}(t, x, \text{req}; \text{Gst}_{\text{Sim}})$
$\text{rep}_1 \leftarrow\$ \mathcal{F}.\text{BlindEval}^{\text{RO}}_{\text{pp}_1}(\text{sk}, t, \text{req}; \text{Gst}_S)$
return rep_b

Oracle $\text{Prim}(x)$

$h_0 \leftarrow\$ \text{Sim}.\text{Eval}^{\text{LimitEval}}(x; \text{Gst}_{\text{Sim}})$
$h_1 \leftarrow\$ \text{RO}.\text{Eval}(x; \text{st}_{\text{RO}})$
return h_b

Figure 1: Pseudorandomness game against semi-honest clients adapted from [74].

Game $\text{POPRIV1}^{\mathcal{A},b}_{\mathcal{F},\text{RO}}(\lambda)$

$q_p \leftarrow 0$
$\text{Gst}_C, \text{Gst}_S \leftarrow \perp, \perp$
$\text{pp} \leftarrow\$ \mathcal{F}.\text{Setup}(1^\lambda)$
$(\text{pk}, \text{sk}) \leftarrow\$ \mathcal{F}.\text{KeyGen}^{\text{RO}}_{\text{pp}}(1^\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\text{PreProc},\text{Run},\text{RO}}(\text{pp}, \text{pk}, \text{sk})$
return b'

Oracle $\text{PreProc}()$

$q_p \leftarrow q_p + 1$
$\text{st}_C, \text{st}_S \leftarrow \mathcal{F}.\text{PreProc}(\text{sk})$
$\text{Gst}_C \leftarrow \text{Gst}_C \parallel \text{st}_C$
$\text{Gst}_S \leftarrow \text{Gst}_S \parallel \text{st}_S$
return st_S

Oracle $\text{Run}(t, x_0, x_1, i_0, i_1)$

if $i_0 > q_p \vee i_1 > q_p$ **then return** $\perp$
Parse $\text{Gst}_C = (\text{st}_{C,1} \parallel \ldots \parallel \text{st}_{C,q_p})$
Parse $\text{Gst}_S = (\text{st}_{S,1} \parallel \ldots \parallel \text{st}_{S,q_p})$
for $j \in \{0, 1\}$ **do**
  $\text{req}_j \leftarrow\$ \mathcal{F}.\text{Request}^{\text{RO}}_{\text{pp}}(\text{pk}, t, x_{j \oplus b}; \text{st}_{C,i_j})$
  if $\text{req}_j = \perp$ **then return** $\perp$
  $\text{rep}_j \leftarrow\$ \mathcal{F}.\text{BlindEval}^{\text{RO}}_{\text{pp}}(\text{sk}, t, \text{req}_j; \text{Gst}_S)$
  if $\text{rep}_j = \perp$ **then return** $\perp$
  $z_j \leftarrow\$ \mathcal{F}.\text{Finalize}^{\text{RO}}_{\text{pp}}(\text{rep}_j; \text{st}_C)$
  if $z_j = \perp$ **then return** $\perp$
$T_0 \leftarrow (\text{req}_0, \text{rep}_0, z_b)$
$T_1 \leftarrow (\text{req}_1, \text{rep}_1, z_{1-b})$
return (T_0, T_1)

Figure 2: Request privacy against semi-honest servers.

4 FORMAL CONSTRUCTION

We now formally present the Pool construction, beginning with a description of the preprocessing phase in Figure 3. The remaining algorithms are presented in Figure 4. Essentially, we have separated the construction described in the introduction into the usual OT preprocessing and online phases. As a result, we describe and prove Pool secure in an $\mathcal{F}_{\text{OT}}$-hybrid model using calls to idealized OT functionalities. In Section 6, we replace the OT functionalities with concrete OT constructions to obtain a concrete scheme.

Preprocessing. This phase consists of two separate Random OT runs. For intuition, we set $\tau = 1$ and ignore t. For the n 1-out-of-2 Random OTs (where sent strings are in $\mathcal{M} = \mathbb{Z}_q$), the client is the sender whereas in the 1-out-of-Δ Random OTs (where sent strings are in $\mathcal{M} = \mathbb{Z}_p$), the server is the sender. In essence, the server and client share random masks that allow them to compute the OPRF in the online phase. The only thing to add is that the bits used by the server (b_i) when receiving their portion of random masks are then shifted by their secret key bits, and these bits ($\bar{b}_i$) and are then sent to the client. The output of the preprocessing phase is captured in the respective states st_C and st_S.

Online phase. We denote the client input as x and the associated vector $\mathbf{a} := H(x)$ for a hash function $H : \{0,1\}^* \to \mathbb{Z}_q^n$. Ultimately, the client wishes to learn $PRF_{\text{sk}}(x) := \lceil \mathbf{a}^\top \text{sk} \rfloor_{q,p}$ where $\text{sk} = (\text{sk}_1, \ldots, \text{sk}_n) \in \{0,1\}^n$ is the server's secret key. To this end, the i-th 1-out-of-2 Random OT from preprocessing will be used by the server to learn *either* r_i *or* $a_i + r_i$ where r_i is a uniform element chosen over $\mathbb{Z}_q$. If $\text{sk}_i = 0$, the server knows r_i from preprocessing. Otherwise it can use its preprocessing output to learn $a_i + r_i$ by subtracting its received 1-out-of-2 OT string from the e_i component of the request message. In the case that the server learns $a_i + r_i$, the r_i terms acts as a one-time pad, information theoretically hiding a_i as r_i is hidden from the server during preprocessing. Additionally, the client sends $\bar{b}' \in \mathbb{Z}_\Delta$ to help the server craft its response. This explains the request algorithm.

For blind evaluation, the server adds together its learned strings to obtain $\tilde{a}_\Sigma = \mathbf{a}^\top \text{sk} + r_\Sigma$, where $r_\Sigma := \sum_{i=1}^n r_i$, where the blinding factor $\sum_{i=1}^n r_i$ is fully known to the client. Therefore, we would like the server to transfer $\lceil \tilde{a}_\Sigma - r_\Sigma \rfloor_{q,p}$ to the client without leaking any further information. A naive way to do this would be to use a 1-out-of-q OT with sender strings $\lceil \tilde{a}_\Sigma - i \rfloor_{q,p}$ for $i = 0, \ldots, q$. Then with knowledge of r_Σ the client could choose to learn the appropriate string. However, the sequence $(\lceil \tilde{a}_\Sigma - i \rfloor_{q,p})_{i=0}^{q-1}$ is simply a cyclic shift of $(0, \ldots, 0, 1, \ldots, 1, \ldots, \ldots, p-1, \ldots, p-1)$, where each value is repeated Δ times. In particular, any cyclic shift of this vector has the property that for any index i whose value is y_i, the entry at index $i + j\Delta$ takes the value $y_i + j \bmod p$. Therefore, we use this redundancy to use a 1-out-of-Δ OT in the blind evaluation algorithm, allowing the server to only send strings $(\lceil \tilde{a}_\Sigma - i \rfloor_{q,p})_{i=0}^{\Delta-1}$. Now, the client learns $\lceil \tilde{a}_\Sigma - (r_\Sigma \bmod \Delta) \rfloor_{q,p}$ from these strings and then corrects using the aforementioned structure to derive $\lceil \mathbf{a}^\top \text{sk} \rfloor_{q,p}$. More concretely, the server sends blinded versions of

PreProc(sk):

For a client C with input $\perp$, server S with input $\mathsf{sk} \in \{0,1\}^n$ and public parameters $\mathsf{pp} = (\tau, n, p, q, \lambda)$, perform the following:

(1) Sample unique session ID $\mathsf{uid} \in \{0,1\}^\lambda$

(2) Run $\mathcal{F}^{q,\tau,n}_{\binom{2}{1}\text{-rOT}}$ on input $(\text{Send}, \mathsf{C}, \mathsf{uid})$ from C and $(\text{Receive}, \mathsf{S}, \mathsf{uid})$ from S. As a result, the client obtains

$$S_\mathsf{C} := \{(r^j_{0,i}, r^j_{1,i}) \in \mathbb{Z}_q \times \mathbb{Z}_q\}_{(i,j)\in[n]\times[\tau]}$$

and the server obtains

$$R_\mathsf{S} := \{(b_i, r^j_{b_i,i}) \in \{0,1\} \times \mathbb{Z}_q\}_{(i,j)\in[n]\times[\tau]}.$$

(3) Run $\mathcal{F}^{p,\tau}_{\binom{\Delta}{1}\text{-rOT}}$ on input $(\text{Send}, \mathsf{S}, \mathsf{uid})$ from S and $(\text{Receive}, \mathsf{C}, \mathsf{uid})$ from C. As a result, S obtains $S_\mathsf{S} := \{(r'_{0,j}, \ldots, r'_{\Delta-1,j}) \in (\mathbb{Z}_p)^\Delta\}^\tau_{j=1}$ and C obtains

$$R_\mathsf{C} := \{(b'_j, r'_{b'_j,j}) \in [0, \Delta-1] \times \mathbb{Z}_p\}^\tau_{j=1}.$$

(4) Send $\bar{b}_i \leftarrow b_i \oplus \mathsf{sk}_i$ to C.

C obtains the state $\mathsf{st}_\mathsf{C} \leftarrow (S_\mathsf{C}, R_\mathsf{C}, \{\bar{b}_i\}^n_{i=1}, (\mathsf{uid}, 0))$.

S obtains the state $\mathsf{st}_\mathsf{S} \leftarrow (S_\mathsf{S}, R_\mathsf{S}, (\mathsf{uid}, 0))$.

Figure 3: Preprocessing phase of Pool.

$(\lceil \tilde{a}_\Sigma - i \rfloor_{q,p})^{\Delta-1}_{i=0}$ in such a way that the client can unblind only the appropriate index using the 1-out-of-Δ OT from preprocessing. The server can ensure this by using its random 1-out-of-Δ OT output and the value of $\bar{b}'$ in the client's request message.

Throughout, the algorithms ensure elements of st_C and st_S are not reused by maintaining and incrementing counters. This is crucial for security, and the request and blind evaluation algorithms check that the counter value is less than τ before running.

Achieving malicious security. As we mentioned in the introduction, one of the limitation of our approach is that we only realize security in the semi-honest model. We would have to use extra PQ commitments and zero-knowledge proofs to guarantee security against a malicious adversary. Concretely speaking, these commitments and proofs would attest to the fact that the preprocessing and online phases are performed jointly and correctly. At the very least, the online proof needs to attest to the correct computation of $a(x) = H(x)$ for some hash function H that is modeled as a random oracle. This comes at a considerable cost to the client.

In an attempt to alleviate this cost, one could potentially use the relatively simple tree-based functions from [13] instead of a random oracle H. A disadvantage of this is that the resulting LWR PRF is only proven secure according to an LWE problem with super-polynomial modulus. Heuristically, the same work [13] suggests that one may use their construction with small moduli (p, q) as long as $p \mid q$. This would enable much simpler zero-knowledge proofs in a malicious client setting somewhat resembling the client zero-knowledge proofs in previous lattice-based OPRF constructions [6, 50]. As a rough estimate, the LaBRADOR lattice-based proof system [22] would be expected to yield proof sizes of around 50KB. Investigating a variety of zero-knowledge proof systems offering trade-offs between proof size, prover time and verifier time is left as future work. We note however, that the required server zero-knowledge proofs are in comparison much simpler and therefore likely to be much more efficient than the client ones.

Aside from proving correctness of $a(x)$, Leap [55] has a similar format to ours, consisting of OT functionalities (plus extra rounds for computing OLE exchanges that we do not require), and utilizes preprocessing. Furthermore, the masking/unmasking routine is relatively similar. As a result, one could potentially adapt the security proof in the malicious model used in that work to obtain a malicious variant of our protocol, utilizing malicious variants of OT, and extractable commitments for binding the adversary to their input choices.

Given there are various applications of OPRFs in the semi-honest setting, our primary goal here is to build a simple, round-optimal, and highly efficient exchange for this functionality. Valuable future work would analyze the capability to adapt our design using the methods described above.

5 CORRECTNESS AND SECURITY PROOFS

In this section, we state theorems saying that Pool satisfies correctness (Theorem 5.1), pseudorandomness against semi-honest clients (Theorem 5.2) and request privacy against semi-honest servers (Theorem 5.3). The security proofs are only overviewed in the main body, but full proofs of all of the theorems below can be found in Appendix D. As a reminder, the $\mathcal{F}^\star_{\star\text{-rOT}}$ functionalities are formally defined in Appendix C.

Theorem 5.1. *Assuming $p \mid q$, the construction $\mathcal{F}$ in Figure 4 is τ-correct for any $\tau = \mathsf{poly}(\lambda)$.*

Theorem 5.2. *Take any $\tau = \mathsf{poly}(\lambda)$ and any (p, q) such that $p \mid q$. Further, assume that* RO *behaves like a random oracle and that the $\mathsf{LWR}^{\{0,1\}}_{n,p,q}$ assumption holds. Then the PPOPRF construction in Figure 4 is pseudorandom against semi-honest clients with τ-reusable preprocessing in the idealized $\left\{\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}, \mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}\right\}$-hybrid model.*

Proof. *(Overview)* We use a hybrid/game-hopping argument starting with the game $G_0 = \text{SH-PPOPRF}^{\mathcal{A},0}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$ and then making a sequence of indistinguishable changes to reach $G_4 = \text{SH-PPOPRF}^{\mathcal{A},1}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$. The simulator Sim and sequence of hybrids is summarized in Figure 6 of Appendix D.2. In this figure, Sim is described without any highlighting whereas changes between subsequent games are highlighted using different colors. The key points are that the Sim.PreProc algorithm uses the dummy key $\mathbf{0}^n$ during preprocessing and Sim.BlindEval sets all of the y_i's uniformly apart from the one that the client should be able to unblind. Calling the latter entry y_{i^*}, Sim uses its LimitEval oracle to set y_{i^*} consistently with outputs of $\mathsf{RO}_{\mathsf{Fn}}$. As a brief overview, the proof with respect to this simulator uses the following steps:

(1) G_1 replaces the $\mathsf{RO}_{\mathsf{Fn}}$ oracle with a LWR PRF (this step uses the LWR assumption).

(2) G_2 updates the preprocessing algorithm to use the PRF secret key (introduced in G_1) instead of the all-0 key (this step uses the fact that the preprocessing output, including $\bar{b}'$ appears uniform from the client-perspective),

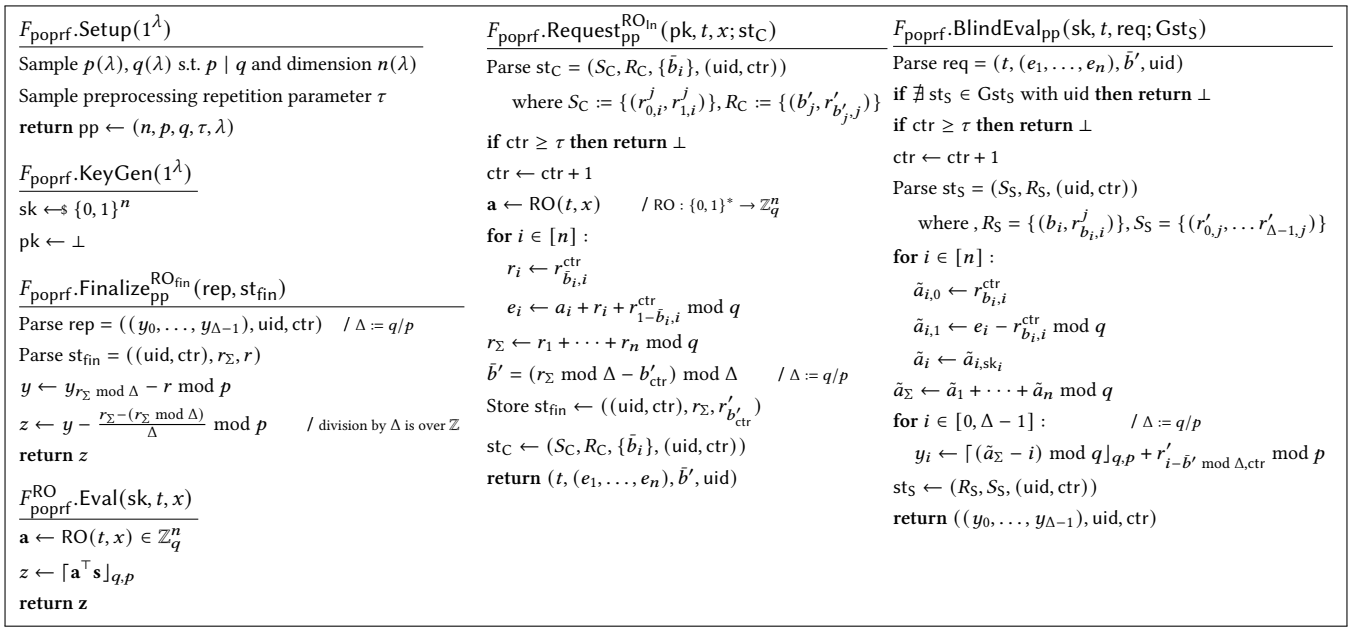


Figure 4: The online phase of Pool, with the OT-based preprocessing from Figure 3.

(3) G_3 finally updates the Sim.BlindEval response to be correct in all entries with respect to the LWR secret key (using uniformity of the blinding values on all response-entries apart from the one accessed by the client).

To complete the proof, we show that G_3 is identical to G_4 from a semi-honest client's perspective. See Appendix D.2 for more details. □

Theorem 5.3. *Assume that $p \mid q$. The PPOPRF construction in Figure 4 is pseudorandom against semi-honest servers in the idealized $\left\{\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}, \mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}\right\}$-hybrid model.*

Proof. *(Overview)* We use a hybrid/game-hopping argument, starting with the game $G_{0,b} = \text{POPRIV1}^{\mathcal{A},b}_{\mathcal{F},\text{Sim},\text{RO}}(\lambda)$ and ending with $G_{3,b}$, where the value of b is no longer used. Thus, $G_{3,0}$ and $G_{3,1}$ are indistinguishable and by extension, so are $G_{0,0}$ and $G_{0,1}$ as required. As a brief overview, we make the following sequence of indistinguishable changes to Run.

(1) We first replace (z_b, z_{1-b}) in Run's output by $(\lceil \text{RO}(t, x_0)^\top \cdot \text{sk} \rfloor_{q,p}, \lceil \text{RO}(t, x_1)^\top \cdot \text{sk} \rfloor_{q,p})$, via PPOPRF correctness.
(2) We next replace $\bar{b}'$ in calls to $\mathcal{F}$.Request by uniform values. This is possible since b'_{ctr} is hidden from the semi-honest server $\mathcal{A}$ via an OT correlation.
(3) We finally replace request elements $(e_1, \ldots, e_n)$ by uniform values. This step is done by showing that each e_i is a sum of terms, exactly one of which is uniformly distributed from the perspective of the $\mathcal{A}$ via an OT correlation.

At this point, the request messages and finalized outputs contained in Run's output (T_0, T_1) are independent of the challenge bit b as desired. The full proof is included in Appendix D.3. □

$\binom{2}{1}$-OT extension	(n, p, q)	# evals	τ	Communication Client	Communication Server	Computation Client	Computation Server
IKNP [56]	$(415, 2^4, 2^8)$	2^0	2^5	756 kB	768 kB	62 ms	62 ms
		2^{13}	2^{18}	17.5 MB	109.5 MB	1.1 s	575 s
	$(482, 2^8, 2^{12})$	2^0	2^4	756 kB	770 kB	79 ms	79 ms
		2^{13}	2^{17}	9.1 MB	96 MB	1.1 s	634 ms
Silent OT [29]	$(415, 2^4, 2^8)$	2^0	2^5	792 kB	792 kB	154 ms	154 ms
		2^{13}	2^{18}	815 kB	887 kB	8.6 s	8.3 s
	$(482, 2^8, 2^{12})$	2^0	2^4	792 kB	792 kB	176 ms	176 ms
		2^{13}	2^{17}	813 kB	879 kB	7 s	7 s

Table 2: Preprocessing performance for "# evals" OPRF evaluations yielding 128 output bits, targeting 128 bits of security.

6 EXPERIMENTAL ANALYSIS

We now compare performance of Pool with previous work, via an open-source implementation[8]. Implementation details and parameter choices are discussed in Section 6.1. In Section 6.2, we compare efficiency with the state-of-the-art in PQ OPRF constructions.

6.1 Implementation Details

Our open-source reference implementation of Pool is written in C++ and relies on libOTe [71] for the OT primitives. The libOTe library is used as is and parametrized with the statistical security parameter set at 40 to match prior work [55, 66]. The implementation is comprised of two distinct parts: the preprocessing phase, which incurs the largest communication and computation costs, and the online phase, which has relatively small communication costs and marginal computation costs.

[8] https://github.com/tremblaythibaultl/ot-pq-oprf

λ	(n, p, q)	Comm.	Client Time	Server Time
100	$(296, 2^4, 2^8)$	9.8 kB	$\delta \cdot 41\,\mu s$	$\delta \cdot 23\,\mu s$
100	$(370, 2^8, 2^{12})$	9.2 kB	$\delta \cdot 50\,\mu s$	$\delta \cdot 28\,\mu s$
128	$(415, 2^4, 2^8)$	13.6 kB	$\delta \cdot 57\,\mu s$	$\delta \cdot 32\,\mu s$
128	$(482, 2^8, 2^{12})$	11.9 kB	$\delta \cdot 64\,\mu s$	$\delta \cdot 36\,\mu s$

Table 3: Single-threaded online performance of Pool with 128 bits of output and λ bits of security, for associated parameters. Note that $\delta = 128/\log(p)$, to indicate that each protocol run outputs only $\log(p)$ bits, and thus needs to be repeated.

All measures presented in this paper were obtained using an AWS `t3.xlarge` EC2 instance equipped with 4 vCPUs with AVX2 enabled and 16 GB memory running ubuntu 24.04. Experiments are run 100 times, and measures are taken as the median of the results.

Preprocessing. The preprocessing phase (see Figure 3) may also be considered as two separate phases, each containing a Random OT interaction. The first phase is a set of $\binom{2}{1}$-OTs instantiated by the Kyber version of the Masny-Rindal [66] protocol extended with IKNP [56] or Silent OT [29]. The second phase consists of a set of $\binom{N}{1}$ OTs which can be instantiated by KKRT [61] or Naor-Pinkas [69], again extending the Kyber version of [66]. Experimentally, we confirm the same findings noticed in [55], that KKRT is not particularly efficient for low values of N. Since we consider $N = \Delta$, where Δ is always 16 in our experiments, we benchmark this $\binom{\Delta}{1}$-OT using the [69] approach. However, our end-to-end implementation includes a KKRT-based variant. As before, τ denotes the number of times the online OPRF protocol can be run before preprocessing must be run again. We further note that in Section 6.2, we report preprocessing using additional optimizations for reducing the number of OTs, but our end-to-end implementation of preprocessing uses a less optimized exchange for simplicity's sake. In Appendix B, we experimentally compare alternative methods for implementing Silent OT, depending on whether we would like to optimize communication (n OTs variant) or runtimes ($n \cdot \kappa$ OTs). In our preprocessing comparison in Tables 2 and 4, we are more interested in lowering communication, so we focus only on the results in the n OTs case.

Online phase. During the online phase, the client computes the Request and Finalize algorithms while the server computes the BlindEval algorithm (see Figure 4). Table 3 gives online communication and computation measures for both the client and the server with respect to each of the proposed parameter sets.

Parameter choices. We give parameters and associated performance figures for both 100-bit and 128-bit security parameters, as is common in prior work. Unless otherwise specified, we aim for a PRF output length of $L = 128$ bits. As presented, the output length is $\log_2(p)$, so a naive way to extend the output length would be to set $p = 2^{128}$. Instead, we may extend the LWR-based PRF to compute $F'_{\mathbf{s}}(x) = \lceil H(x) \cdot \mathbf{s} \rfloor_{q,p}$ where $H : \{0,1\}^* \to \mathbb{Z}_q^{\lceil L/\log_2(p) \rceil \times n}$. It is trivial to extend Pool to handle this case by running $\delta := \lceil L/\log_2(p) \rceil$ evaluations of the single-row construction in parallel (one per row of $F'_{\mathbf{s}}$). Assuming L and $\log_2(p)$ are powers-of-2, the approximate theoretical cost in the online phase then becomes $256 + (\lceil n \log(q) + \log(\Delta) \rceil + \lceil \Delta \cdot \log(p) \rceil) \cdot \delta$ bits. This assumes a 256-bit uid. After some experimentation with the Lattice Estimator [7], we arrive at the suggested parameter sets given in Table 3.

Comparison. In Section 6.2 we compare Pool with the recent Leap construction [55] from Eurocrypt'25, at the 128-bit security level. There are a few reasons for this. Firstly, Leap is the most efficient OPRF candidate that has an open-source implementation, which means that we can run their benchmarking code on the same hardware as ours for a fair performance comparison. Other practical approaches, such as [2, 49, 75], do not provide such code, despite giving benchmarks. We also note that recent cryptanalysis has shown that alternating moduli constructions [2, 49] may require more conservative parameter choices due to attacks on the underlying weak PRFs, that would decrease the quoted efficiency [9]. Furthermore, these constructions profit from making evaluations in a batched amortized setting, where runtimes and bandwidth are divided by the number of evaluations requested. As shown in [75], costs rise fairly dramatically (beyond the realms of practicality) in a single-evaluation setting. Since many OPRF applications require only low numbers of evaluations (e.g. [45, 59]), we include performance assessments in a single-evaluation setting for fairness.

6.2 Performance Analysis

We detail the preprocessing and online performance of Pool across different parameter settings (see Tables 2 and 3). We then pick the most efficient setting of Pool for comparing with Leap across both phases (see Tables 4 and 5). For the comparison, we assume 64 bits of PRF output, since this is the output length considered by Leap.

Preprocessing. In Table 2, we run preprocessing using both IKNP-based [56] and Silent OT-based [29] OT extension at the 128-bit security level.[9] In the single evaluation setting, the two methods yield similar communication costs of around 750 kB-800 kB per party whereas Silent OT takes around two to three times longer. However, in the batched setting with 2^{13} evaluations, the Silent OT method yields considerably worse computational costs (between 7 and 9 seconds), but the communication costs are almost unchanged from the single evaluation setting. Contrarily, the batched communication cost using IKNP requires orders of magnitude more communication, while the computational costs remain low at less than 180 ms. We note that the benchmarks provided for Silent OT are obtained from an implementation that prioritizes optimizing communication. In particular, the $\binom{2}{1}$-OTs with output length $\tau \cdot \log(q)$ are benchmarked by producing 128-bit sender strings in libOTe and expanding to $\tau \cdot \log(q)$ bits using a PRNG. In Appendix B, we discuss how modifications to this method can be used to reduce runtimes at the cost of increasing the server's communication.

Online phase. Table 3 shows the online costs for the protocol for λ-bit security levels (where $\lambda \in \{100, 128\}$). Overall, total communication varies between 9.2 kB and 13.6 kB, depending on the parametrization. Single-threaded client execution times for achieving 128-bit PRF outputs (via δ repetitions) range from $16 \cdot 52\,\mu s = 832\,\mu s$ to $32 \cdot 50\,\mu s = 1.6$ ms, depending on the value of p (and thus δ). For the server, we have execution times ranging from $16 \cdot 32\,\mu s = 512\,\mu s$ to $32 \cdot 28\,\mu s = 896\,\mu s$. We note that the δ

[9] This upper-bounds the performance of 100-bit security levels.

repetitions can be parallelized, improving the reported times by factors of up to 16 or 32 depending on whether $p = 2^8$ or 2^4.

Comparison with Leap. In Tables 4 and 5 we compare the preprocessing and online performance, respectively, of Pool with that of Leap, at the 128-bit security level.

Starting with the case of preprocessing using IKNP OT extension, we see that the total communication overhead of Leap is around half the cost of Pool in the single evaluation setting, settling at ∼750 kB, instead of ∼1.5 MB for Pool. However, in the multiple evaluation setting, Leap incurs a very large communication cost of ∼271 MB on the client-side. In contrast, Pool incurs a much lower total preprocessing cost of ∼53 MB in this setting, with 49.1 MB being included in the server-side costs. In terms of runtimes, both OPRFs are comparable with Pool performing significantly better in the multiple evaluation setting.

In the Silent OT case, the communication cost of both protocols is dramatically lowered. The base communication cost of performing preprocessing for a single OPRF evaluation in Pool is around double the cost of Leap, but still less than 1 MB for either client or server. The communication cost of running Silent OT-based preprocessing for multiple evaluations is almost equivalent to the single evaluation costs in both protocols, due to the very gradual growth of communication using this form of OT extension. In terms of runtimes, the costs of Pool are significantly lower than the cost of Leap in the multiple evaluation setting, taking only around 3 s of both client and server time, while the costs of Leap are over 2× more expensive. The single evaluation runtimes are almost equivalent.

In the case of online costs in Table 5, we firstly notice that the overall communication cost in Pool is around 25-30% of the communication cost in Leap. In terms of computation time, the number of repetitions of Pool determines the amount of time. In a purely single-threaded execution (which is the fairest way of comparing with Leap's benchmarks), we see that the client costs are either 912 µs or 512 µs, depending on the parameter set. In the server's case, this is either 512 µs or 288 µs, respectively. Therefore, server runtimes in Pool can be up to an order of magnitude more efficient than in Leap, while client costs can be up to 6× smaller. Recall further that Leap has an online phase consisting of at least 3 rounds, and thus the latency of the additional message could worsen the runtimes of Leap further in a real-world experiment (especially given the additional bandwidth overhead).

Discussion. Overall, we see that the online phase of Pool is more efficient than Leap in all cost metrics. Additionally, the number of online rounds (in the preprocessing model) in Leap is three whereas Pool is round optimal, meaning that additional network latency could drive the approaches even further apart. From a usability perspective, we also note that the server must initiate the connection in Leap. This is a burden for real-world applications, where clients must typically initiate the connection (at the HTTP layer, for example). Therefore, a real-world implementation of Leap could in fact require four separate messages (including some sort of client hello message). For preprocessing, we observe that Pool has overheads somewhat inline with Leap; with generally higher communication but lower runtimes. We highlight that the preprocessing overhead in either protocol is amortizable and input-independent, meaning it can be run in the background before the "online" exchange begins.

	$\binom{2}{1}$-OT Ext.	# evals	Communication		Computation	
			Client	Server	Client	Server
[55]	IKNP	2^0	426 kB	328 kB	26 ms	27 ms
		2^{13}	271 MB	328 kB	790 ms	881 ms
	Silent OT	2^0	445.9 kB	375.7 kB	27 ms	27 ms
		2^{13}	472 kB	479 kB	6.4 s	11.1 s
Pool	IKNP	2^0	756 kB	762 kB	52 ms	52 ms
		2^{13}	4.9 MB	48.1 MB	444 ms	219 ms
	Silent OT (n OTs)	2^0	792 kB	792 kB	64 ms	63 ms
		2^{13}	811 kB	871 kB	3 s	3 s

Table 4: Communication and computation cost of preprocessing algorithms provided by Leap [55] and this work (Pool) with $(n, p, q) = (482, 2^8, 2^{12})$. The parameter τ is set to match the output size of [55], i.e. $\tau = \#\,\text{evals} \cdot 2^3$, for 64 output bits.

		Comm.	Client time	Server Time
Leap [55]		23 kB	3 ms	4 ms
Pool	$(n, p, q) = (415, 2^4, 2^8)$	6.8 kB	$16 \cdot 57$ µs	$16 \cdot 32$ µs
	$(n, p, q) = (482, 2^8, 2^{12})$	5.9 kB	$8 \cdot 64$ µs	$8 \cdot 36$ µs

Table 5: Communication and computation cost of online OPRF evaluation from Leap [55] and Pool. Performance and sizes are with respect to 64 bits of PRF output, to match Leap.

7 CONCLUSION

We proposed Pool, a round-optimal OPRF from a standard PQ assumption (binary LWR). We showed (via a provided implementation) that Pool is competitive or more efficient than existing OPRFs, that are based on less well-studied, yet plausible, post-quantum assumptions. Furthermore, it is orders of magnitude faster than OPRFs based on standard PQ assumptions. Viable applications for Pool include PSOs and MPC use-cases, and its basis on standard assumptions make it a strong candidate for Internet standardization use-cases [44], where strong security guarantees are paramount.

8 ACKNOWLEDGEMENTS

Alex Davidson's work was supported by FCT and the CMU Portugal program through project "Fully-Homomorphic Encryption from Post-Quantum Code-Based Assumptions", ref. 2024.12595.CMU, and the LASIGE Research Unit, ref. UID/00408/2025 - LASIGE. The authors would like to thank Fernando Virdia for providing helpful discussions.

REFERENCES

[1] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. 2014. Candidate weak pseudorandom functions in $AC^0 \circ MOD_2$. In *ITCS 2014*, Moni Naor (Ed.). ACM, 251–260. https://doi.org/10.1145/2554797.2554821

[2] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. 2024. Improved Alternating-Moduli PRFs and Post-quantum Signatures. In *CRYPTO 2024, Part VIII (LNCS, Vol. 14927)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 274–308. https://doi.org/10.1007/978-3-031-68397-8_9

[3] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. 2018. Estimate All the LWE, NTRU Schemes!, See [35], 351–367. https://doi.org/10.1007/978-3-319-98113-0_19

[4] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. 2024. Crypto Dark Matter on the Torus - Oblivious PRFs from Shallow PRFs and TFHE. In *EUROCRYPT 2024, Part VI (LNCS, Vol. 14656)*, Marc Joye and Gregor Leander (Eds.). Springer, Cham, 447–476. https://doi.org/10.1007/978-3-031-58751-1_16

[5] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. 2021. Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices, See [52], 261–289. https://doi.org/10.1007/978-3-030-75248-4_10

[6] Martin R. Albrecht and Kamil Doruk Gür. 2024. Verifiable Oblivious Pseudorandom Functions from Lattices: Practical-Ish and Thresholdisable. In *ASIACRYPT 2024, Part IV (LNCS, Vol. 15487)*, Kai-Min Chung and Yu Sasaki (Eds.). Springer, Singapore, 205–237. https://doi.org/10.1007/978-981-96-0894-2_7

[7] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.* 9, 3 (2015), 169–203. http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml

[8] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. 2013. Learning with Rounding, Revisited - New Reduction, Properties and Applications, See [32], 57–74. https://doi.org/10.1007/978-3-642-40041-4_4

[9] Irati Manterola Ayala and Håvard Raddum. 2024. Zeroed Out: Cryptanalysis of Weak PRFs in Alternating Moduli. *IACR Cryptol. ePrint Arch.* (2024), 2055. https://eprint.iacr.org/2024/2055

[10] Hayo Baan, Sauvik Bhattacharya, Scott R. Fluhrer, Óscar García-Morchón, Thijs Laarhoven, Ronald Rietman, Markku-Juhani O. Saarinen, Ludo Tolhuizen, and Zhenfei Zhang. 2019. Round5: Compact and Fast Post-quantum Public-Key Encryption. In *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11505)*, Jintai Ding and Rainer Steinwandt (Eds.). Springer, 83–102. https://doi.org/10.1007/978-3-030-25510-7_5

[11] Hayo Baan, Sauvik Bhattacharya, Óscar García-Morchón, Ronald Rietman, Ludo Tolhuizen, Jose Luis Torre-Arce, and Zhenfei Zhang. 2017. Round2: KEM and PKE based on GLWR. *IACR Cryptol. ePrint Arch.* (2017), 1183. http://eprint.iacr.org/2017/1183

[12] Abhishek Banerjee, Hai Brenner, Gaëtan Leurent, Chris Peikert, and Alon Rosen. 2015. SPRING: Fast Pseudorandom Functions from Rounded Ring Products. In *FSE 2014 (LNCS, Vol. 8540)*, Carlos Cid and Christian Rechberger (Eds.). Springer, Berlin, Heidelberg, 38–57. https://doi.org/10.1007/978-3-662-46706-0_3

[13] Abhishek Banerjee and Chris Peikert. 2014. New and Improved Key-Homomorphic Pseudorandom Functions. In *CRYPTO 2014, Part I (LNCS, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Berlin, Heidelberg, 353–370. https://doi.org/10.1007/978-3-662-44371-2_20

[14] Abhishek Banerjee, Chris Peikert, and Alon Rosen. 2012. Pseudorandom Functions and Lattices. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Berlin, Heidelberg, 719–737. https://doi.org/10.1007/978-3-642-29011-4_42

[15] Andrea Basso. 2022. Poster: A Post-Quantum Oblivious PRF from Isogenies, See [76], 3327–3329. https://doi.org/10.1145/3548606.3563542

[16] Andrea Basso. 2024. POKE: A Framework for Efficient PKEs, Split KEMs, and OPRFs from Higher-dimensional Isogenies. Cryptology ePrint Archive, Report 2024/624. https://eprint.iacr.org/2024/624

[17] Andrea Basso. 2024. A Post-Quantum Round-Optimal Oblivious PRF from Isogenies. In *SAC 2023 (LNCS, Vol. 14201)*, Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen (Eds.). Springer, Cham, 147–168. https://doi.org/10.1007/978-3-031-53368-6_8

[18] Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. 2021. Cryptanalysis of an Oblivious PRF from Supersingular Isogenies. In *ASIACRYPT 2021, Part I (LNCS, Vol. 13090)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 160–184. https://doi.org/10.1007/978-3-030-92062-3_6

[19] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO'95 (LNCS, Vol. 963)*, Don Coppersmith (Ed.). Springer, Berlin, Heidelberg, 97–109. https://doi.org/10.1007/3-540-44750-4_8

[20] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. 2017. NTRU Prime: Reducing Attack Surface at Low Cost. In *SAC 2017 (LNCS, Vol. 10719)*, Carlisle Adams and Jan Camenisch (Eds.). Springer, Cham, 235–260. https://doi.org/10.1007/978-3-319-72565-9_12

[21] Ward Beullens, Lucas Dodgson, Sebastian Faller, and Julia Hesse. 2025. The 2Hash OPRF Framework and Efficient Post-Quantum Instantiations. Springer-Verlag.

[22] Ward Beullens and Gregor Seiler. 2023. LaBRADOR: Compact Proofs for R1CS from Module-SIS, See [53], 518–548. https://doi.org/10.1007/978-3-031-38554-4_17

[23] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. 2016. On the Hardness of Learning with Rounding over Small Modulus. In *TCC 2016-A, Part I (LNCS, Vol. 9562)*, Eyal Kushilevitz and Tal Malkin (Eds.). Springer, Berlin, Heidelberg, 209–224. https://doi.org/10.1007/978-3-662-49096-9_9

[24] Andrej Bogdanov and Alon Rosen. 2017. Pseudorandom Functions: Three Decades Later. Cryptology ePrint Archive, Report 2017/652. https://eprint.iacr.org/2017/652

[25] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. 2018. Exploring Crypto Dark Matter: New Simple PRF Candidates and Their Applications. In *TCC 2018, Part II (LNCS, Vol. 11240)*, Amos Beimel and Stefan Dziembowski (Eds.). Springer, Cham, 699–729. https://doi.org/10.1007/978-3-030-03810-6_25

[26] Dan Boneh, Dmitry Kogan, and Katharine Woo. 2020. Oblivious Pseudorandom Functions from Isogenies. In *ASIACRYPT 2020, Part II (LNCS, Vol. 12492)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, Cham, 520–550. https://doi.org/10.1007/978-3-030-64834-3_18

[27] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. 2013. Key Homomorphic PRFs and Their Applications, See [32], 410–428. https://doi.org/10.1007/978-3-642-40041-4_23

[28] Daniel Bourdrez, Hugo Krawczyk, Kevin Lewi, and Christopher A. Wood. 2021. *The OPAQUE Asymmetric PAKE Protocol.* Internet-Draft draft-irtf-cfrg-opaque-07. IETF Secretariat. https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-07.txt https://www.ietf.org/archive/id/draft-irtf-cfrg-opaque-07.txt.

[29] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Cham, 489–518. https://doi.org/10.1007/978-3-030-26954-8_16

[30] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2021. Low-Complexity Weak Pseudorandom Functions in AC0[MOD2], See [65], 487–516. https://doi.org/10.1007/978-3-030-84259-8_17

[31] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, 136–145. https://doi.org/10.1109/SFCS.2001.959888

[32] Ran Canetti and Juan A. Garay (Eds.). 2013. *CRYPTO 2013, Part I*. LNCS, Vol. 8042. Springer, Berlin, Heidelberg.

[33] Ran Canetti, Pratik Sarkar, and Xiao Wang. 2020. Blazing Fast OT for Three-Round UC OT Extension. In *PKC 2020, Part II (LNCS, Vol. 12111)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer, Cham, 299–327. https://doi.org/10.1007/978-3-030-45388-6_11

[34] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. 2022. SoK: Oblivious Pseudorandom Functions. In *2022 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, 625–646. https://doi.org/10.1109/EuroSP53844.2022.00045

[35] Dario Catalano and Roberto De Prisco (Eds.). 2018. *SCN 18*. LNCS, Vol. 11035. Springer, Cham.

[36] Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). 2019. *ACM CCS 2019*. ACM Press.

[37] Sofía Celi, Alex Davidson, Steven Valdez, and Christopher A. Wood. 2024. Privacy Pass Issuance Protocols. *RFC* 9578 (2024), 1–36. https://doi.org/10.17487/RFC9578

[38] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. 2021. Adventures in Crypto Dark Matter: Attacks and Fixes for Weak Pseudorandom Functions, See [52], 739–760. https://doi.org/10.1007/978-3-030-75248-4_26

[39] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. 2018. Lizard: Cut Off the Tail! A Practical Post-quantum Public-Key Encryption from LWE and LWR, See [35], 160–177. https://doi.org/10.1007/978-3-319-98113-0_9

[40] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Part I (LNCS, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer, Berlin, Heidelberg, 3–33. https://doi.org/10.1007/978-3-662-53887-6_1

[41] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (Jan. 2020), 34–91. https://doi.org/10.1007/s00145-019-09319-x

[42] Ivan Damgård. 1990. On the Randomness of Legendre and Jacobi Sequences. In *CRYPTO'88 (LNCS, Vol. 403)*, Shafi Goldwasser (Ed.). Springer, New York, 163–172. https://doi.org/10.1007/0-387-34799-2_13

[43] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 2018. Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM. In *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10831)*, Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi (Eds.). Springer, 282–305. https://doi.org/10.1007/978-3-319-89339-6_16

[44] Alex Davidson, Armando Faz-Hernández, Nick Sullivan, and Christopher A. Wood. 2023. Oblivious Pseudorandom Functions (OPRFs) Using Prime-Order Groups. *RFC* 9497 (2023), 1–61. https://doi.org/10.17487/RFC9497

[45] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PoPETs* 2018, 3 (July 2018), 164–180. https://doi.org/10.1515/popets-2018-0026

[46] Alex Davidson, Peter Snyder, E. B. Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. 2022. STAR: Secret Sharing for Private Threshold Aggregation Reporting, See [76], 697–710. https://doi.org/10.1145/3548606.3560631

[47] Cyprien Delpech de Saint Guilhem and Robi Pedersen. 2024. New Proof Systems and an OPRF from CSIDH. In *PKC 2024, Part II (LNCS, Vol. 14603)*, Qiang Tang and Vanessa Teague (Eds.). Springer, Cham, 217–251. https://doi.org/10.1007/978-3-031-57725-3_8

[48] Amit Deo, Marc Joye, Benoit Libert, Benjamin R. Curtis, and Mayeul de Bellabre. 2024. Homomorphic Evaluation of LWR-based PRFs and Application to Transciphering. Cryptology ePrint Archive, Report 2024/665. https://eprint.iacr.org/2024/665

[49] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. 2021. MPC-Friendly Symmetric Cryptography from Alternating Moduli: Candidates, Protocols, and Applications, See [65], 517–547. https://doi.org/10.1007/978-3-030-84259-8_18

[50] Muhammed F. Esgin, Ron Steinfeld, Erkan Tairi, and Jie Xu. 2024. LeOPaRd: Towards Practical Post-Quantum Oblivious PRFs via Interactive Lattice Problems. Cryptology ePrint Archive, Paper 2024/1615. https://eprint.iacr.org/2024/1615

[51] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004 (LNCS, Vol. 3027)*, Christian Cachin and Jan Camenisch (Eds.). Springer, Berlin, Heidelberg, 1–19. https://doi.org/10.1007/978-3-540-24676-3_1

[52] Juan Garay (Ed.). 2021. *PKC 2021, Part II*. LNCS, Vol. 12711. Springer, Cham.

[53] Helena Handschuh and Anna Lysyanskaya (Eds.). 2023. *CRYPTO 2023, Part V*. LNCS, Vol. 14085. Springer, Cham.

[54] Lena Heimberger, Tobias Hennerbichler, Fredrik Meisingseth, Sebastian Ramacher, and Christian Rechberger. 2024. OPRFs from Isogenies: Designs and Analysis. In *ASIACCS 24*, Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas (Eds.). ACM Press. https://doi.org/10.1145/3634737.3645010

[55] Lena Heimberger, Daniel Kales, Riccardo Lolato, Omid Mir, Sebastian Ramacher, and Christian Rechberger. 2025. LEAP: A Fast, Lattice-based OPRF with Application to Private Set Intersection. Springer-Verlag.

[56] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO 2003 (LNCS, Vol. 2729)*, Dan Boneh (Ed.). Springer, Berlin, Heidelberg, 145–161. https://doi.org/10.1007/978-3-540-45146-4_9

[57] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. 2014. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, Berlin, Heidelberg, 233–253. https://doi.org/10.1007/978-3-662-45608-8_13

[58] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. 2016. 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*. IEEE Computer Society Press, 276–291. https://doi.org/10.1109/EuroSP.2016.30

[59] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks. In *EUROCRYPT 2018, Part III (LNCS, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Cham, 456–486. https://doi.org/10.1007/978-3-319-78372-7_15

[60] Stanislaw Jarecki and Xiaomin Liu. 2009. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC 2009 (LNCS, Vol. 5444)*, Omer Reingold (Ed.). Springer, Berlin, Heidelberg, 577–594. https://doi.org/10.1007/978-3-642-00457-5_34

[61] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 818–829. https://doi.org/10.1145/2976749.2978381

[62] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. 2019. Scalable Private Set Union from Symmetric-Key Techniques. In *ASIACRYPT 2019, Part II (LNCS, Vol. 11922)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, Cham, 636–666. https://doi.org/10.1007/978-3-030-34621-8_23

[63] Hanjun Li, Sela Navot, and Stefano Tessaro. 2024. POPSTAR: Lightweight Threshold Reporting with Reduced Leakage. In *USENIX Security 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association.

[64] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. 2019. Protocols for Checking Compromised Credentials, See [36], 1387–1403. https://doi.org/10.1145/3319535.3354229

[65] Tal Malkin and Chris Peikert (Eds.). 2021. *CRYPTO 2021, Part IV*. LNCS, Vol. 12828. Springer, Cham, Virtual Event.

[66] Daniel Masny and Peter Rindal. 2019. Endemic Oblivious Transfer, See [36], 309–326. https://doi.org/10.1145/3319535.3354210

[67] Thibault Meunier. 2023. Evolution of Privacy Pass at Cloudflare. (2023). USENIX PEPR.

[68] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin M. Case. 2024. Delegated Private Matching For Compute. *PoPETs* 2024, 2 (April 2024), 49–72. https://doi.org/10.56553/popets-2024-0040

[69] Moni Naor and Benny Pinkas. 1999. Oblivious Transfer and Polynomial Evaluation. In *31st ACM STOC*. ACM Press, 245–254. https://doi.org/10.1145/301250.301312

[70] Parker Newton and Silas Richelson. 2023. A Lower Bound for Proving Hardness of Learning with Rounding with Polynomial Modulus, See [53], 805–835. https://doi.org/10.1007/978-3-031-38554-4_26

[71] Lance Roy Peter Rindal. 2025. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[72] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS: Fast, Malicious Private Set Intersection. In *EUROCRYPT 2020, Part II (LNCS, Vol. 12106)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Cham, 739–767. https://doi.org/10.1007/978-3-030-45724-2_25

[73] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. https://doi.org/10.1145/1060590.1060603

[74] Nirvan Tyagi, Sofia Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. 2022. A Fast and Simple Partially Oblivious PRF, with Applications. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Cham, 674–705. https://doi.org/10.1007/978-3-031-07085-3_23

[75] Yibin Yang, Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, and Tal Rabin. 2024. Gold OPRF: Post-Quantum Oblivious Power Residue PRF. *IACR Cryptol. ePrint Arch.* (2024), 1955. https://eprint.iacr.org/2024/1955

[76] Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). 2022. *ACM CCS 2022*. ACM Press.

[77] Zama. 2022. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data. https://github.com/zama-ai/tfhe-rs.

A COMPARISON WITH MALICIOUS OPRFS

In Table 1, we compared with OPRF protocols that only target semi-honest security by design. However, malicious OPRFs can be assumed to be semi-honest, if we simply remove non-interactive mechanisms for committing to and proving certain actions, that ensure the correct functioning of the protocol under arbitrary adversarial attacks. Therefore, to avoid missing comparisons with schemes that may be efficient in the semi-honest paradigm, we include Table 6. This table documents the costs of such protocols after removing zero-knowledge proofs and other extraneous material that is necessary for achieving malicious security, and compares performance with Pool. We note that the state-of-the-art, prior to Pool, for lattice-based OPRFs in the semi-honest paradigm remains LEAP [55].

B SILENT OT PERFORMANCE TRADE-OFFS

In Table 2, we addressed the issue that libOTe only allows sender strings with a maximum length of 128 bits whereas the first phase requires n independent $\binom{2}{1}$-OTs with sender strings of length $\tau \cdot \log(q)$. There are two ways to overcome this. The first is to expand libOTe's 128-bit outputs using a PRNG. We refer to this strategy as "Silent OT (n OTs)" in Table 7. This method is also used in Tables 2 and 4. A second method is to replace n OTs with sender strings of length $\tau \cdot \log(q)$ by $n \cdot \kappa$ OTs of sender length $\tau \cdot \log(q)/\kappa$. For this latter method, one can set the parameter κ so that $\tau \cdot \log(q)/\kappa < 128$. To do so, we choose $\kappa = \lceil \tau \cdot \log(q)/128 \rceil$. In order to implement this second method, the preprocessing phase must send $n \cdot \kappa$ correction bits in $\{\bar{b}_i\}_{i=0}^{n\kappa-1}$ which adds to the communication costs of preprocessing. We refer to this second strategy as "Silent OT ($n \cdot \kappa$ OTs)" in Table 7.

The "Silent OT ($n \cdot \kappa$ OTs)"-method is more bandwidth-intensive, but reduces computation time as no PRNG expansion takes place. This gives additional flexibility for implementing Pool from Silent OT, depending on which performance measure is important for the application. In Table 2, we are more interested in optimizing communication to be as competitive with Leap as possible, and so we only quote the numbers from the n OTs variant.

Construction	Assumption	Online Rounds	Preprocessing	Online comms.
PQ 2HashDH [21]	Legendre PRF	9	✗	590KB
PQ 2HashDH [21]	Legendre PRF	2*	✓	198KB
[54]	Isogenies $\mathbb{F}_p$ + lattices	2	✗	21KB
[54]	Isogenies $\mathbb{F}_p$ + lattices	4	✗	35KB
[47]	Isogenies $\mathbb{F}_p$	2	✗**	16.4KB
LeOPaRd [50]	iMLWE-RU, MLWE, MSIS	2	✗	84.4KB / 29KB† ***
[4]	FHE, Alternating Moduli [25]	2	✗	4.1KB / 8.2KB****
[6]	RLWE, SIS	2	✗	70.3KB
Gold OPRF [75]	Legendre PRF	2	✓	774KB / 100B†
Pool	Binary LWR (100 bits sec.)	2	✓	9.2KB
Pool	Binary LWR (128 bits sec.)	2	✓	11.9KB

Table 6: Brief comparison of Pool with malicious OPRF constructions (without quantum cryptanalysis and for security levels of 100 bits, or higher), interpreted in the semi-honest regime (e.g. bandwidth associated with ZK proofs/commitments are removed). †Communication value corresponds to a batched evaluation setting, where costs of a single query are amortized over the total batch size. *All key-/input-independent messages (i.e. all messages apart from the last two of [21, Fig. 13]) are counted as preprocessing here. **The scheme utilizes a trusted setup, but it is suggested that using preprocessing allows removing the trusted setup. ***Parameters derived from [50, Tab. 4], with correctness parameter $2^{-\kappa} = 2^{-64}$. ****The client sends a bootstrapping key of size 14.7MB for the first value, and 2.4MB for the second, during a one-time setup.

$\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ interacts with a sender S and a receiver R (and possibly an actively malicious sender S* and receiver R*).

(1) On input (Receive, rec, uid) from R; if no message of the form (rec, uid, $\mathbf{b}$) has been recorded, sample $\mathbf{b} \leftarrow\$ \mathbb{Z}_\Delta^\ell$, store (rec, uid, $\mathbf{b}$) and send (rec, uid) to S and ($\mathbf{b}$, uid) to R.
(2) On input (Send, sen, uid) from S; if no message of the form (sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) is recorded and a message of the form (rec, uid, $\mathbf{b}$) is stored, sample $\mathbf{a}_0 := (a_{0,1}, \ldots, a_{0,\ell}), \mathbf{a}_1 := (a_{1,1}, \ldots, a_{1,\ell})$ where each $a_{i,j} \leftarrow\$ \mathbb{Z}_p$, store (sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) in memory and send (sent, uid, $(\mathbf{b}, (a_{b_i,i})_{i=1}^\ell)$) to R and (sent, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) to S. Ignore future messages with the same uid.
(3) On input (Receive*, rec, uid, $\mathbf{b}$) from R* where $\mathbf{b} \in \mathbb{Z}_\Delta^\ell$; if no message of the form (rec, uid, $\mathbf{b}$) has been recorded, store (rec, uid, $\mathbf{b}$) and send (rec, uid) to S and ($\mathbf{b}$, uid) to R.
(4) On input (Send*, sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) from S* where $\mathbf{a}_0 = (a_{0,1}, \ldots, a_{0,\ell}), \mathbf{a}_1 = (a_{1,1}, \ldots, a_{1,\ell})$ for $a_{i,j} \in \mathbb{Z}_p$; if no message of the form (sen, uid, $(\mathbf{a}'_0, \mathbf{a}'_1)$) is recorded but a message of the form (rec, uid, $\mathbf{b}$) is stored, send (sent, uid, $(\mathbf{b}, (a_{b_i,i})_{i=1}^\ell)$) to R and (sent, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) to S. Ignore future messages with the same uid.

$\mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}$ interacts with a sender S and a receiver R (and possibly an actively malicious sender S* and receiver R*).

(1) On input (Receive, rec, uid) from R; if no message of the form (rec, uid, $\mathbf{b}$) has been recorded, sample $\mathbf{b} \leftarrow\$ \{0,1\}^\ell$, store (rec, uid, $\mathbf{b}$) and send (rec, uid) to S and ($\mathbf{b}$, uid) to R.
(2) On input (Send, sen, uid) from S; if no message of the form (sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) is recorded and a message of the form (rec, uid, $\mathbf{b}$) is stored, sample $\mathbf{a}_0 := (a_{0,1}, \ldots, a_{0,\ell}), \mathbf{a}_1 := (a_{1,1}, \ldots, a_{1,\ell})$ where each $a_{i,j} \leftarrow\$ \mathbb{Z}_q^\tau$, store (sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) in memory and send (sent, uid, $(\mathbf{b}, (a_{b_i,i})_{i=1}^\ell)$) to R and (sent, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) to S. Ignore future messages with the same uid.
(3) On input (Receive*, rec, uid, $\mathbf{b}$) from R* where $\mathbf{b} \in \{0,1\}^\ell$; if no message of the form (rec, uid, $\mathbf{b}$) has been recorded, store (rec, uid, $\mathbf{b}$) and send (rec, uid) to S and ($\mathbf{b}$, uid) to R.
(4) On input (Send*, sen, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) from S* where $\mathbf{a}_0 = (a_{0,1}, \ldots, a_{0,\ell}), \mathbf{a}_1 = (a_{1,1}, \ldots, a_{1,\ell})$ for $a_{i,j} \in \mathbb{Z}_q^\tau$; if no message of the form (sen, uid, $(\mathbf{a}'_0, \mathbf{a}'_1)$) is recorded but a message of the form (rec, uid, $\mathbf{b}$) is stored, send (sent, uid, $(\mathbf{b}, (a_{b_i,i})_{i=1}^\ell)$) to R and (sent, uid, $(\mathbf{a}_0, \mathbf{a}_1)$) to S. Ignore future messages with the same uid.

Figure 5: Idealized Random OT functionalities. The sent strings in $\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ are in $\mathbb{Z}_p$ and the choice "bits" are in $\mathbb{Z}_\Delta$. In $\mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}$, the sent strings are in $\mathbb{Z}_q^\tau$ and the choice values are bits.

C RANDOM OT

The formal definition of the *random* OT-functionalities that Pool is designed from are given in Figure 5. These are simply special parametrizations of the Random OT functionality written in [33]. Since we will be assuming semi-honest adversaries against these functionalities, the Send* and Receive* will not be considered in any of our full security proofs. This is because such messages are only sent by actively malicious adversaries. However, we leave these in the description of the functionalities as they become useful when considering malicious security.

$\binom{2}{1}$-OT extension	(n, p, q)	# evals	τ	Communication		Computation	
				Client	Server	Client	Server
Silent OT [29] ($n \cdot \kappa$ OTs)	$(415, 2^4, 2^8)$	2^0	2^5	792 kB	792 kB	108 ms	107 ms
		2^{13}	2^{18}	926 kB	1.9 MB	2.8 s	4.7 s
	$(482, 2^8, 2^{12})$	2^0	2^4	800 kB	795 kB	95 ms	94 ms
		2^{13}	2^{17}	924 kB	1.6 MB	2.1 s	3.6 s
Silent OT [29] (n OTs)	$(415, 2^4, 2^8)$	2^0	2^5	792 kB	792 kB	154 ms	154 ms
		2^{13}	2^{18}	815 kB	887 kB	8.6 s	8.3 s
	$(482, 2^8, 2^{12})$	2^0	2^4	792 kB	792 kB	176 ms	176 ms
		2^{13}	2^{17}	813 kB	879 kB	7 s	7 s

Table 7: Preprocessing performance for alternative implementations of Silent OT.

D DEFERRED PROOFS

D.1 Correctness

Proof. Take any Gst_S, inputs $(t_j, x_j)_{j=1}^{\tau}$ and pk, sk in the output space of $\mathcal{F}.\mathsf{PK} \times \mathcal{F}.\mathsf{SK}$. Calling $\mathsf{PreProc}(\mathsf{sk})$ yields a state st_C and st_S where the latter is added to Gst_S. Throughout, we use the state notation explicit in Figure 4. We now show for $j = 1, \ldots, \tau$, that the z_j's calculated after running $\mathcal{F}.\mathsf{Request}$, $\mathcal{F}.\mathsf{BlindEval}$ and $\mathcal{F}.\mathsf{Finalize}$ on (t_j, x_j) satisfy $z_j = \mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t_j, x_j)$ as required by the τ-correctness definition (Definition 3.2). As the same client state is used throughout, uid remains unchanged throughout this proof.

For $j = 1$, we have that the value of ctr in st_C gets set to 1 during its $\mathcal{F}.\mathsf{Request}(t_1, x_1)$ call. During this call, elements r_i are defined and the vector $\mathbf{e} = (e_1, \ldots, e_n)$ is calculated inside the loop over $i \in [n]$. The ultimate request takes the form $\mathsf{req}_1 = (t_1, \mathbf{e}, \bar{b}', \mathsf{uid})$ where uid tells the server which part of its Gst_S to use for its response. Additionally, C stores $\mathsf{st}^1_{\mathsf{fin}} = ((\mathsf{uid}, \mathsf{ctr}), r_\Sigma, r'_{b'_{\mathsf{ctr}}})$ for later.

Next we consider the call $\mathcal{F}.\mathsf{BlindEval}(\mathsf{sk}, t_1, \mathsf{req}_1; \mathsf{Gst}_\mathsf{S})$ that computes a response for $j = 1$. Since for every $i = 1, \ldots, n$, Gst_S contains $r^1_{b_i, i}$ for some $b_i \in \{0, 1\}$, we can show that $\tilde{a}_{i,\mathsf{sk}_i} = a_i \cdot \mathsf{sk}_i + r_i$ where r_i was defined during the $\mathcal{F}.\mathsf{Request}(\mathsf{pk}, t_1, x_1; \mathsf{st}_\mathsf{C})$ call. To see why, recall that $\bar{b}_i = \mathsf{sk}_i \oplus b_i$ from preprocessing. So if $\mathsf{sk}_i = 0$, we have $\tilde{a}_{i,\mathsf{sk}_i} = r^1_{b_i,i} = r_i$. On the other hand, if $\mathsf{sk}_i = 1$, $\tilde{a}_{i,\mathsf{sk}_i} = a_i + r_i + r^1_{\mathsf{sk}_i \oplus \bar{b}_i} - r^1_{b_i,i} = a_i + r_i$. Therefore, $\tilde{a}_i = \tilde{a}_{\mathsf{sk},i} = a_i \cdot \mathsf{sk}_i + r_i$, implying that $\tilde{a}_\Sigma = \mathbf{a}^\top \cdot \mathsf{sk} + r_\Sigma$. It follows that the response rep_1 consists of elements of the form $y_i = \lceil \mathbf{a}^\top \mathsf{sk} + r_\Sigma - i \rfloor_{q,p} + r'_{i - \bar{b}'}$ for $i \in [0, \Delta - 1]$. In particular, we write $\mathsf{rep}_1 = ((y_0, \ldots, y_{\Delta-1}), \mathsf{uid}, \mathsf{ctr})$.

During $\mathcal{F}.\mathsf{Finalize}(\mathsf{rep}_1, \mathsf{st}^1_{\mathsf{fin}})$, the client retrieves r_Σ that was calculated during req_1 and $r = r'_{b'_{\mathsf{ctr}}}$. We now write $r_\Sigma = k \cdot \Delta + r_\Sigma \bmod \Delta$ for $k = \frac{r_\Sigma - (r_\Sigma \bmod \Delta)}{\Delta} \in \mathbb{Z}$. It then computes

$$\begin{aligned} y \leftarrow y_{r_\Sigma \bmod \Delta} - r &= \lceil \mathbf{a}^\top \mathsf{sk} + r_\Sigma - (r_\Sigma \bmod \Delta) \rfloor_{q,p} \\ &= \lceil \mathbf{a}^\top \mathsf{sk} + k \cdot \Delta \bmod q \rfloor_{q,p} \\ &= \lceil \mathbf{a}^\top \mathsf{sk} \rfloor_{q,p} + k. \end{aligned}$$

The first equality follows by virtue of the fact that $b'_{\mathsf{ctr}} = r_\Sigma - \bar{b}' \bmod \Delta$, the second by definition of k and the third by the fact that $\Delta = q/p \in \mathbb{Z}$. Since the output of finalize subtracts k from y, it follows that $z_1 = \mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t_1, x_1)$.

As the counter ctr in states st_C and st_S with respect to uid remain synchronized, it follows by an identical argument that $z_j = \mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t_j, x_j)$ for $j \in [2, \tau]$ which completes the proof. □

D.2 Semi-Honest Client Proof

Proof. (Of Theorem 5.2) We step through the sequence of games following the high-level overview included in the main body.

G_0 : This is the $\text{SH-PPOPRF}^{\mathcal{A},0}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$ game with Sim described by the unhighlighted pseudocode in Figure 6.

G_1 : This is G_0 apart from the change highlighted in gray. In particular, the random function $\mathsf{RO}_{\mathsf{Fn}}$ which takes inputs $(t, x) \in \mathcal{F}.\mathsf{T} \times \mathcal{F}.\mathsf{In}$ is replaced by the LWR function $\lceil \mathsf{Sim.Eval}(t, x)^\top \mathsf{sk}_{\mathsf{Sim}} \rfloor_{q,p}$. By Lemma D.1, we have $|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{LWR}^{\{0,1\}}_{\mathsf{n,p,q}}}_{\mathcal{B}}(\lambda)$ for some PPT adversary $\mathcal{B}$ if RO behaves like a random oracle.

G_2 : This is G_1 apart from that during Sim.PreProc, the call to $\mathcal{F}.\mathsf{PreProc}$ takes input $\mathsf{sk}_{\mathsf{Sim}}$ instead of $\mathbf{0}$ as highlighted in yellow. Looking closely at $\mathcal{F}.\mathsf{PreProc}(\mathsf{sk}_{\mathsf{Sim}})$ in Figure 3 and using the same notation, the input sk is only used in the computation of $\bar{b}_i \leftarrow b_i \oplus \mathsf{sk}_i$. However, the client's view during preprocessing i.e., st_C is independent of b_i due to the $\mathcal{F}^{q,\tau,n}_{\binom{2}{1}\text{-rOT}}$ functionality. Further, the entire view of $\mathcal{A}$ (with respect to all oracle responses computed by Sim) is independent of b_i which means that $\bar{b}_i$ is uniformly distributed from $\mathcal{A}$'s perspective. Therefore, b_i serves as a one-time pad that hides sk_i during preprocessing. Using this argument over all calls to $\mathsf{Sim.PreProc}()$, we arrive at $\Pr[G_2 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1]$.

G_3 : This is G_2, apart from a change in the computation of y_i during Sim.BlindEval highlighted in orange in Figure 6. In particular, for $i \in [0, \Delta - 1] \setminus \{r_\Sigma \bmod \Delta\}$,

$$y_i \leftarrow \lceil \mathsf{Sim.Eval}(t, x)^\top \mathsf{sk}_{\mathsf{Sim}} + r_\Sigma - i \bmod q \rfloor_{q,p} + r'_{i - \bar{b}' \bmod \Delta, \mathsf{ctr}} \bmod p \qquad (3)$$

for $\bar{b}' \in [0, \Delta - 1]$ used in the request message. We now note from Figure 3 that $\{r'_{b'_i, i}\}_{i=1}^{\tau}$ is saved as part of st_C whereas $R'_{\mathsf{hidden}} := \{r'_{k,i} : k \in [0, \Delta - 1] \setminus \{b'_i\}\}_{i=1}^{\tau}$ are uniformly chosen values hidden from $\mathcal{A}$ during a preprocessing oracle query, thanks to the $\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ functionality. The only time R'_{hidden} is used in $\mathcal{A}$'s oracle queries is when y_i is computed according to (3). Therefore, the y_i's computed in G_3 remain uniformly distributed from $\mathcal{A}$'s perspective since the uniformly distributed elements of R'_{hidden} are used as one-time pads in (3). Applying this argument over all Sim.BlindEval queries made by $\mathcal{A}$, we have $\Pr[G_3 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1]$.

G_4 : This final game is precisely $\text{SH-PPOPRF}^{\mathcal{A},1}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$. It is immediately clear that the behavior of Sim in G_3 implies that oracles Eval, PreProc, LimitEval, Prim compute their outputs identically to those same oracles in G_4. This follows easily from the fact that $(\mathsf{sk}, \mathsf{pp}_1)$ and $(\mathsf{sk}_{\mathsf{Sim}}, \mathsf{pp}_0)$ are identically distributed and the instructions defining (Sim.PreProc, Sim.Eval, Eval) in G_3 are identical to those defining ($\mathcal{F}.\mathsf{PreProc}$, RO, Eval) in G_4 where RO is assumed to behave as a random oracle. What is left is to show that the behavior of Sim.BlindEval in G_3 is identical to $\mathcal{F}.\mathsf{BlindEval}$ in G_4. Take any (t, x, req) and st_C. In G_4, we have that $\tilde{a}_\Sigma$

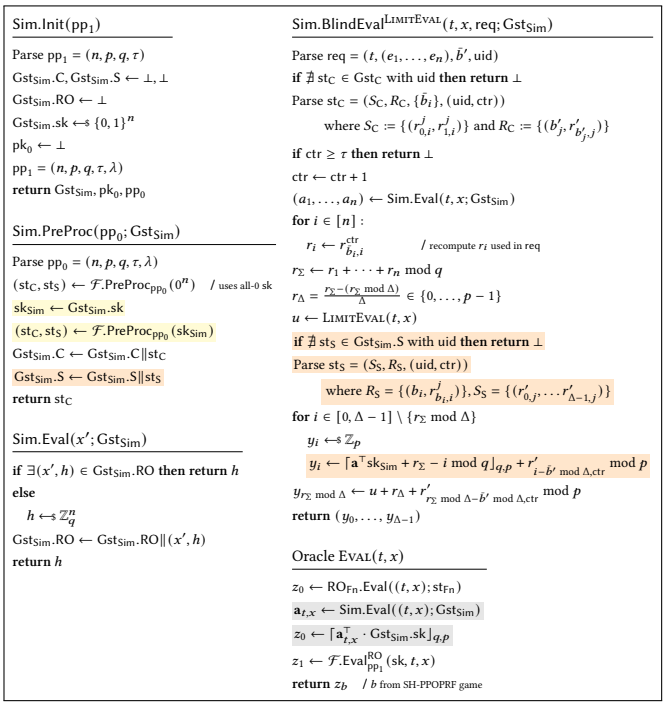


Figure 6: The Sim algorithms for the proof of Theorem 5.2. The original simulator algorithms are not highlighted. The first , second and third game changes are highlighted in gray , yellow and orange respectively.

from $\mathcal{F}$.BlindEval satisfies $\tilde{a}_\Sigma = \mathsf{RO}(t,x)^\top \mathsf{sk} + r_\Sigma$ where $r_\Sigma = r_1 + \cdots + r_n$ for the blinding factors used to produce req. Therefore, for $\mathsf{sk} \in \{0,1\}^n$, we have $\forall i \in [0, \Delta - 1]$, that

$$y_i \leftarrow \lceil \mathsf{RO}(t,x)^\top \mathsf{sk} + r_\Sigma - i \rfloor_{q,p} + r_{i - \bar{b}' \bmod \Delta, \mathsf{ctr}} \bmod p.$$

For $i \in [0, \Delta - 1] \setminus \{r_\Sigma \bmod \Delta\}$, it is clear that y_i is computed identically in G_3's Sim.BlindEval and G_4's $\mathcal{F}$.BlindEval. Now for $i^* = r_\Sigma \bmod \Delta$, we can write

$$\begin{aligned} y_{i^*} &= \lceil \mathsf{RO}(t,x)^\top \mathsf{sk} + r_\Sigma - i^* \rfloor_{q,p} + r_{i^* - \bar{b}' \bmod \Delta} \\ &= \lceil \mathsf{RO}(t,x)^\top \mathsf{sk} + (r_\Sigma - r_\Sigma \bmod \Delta) \rfloor_{q,p} + r_{i^* - \bar{b}' \bmod \Delta} \bmod p \\ &= \lceil \tfrac{1}{\Delta} \mathsf{RO}(t,x)^\top \mathsf{sk} + r_\Delta \rceil + r_{i^* - \bar{b}' \bmod \Delta} \bmod p \\ &= \lceil \mathsf{RO}(t,x)^\top \mathsf{sk} \rfloor_{q,p} + r_\Delta + r_{i^* - \bar{b}' \bmod \Delta} \bmod p \end{aligned}$$

where $r_\Delta := \frac{r_\Sigma - (r_\Sigma \bmod \Delta)}{\Delta} \in \mathbb{Z}$. From here, it is clear that the instructions of $\mathcal{F}$.BlindEval and Sim.BlindEval from G_3 produce identically distributed responses for any given (t, x, req) and st_C. Therefore, G_3 and G_4 are identically distributed and $\Pr[G_3 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1]$.

Applying the triangle inequality yields

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_4 \Rightarrow]| \leq \sum_{i=0}^{3} |\Pr[G_i \Rightarrow 1] - \Pr[G_{i+1} \Rightarrow 1]| = \mathsf{Adv}^{\mathsf{LWR}_{n,p,q}}_{\mathcal{B}}(\lambda)$$

completing the proof. □

Lemma D.1. *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that* $|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| = \mathsf{Adv}^{\mathsf{LWR}_{n,p,q}}_{\mathcal{B}}(\lambda)$.

Proof. We describe a PPT algorithm $\mathcal{B}$ that attempts to distinguish between an oracle O_0 that returns uniform samples $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$ and an oracle O_1 that returns $\mathsf{LWR}_{n,p,q}$ samples $(\mathbf{a}, \lceil \mathbf{a}^\top \mathbf{s} \rfloor_{q,p})$ where $\mathbf{a} \sim U(\mathbb{Z}_q^n)$ are freshly sampled and $\mathbf{s} \sim U(\{0,1\}^n)$ is reused across all queries. For $b \in \{0,1\}$, $\mathcal{B}$ will perfectly simulate G_b if it has oracle access to O_b.

$\mathcal{B}$ runs G_0 using $\mathcal{A}$ as a subroutine, by executing all steps honestly apart from the Sim.Eval and Eval algorithms. In order to implement Sim.Eval on inputs x' that have not been queried before, $\mathcal{B}$ queries its challenge oracle O_b to obtain $(\mathbf{a}_{x'}, u_{x'})$, adds $(x', \mathbf{a}_{x'})$ to $\mathsf{Gst}_{\mathsf{Sim}}$.RO and returns $\mathbf{a}_{x'}$ to $\mathcal{A}$. Clearly, this precisely emulates Sim.Eval as the outputs are uniform values regardless of whether it is interacting with O_0 or O_1. Repeated evaluations are handled as described in Sim (i.e. are answered consistently). Additionally $\mathcal{B}$ saves $u_{x'}$ alongside $(x', \mathbf{a})$ for later. Now, whenever $\textsc{Eval}(x')$ is called for x' of the form (t, x), $\mathcal{B}$ simply runs its emulation of Sim.Eval(x'), fetches $u_{x'}$ and sets $z_0 \leftarrow u_{x'}$. The final output of $\mathcal{B}$ is set to the output of $\mathcal{A}$.

If $\mathcal{B}$ is interacting with O_1, we argue that it perfectly simulates G_1 even though Gst.sk is not necessarily the vector $\mathbf{s}$ used in $\mathcal{B}$'s simulation of Eval. To see why, recall that in G_1 (the gray highlighted game of Figure 6), the value of $\mathsf{Gst}_{\mathsf{Sim}}$.sk is only ever used via oracle access to Eval which sets z_0 according to $\lceil \mathbf{a}^\top \mathbf{s} \rfloor_{q,p}$. Therefore, when $\mathcal{B}$ interacts with O_1, the view of $\mathcal{A}$ is independent of $\mathsf{Gst}_{\mathsf{Sim}}$.sk and depends only on $\mathbf{s}$. In particular, $\mathcal{A}$'s output distribution is identical whether $\mathsf{Gst}_{\mathsf{Sim}}.\mathsf{sk} = \mathbf{s}$ or not. Since $\mathbf{s}$ and $\mathsf{Gst}_{\mathsf{Sim}}$.sk are identically distributed, $\Pr[\mathcal{B}^{O_1} \Rightarrow 1] = \Pr[G_1 \Rightarrow 1]$.

On the other hand, if $\mathcal{B}$ is interacting with O_0, the values $u_{x'}$ are truly uniform values. In other words, the z_0 values used to answer Eval queries are essentially random function evaluations as in G_0. As a result, $\Pr[\mathcal{B}^{O_0} \Rightarrow 1] = \Pr[G_0 \Rightarrow 1]$. We may now conclude that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| = |\Pr[\mathcal{B}^{O_0} \Rightarrow 1] - \Pr[\mathcal{B}^{O_1} \Rightarrow 1]|$$

which completes the proof. □

D.3 Semi-Honest Server Proof

Proof. (Of Theorem 5.3) We step through the sequence of games following the high-level overview included in the main body.

$G_{0,b}$: For $b \in \{0,1\}$, this is the original $\mathrm{POPRIV1}^{\mathcal{A},b}_{\mathcal{F},\mathsf{Sim},\mathsf{RO}}(\lambda)$ game.

$G_{1,b}$: For $b \in \{0,1\}$, this game adapts G_0 in the way that z_b and z_{1-b} are computed during calls to Run. Specifically, z_b is set to $\mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t, x_0)$ and z_{1-b} is set to $\mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t, x_1)$ in the cases where Run does not abort. This means that the responses are no longer used to compute (z_b, z_{1-b}). Note that due to the description of $(\mathcal{F}.\mathsf{Request}, \mathcal{F}.\mathsf{BlindEval}, \mathcal{F}.\mathsf{Finalize})$, Run only aborts when the adversary asks it to run the PPOPRF on states where the counter ctr is at least τ. It then follows from the proof of τ-correctness, that the values (z_b, z_{1-b}) are identical when considering $G_{0,b}$ or $G_{1,b}$. Therefore, $\Pr[G_{0,b} \Rightarrow 1] = \Pr[G_{1,b} \Rightarrow 1]|$.

$G_{2,b}$: For $b \in \{0,1\}$, this game adapts $G_{1,b}$ by replacing the $\bar{b}'$ term in both req_0 and req_1 during Run calls by independent and uniformly chosen values in $\mathbb{Z}_\Delta$. Note that $\bar{b}' = (r_\Sigma \bmod \Delta) - b'_{\mathsf{ctr}} \bmod \Delta$ and that $(b'_{\mathsf{ctr}})_j$ is uniform and independent of all the server states and $\mathcal{A}$'s entire view of the PreProc oracle (thanks to the $\mathcal{F}^{p,\ell}_{\binom{\Delta}{1}\text{-rOT}}$ functionality). We can then infer that $\bar{b}'$ is uniformly distributed from $\mathcal{A}$'s perspective in $G_{1,b}$ and that $\Pr[G_{1,b} \Rightarrow 1] = \Pr[G_{2,b} \Rightarrow 1]|$.

$G_{3,b}$: For $b \in \{0,1\}$, this game adapts $G_{2,b}$ by freshly sampling $(e_1, \ldots, e_n) \leftarrow\!\$\ \mathbb{Z}_q^n$ every time $\mathcal{F}$.Request is run during calls to Run. Note that in $G_{2,b}$, we may write $e_i = a_i + r^{\mathsf{ctr}}_{\bar{b}_i,i} + r^{\mathsf{ctr}}_{1-\bar{b}_i,i}$. Now, from the $\mathcal{F}^{q,\tau,\ell}_{\binom{2}{1}\text{-rOT}}$ functionality, $\mathcal{A}$ learns *either* $r^{\mathsf{ctr}}_{\bar{b}_i,i}$ *or* $r^{\mathsf{ctr}}_{1-\bar{b}_i,i}$. In particular, it does not know both of these values. Since $r^{\mathsf{ctr}}_{\bar{b}_i,i}$ and $r^{\mathsf{ctr}}_{1-\bar{b}_i,i}$ are sampled uniformly by the aforementioned OT functionality, it follows that $(e_1, \ldots, e_n)$ is uniformly distributed from $\mathcal{A}$'s perspective in $G_{2,b}$ by uniformity of the unknown term. It follows that $\Pr[G_{2,b} \Rightarrow 1] = \Pr[G_{3,b} \Rightarrow 1]|$.

In $G_{3,b}$, the Run oracle outputs triples of the form $T_0 = (\mathsf{req}'_0, \mathsf{rep}'_0, z_0)$ and $T_1 = (\mathsf{req}'_1, \mathsf{rep}'_1, z_1)$. In these triples, req'_j is a simulated request that is independent of any input as $(e_1, \ldots, e_n, \bar{b}')$ are sampled uniformly. Further, rep'_i is just an honestly computed response using a state known to $\mathcal{A}$, and $z_j = \mathcal{F}.\mathsf{Eval}(\mathsf{sk}, t, x_j)$. Therefore, $G_{3,b}$ is totally independent of the challenge bit b. In other words $\Pr[G_{3,0} \Rightarrow 1] = \Pr[G_{3,1} \Rightarrow 1]|$. Traversing through the games from $G_{0,0}$ to $G_{3,0}$ and then from $G_{3,1}$ back to $G_{0,1}$ completes the proof. □