

A post-quantum Distributed OPRF from the Legendre PRF

Novak Kaluderovic¹[0000–0002–4309–5412], Nan Cheng²[0000–0003–2438–4080], and
Aikaterini Mitrokotsa²[0000–0002–7073–0258]

¹ Zircuit, kolja@zircuit.com

² University of St. Gallen, St. Gallen, Switzerland
{nan.cheng,katerina.mitrokotsa}@unisg.ch

Abstract. A distributed OPRF enables a client to evaluate a pseudo-random function on a client-chosen input using a distributed key shared among multiple servers, ensuring that the servers learn nothing about the input or output, while the client learns nothing about the key. We present a post-quantum OPRF suitable for a distributed server setting, requiring only two rounds of communication between the client and servers, with server-to-server communication limited to a pre-computation phase. Our approach leverages the Legendre PRF, computed through a single MPC multiplication and opening during the online phase. We introduce a novel MPC technique that achieves multiplication and opening in one round using replicated secret sharing (RSS) in a malicious adversarial model. This allows for the quantum-secure, verifiable evaluation of the Legendre OPRF against malicious adversaries under a threshold assumption, without requiring inter-server communication. Beyond the Legendre PRF, this method is also of interest for general MPC operations. To our knowledge, our distributed OPRF (dOPRF) is the first post-quantum construction with these properties. We compare our approach to state-of-the-art MPC solutions and provide an implementation of our proposed dOPRF, benchmarking it against existing OPRF constructions. In practical settings, our results demonstrate superior efficiency.

Keywords: Distributed OPRF · Legendre PRF · MPC.

1 Introduction

An oblivious PRF (OPRF) is a 2PC protocol that involves two parties, a client (receiver) and a server (sender) in which the server has a PRF key k and the client learns $F_k(x)$, where F is a PRF and x is the input chosen by the client. The design of the OPRF ensures that the server cannot gain any knowledge about the client’s input or output, while it also prevents the client from learning anything about the server’s key.

OPRFs were originally introduced by Freedman et al. [23], and have since found a wide range of applications. They serve as a fundamental building block for creating password-authenticated key exchange [2, 3], password secret sharing

[10,18], single sign-on protocols [4,10], and private set intersection [31,32], among many other use cases. Many existing practical OPRF constructions base their security on the difficulty of solving the discrete logarithm problem. This reliance makes such constructions vulnerable to attacks by quantum computers. Post-quantum OPRFs are an active area of research, and current constructions are either broken [9,12], inefficient [6,8], only presented in the semi-honest server model [5,19,21], or require many rounds of communication [11,12], and neither of them are distributed. While existing generic MPC protocols [14,15] offer strong security guarantees, they come with multiple rounds of communication between the client and servers, which can lead to delays as the client waits for the servers to exchange data before receiving the final output.

A variation of Oblivious Pseudorandom Functions (OPRFs) is the distributed OPRF (dOPRF), originally introduced by Jarecki et al. [30]. This model involves multiple servers that collaboratively evaluate the function. The secret key k is distributed among these servers, allowing them and the client to jointly compute the function $F_k(x)$ on the client's input x without revealing any information about x to the servers. Furthermore, the distributed nature of the key enhances protection against server breaches. This leads us to the question: *Is it possible to construct a distributed OPRF with optimized communication rounds?* In this work, we provide a positive answer by introducing a dOPRF protocol in merely two rounds with post-quantum security guarantees in a fully malicious model.

Our Contributions. (i) *We proposed a verifiable, non-interactive maliciously secure MPC multiply-and-open protocol.* This 2-round protocol is based on a Replicated Secret Sharing (RSS) scheme and serves as the foundation for our dOPRF protocol. This protocol can also be applied directly to compute other applications such as the inner product of two vectors, used in biometric authentication. (ii) *We proposed a 2-round dOPRF in the fully malicious adversarial model.* Leveraging above *multiply-and-open* protocol, we construct a dOPRF where a client and multiple servers collaboratively compute $\mathcal{C}_{\text{PRF}} : (x, k, s^2) \mapsto (x + k)s^2$. The client shares their input with the servers and receives the output in just two communication rounds. Our proposed dOPRF ensures *output verification*, allowing the client to confirm the correctness of the output. Moreover, it also guarantees *input verification*, which protect honest servers from a malicious client, ensuring they do not leak sensitive data, resulting in a random output when the client misbehaves. (iii) *We performed detailed and comprehensive evaluation over our dOPRF primitive.* We did comprehensive comparison between our protocol and existing state-of-the-arts, and thus verified its practical efficiency with small amount of servers.

1.1 Adversarial model

We design our protocols to withstand malicious adversarial behavior, where an adversary is considered *malicious* if they can adopt any strategy within their capabilities, while being probabilistic polynomial-time bounded. The participants are the client and n servers, with t representing the threshold of corrupted servers. The client may or may not be corrupted, independent of t corrupted

servers. Our protocols are provably secure when $t < n/3$ in a malicious setting. Non-corrupted servers are *honest*, adhering to the protocol without collusion.

We assume secure communication channels between the client and servers, including inter-server communication during pre-computation. Servers hold secret shares as randomness seeds, which can be generated via a non-interactive key exchange (NIKE) method. These seeds are utilized for generating random values during the protocol’s setup. Finally, we assume a static adversary, meaning the set of corrupted parties is fixed at the protocol’s start.

1.2 Highlevel Overview

The Legendre PRF can be computed using simple field arithmetic. We present an algorithm for the Legendre OPRF in a malicious setting (Section 5.2), structured in two phases as illustrated in Fig. 1. Both phases ensure post-quantum security, with the only non-information-theoretic components being hash functions and PRFs used for verification. The computation of the Legendre PRF function $F_k(x)$ is information-theoretically equivalent to revealing the field value $(x+k)s^2$, where s is a random non-zero field element unknown to both the client and servers. Thus, computing the Legendre PRF involves one squaring, one addition, and one multiplication.

We utilize replicated secret sharing (RSS) for secure computation, storing each field element x, k, s^2 as RSS shares among the servers. The linearity of RSS facilitates straightforward secure addition. For the efficient computation of the Legendre PRF, we focus on two operations: *RSS to RSS multiplication* (Section A) and *RSS to RSS multiplication-with-opening* (Section 4.1), corresponding to the square computation in the offline and online phase, respectively.

1.3 Related work

Oblivious PRFs have been systematically studied by Casacuberta et al. [13]. The authors divide them into four main categories based on the underlying assumptions – Naor-Reingold (NR), Hashed Diffie-Hellman (HashDH and 2HashDH), and Dodis-Yampolskiy (DY), all of which are based on classical security assumptions. In Tab. 1 we compare some popular OPRFs against ours. *Partially oblivious* property means that the input provided by the client is split into a private part x_{priv} which remains secret, and public part x_{pub} which is revealed to the server(s). *Verifiability* allows the client to be convinced of the correctness of the received output. A *distributed* OPRF works in a multi-server setting (i.e., the secret key is distributed to multiple servers). In our case, the verifiability stems from the threshold assumption of a (super)majority of honest servers.

In Tab. 1, we compare our protocol with state-of-the-art oblivious pseudo-random functions (OPRFs) across various characteristics, including security and efficiency. The comparison highlights the partially oblivious and verifiable properties, as well as the ability to operate in a distributed manner. Our proposed protocol, based on the Legendre PRF, demonstrates significant advantages with

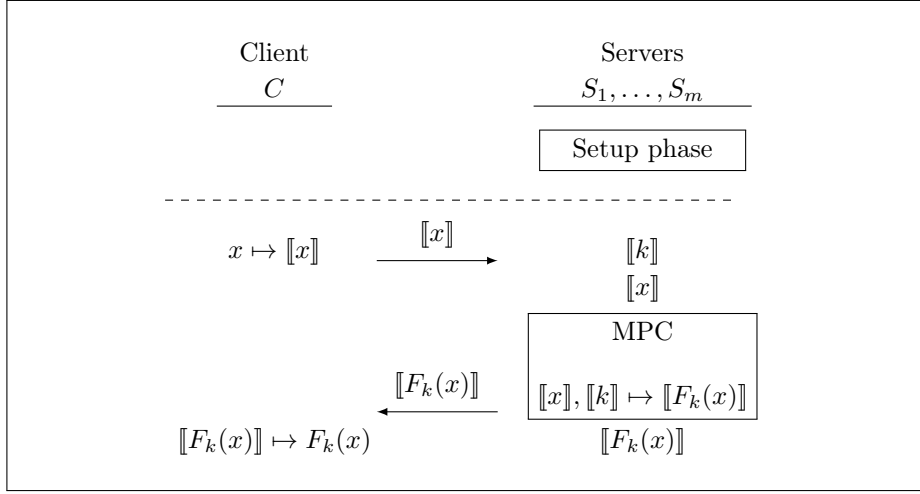


Fig. 1: MPC based distributed OPRF

respect to both communication efficiency and computational performance, requiring only two rounds for its execution.

Recent research has explored various OPRF constructions. Notable works include lattice-based constructions by Albrecht et al. [6] and isogeny-based protocols by Boneh et al. [12]. Dodgson [20] established security for the Legendre OPRF in the UC model, while Beullens et al. [11] proposed a practical approach with significant communication rounds. Faller et al. [21] discussed efficient OPRFs using garbled circuits, and Albrecht et al. [5] provided a post-quantum OPRF that is efficient but incomplete without zero-knowledge proofs. In contrast, our dOPRF exhibits superior efficiency compared to existing protocols, requiring fewer communication rounds.

On distributed security. We note that our protocol offers different security properties in the distributed and threshold setting than previous classical constructions. Other dOPRFs preserve the secrecy of the input x when an adversary takes control of $> t$ servers. In our scenario a weaker security assumption holds – both the input x and the key k are revealed once a malicious party breaches a threshold of the servers. This may seem like a strong security relaxation, we argue that the benefits outweigh the drawbacks. A malicious party that takes full ($> t$ threshold) control of the servers always learns the secret key k . This allows for a brute-force dictionary attack which can be used to reveal users’ inputs. In general users’ inputs are passwords (or hashes thereof) which are not assumed to be of high-entropy, and this attack would reveal any such inputs. A quantum computer would speed these attacks even further by employing a faster search algorithm such as Grover’s search [26].

Table 1: Comparison to state-of-the-art OPRFs.

OPRF	PRF	Partially oblivious	Verifiable	Distributed	Quantum resistant
[27, 29]	NR	✗	✓	✗	✗
[31]	DY	✗	✓	✗	✗
[10, 28, 30]	2HashDH	✓	✗	✓	✗
[6, 12]	Isogenies	✗	✓	✗	✓
[21]	GC	✗	✗	✗	✓
[11, 20]	Legendre PRF	✗	✓	✗	✓
Our work	Legendre PRF	✓	✓	✓	✓

2 Preliminaries

Notation. Let p be a prime. We assume that p is public and $p \approx 2^{2\lambda_c}$ or $p \approx 2^{3\lambda_q}$ where λ_c and λ_q are classical and quantum security parameters respectively. We denote with $\mathbb{F}_p$ the finite field of p elements. For any set S , we denote with $\#S$ the cardinality of S . We use subscripts to distinguish the terms of a sum ($a = \sum a_i$) or to index parties in a protocol (server S_i), while we use superscripts to denote values presumed to be equal and allocated to various parties (a^i given to party associated with index i).

Secret sharing. Secret sharing is a method for distributing a secret value $x \in \mathbb{F}_p$ among multiple parties, so no single party, or a predefined group of parties, holds any information about the secret. We call t a threshold, and a corresponding secret sharing scheme a (t, n) scheme, if any group of at most t colluding parties cannot extract any information about the secret, while any group of $\geq t + 1$ parties can compute the secret x . We denote with $\llbracket x \rrbracket$ a generic secret sharing of a field element $x \in \mathbb{F}_p$, i.e., an n -tuple $(x_1, \dots, x_n)$ such that each share x_i is owned by party i .

Distributed Oblivious PRFs. A *Pseudorandom Function* (PRF) family is a collection of functions $\{F_k\}_{k \in \mathcal{K}}$ with the same domain and co-domain, indexed by some key set $\mathcal{K}$, where a randomly sampled function from this family is difficult to be distinguished from a truly random function with the same domain and co-domain. An *Oblivious Pseudorandom Function* (OPRF) is a two-party privacy-preserving secure PRF protocol between a client and a server. An OPRF is called *distributed* (dOPRF) if there are multiple servers taking the role of the OPRF evaluator. A distributed (t, n) OPRF (dOPRF) is an OPRF variant which tolerates up to t malicious colluding servers among n servers.

2.1 The Legendre PRF

The Legendre PRF originated from Damgård [16]. Although initially impractical due to being significantly slower than generic PRF counterparts, interest in the Legendre PRF [22] increased after Grassi et al. [25] identified it as a suitable PRF for MPC, crediting to its homomorphic property of the Legendre symbol and its simple structure, making it MPC friendly.

Definition 1 (Legendre symbol). We define the Legendre symbol:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} = \begin{cases} 1 & \text{if } a \in \mathbb{F}_p \text{ is a square mod } p \\ -1 & \text{if } a \in \mathbb{F}_p \text{ is not a square mod } p. \end{cases}$$

In general, the Legendre symbol is defined by setting $\left(\frac{0}{p}\right) = 0$, which makes the symbol multiplicative, but comes at a cost of increasing the size of the co-domain. We will assume that $\left(\frac{0}{p}\right) = 1$ and that the symbol is multiplicative, which is a reasonable assumption, since for our use-case the inputs are random, and the probability that the input is $x = 0$ is negligible since the input is an element not known to the adversary. In particular, we assume

$$\left(\frac{a}{p}\right) \left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right) \quad \text{for all } a, b \in \mathbb{F}_p. \quad (1)$$

Definition 2 (Legendre PRF). We define the Legendre PRF to be the family of functions $\{F_k\}_{k \in \mathcal{K}}$ the key space being $\mathcal{K} = \mathbb{F}_p$, the functions parameterised as $F_k : \mathbb{F}_p \rightarrow \{-1, +1\}$ and defined as $F_k(x) := \left(\frac{x+k}{p}\right)$. Under this definition, the Legendre PRF outputs only a single bit. These functions can be expanded to a λ -bit output function by increasing the keyspace to $\mathcal{K}^\lambda$ and for each $k = (k_1, \dots, k_\lambda) \in \mathcal{K}^\lambda$ defining

$$F_k(x) := \left(\left(\frac{x+k_1}{p}\right), \left(\frac{x+k_2}{p}\right), \dots, \left(\frac{x+k_\lambda}{p}\right) \right).$$

Multi-party Legendre Symbol Computation. The Legendre symbol $\left(\frac{a}{p}\right)$ of a secret shared input $\llbracket a \rrbracket$ can be computed via MPC. During the setup phase, the servers sample a random value $\llbracket s \rrbracket$ and compute $\llbracket s^2 \rrbracket$. In the subsequent online evaluation, the servers compute $\llbracket a \rrbracket, \llbracket s^2 \rrbracket \mapsto \llbracket as^2 \rrbracket$ and reveal the result to a designated receiver. This enables the receiver to locally compute the Legendre symbol as follows: $\left(\frac{as^2}{p}\right) = \left(\frac{a}{p}\right) \left(\frac{s^2}{p}\right) = \left(\frac{a}{p}\right)$.

Legendre PRF hardness assumptions. The security of the Legendre PRF relies on the Shifted Legendre Symbol Problem (SLSP) and the Decisional Shifted Legendre Symbol Problem (DSLSP) [35]. SLSP is at least as hard as DSLSP, indicating $\text{DSLSP} \leq \text{SLSP}$ under polynomial time reductions. For a more detailed security property of the Legendre PRF, please refer to the full version of this work [33].

3 Arithmetic MPC and Secret Sharing

We refer to the parties that hold secret shares and participate in multi-party computation as *servers*. The terms *server* and *server index* are used interchangeably to denote a server $S_1, \dots, S_n$ or its corresponding index $i = 1, \dots, n$. Similarly, we use *share* and *share index* to refer to a share of a secret held by some servers and the index of that share.

Additive secret sharing (ASS) [1] shares a field element $x \in \mathbb{F}_p$ among at least two servers indexed by some set $I \subseteq \{1, \dots, n\}$, with the security threshold

$t = \#I - 1$. Any $\leq t$ colluding parties have no information about x . We denote an ASS secret sharing of x as $[x]$, in this scheme, let $\mathcal{T} = I$ be the set of share indices, then each server S_i holds the term x_T where $T = i$ satisfying $x = \sum_{T \in \mathcal{T}} x_T$. **Replicated secret sharing (RSS)** is a generalisation of ASS, in which the same additive secret shares of an element are provided to multiple parties. Following the notation of [7], we have:

Definition 3 (Replicated secret sharing). *The (t, n) -Replicated secret sharing scheme among n servers is defined as follows: The set of share indices denoted as $\mathcal{T}$ is:*

$$\mathcal{T} := \{A \subseteq \{1, \dots, n\} \mid \#A = t\},$$

and an element $x \in \mathbb{F}_p$ is secret shared by constructing an additive secret sharing of size $\#\mathcal{T} = \binom{n}{t}$, i.e.

$$x = \sum_{T \in \mathcal{T}} x_T$$

where all but one addends are sampled uniformly at random from $\mathbb{F}_p$, and the last one is selected so that the equation above is satisfied. The shares held by server S_i are exactly those x_T of index T where $i \notin T$. An RSS secret sharing of a field element x is denoted with $\llbracket x \rrbracket_R$.

Additionally, we define $\mathcal{T}_i \subseteq \mathcal{T}$ to be the set of shares held by party S_i , i.e., $\mathcal{T}_i := \{T \in \mathcal{T} \mid i \notin T\}$; the shares held by any pair of servers S_i, S_j as $\mathcal{T}_{ij} := \mathcal{T}_i \cap \mathcal{T}_j$. Each server S_i holds exactly $\binom{n-1}{t}$ shares. $\mathcal{S}_T \subseteq \{1, \dots, n\}$ is defined as the set of indices of servers holding the share T , $\mathcal{S}_T := \{i \in \{1, \dots, n\} \mid T \in \mathcal{T}_i\} = \{1, \dots, n\} \setminus T$; We further define the set of servers holding any pair of shares T_1, T_2 as $\mathcal{S}_{T_1 T_2} := \mathcal{S}_{T_1} \cap \mathcal{S}_{T_2}$. Each share T is held by exactly $n - t$ servers.

By abuse of notation, we denote with $\llbracket x \rrbracket_R := ((x_T^i)_{T \in \mathcal{T}_i})_{i=1, \dots, n}$ the set of shares distributed to the servers by a (possibly corrupt) dealer. Each server S_i receives $(x_T^i)_{T \in \mathcal{T}_i}$ where x_T^i is the share associated to index T . We call $\llbracket x \rrbracket_R$ *valid* if $x_T^i = x_T^j$ for all $i, j \in \mathcal{S}_T$ and all $T \in \mathcal{T}$, then $\llbracket x \rrbracket_R$ is a replicated secret sharing $\llbracket x \rrbracket_R$ of $x = \sum_{T \in \mathcal{T}} x_T^i$; Otherwise the sharing is called *invalid*, and it does not correspond to an RSS sharing of a field element.

Theorem 1 (t-Privacy of Replicated Secret Sharing). *Let $\llbracket x \rrbracket_R$ be an RSS of x distributed to n parties. For any subset of parties $\mathcal{S}$, with $|\mathcal{S}| \leq t$, the following conditions hold:*

1. *The view of the parties in $\mathcal{S}$ reveals no information about the secret x . Formally, for any secret x and any possible shares $s_1, s_2, \dots, s_t$ held by the parties in $\mathcal{S}$,*

$$\mathbb{P}[x \mid (x_T^1)_{T \in \mathcal{T}_1}, (x_T^2)_{T \in \mathcal{T}_2}, \dots, (x_T^t)_{T \in \mathcal{T}_t}] = \mathbb{P}[x].$$

2. *The shares of the parties in $\mathcal{S}$ can be perfectly simulated. That is, there exists a simulator Sim that generates shares $(\hat{x}_T^1)_{T \in \mathcal{T}_1}, (\hat{x}_T^2)_{T \in \mathcal{T}_2}, \dots, (\hat{x}_T^t)_{T \in \mathcal{T}_t}$ indistinguishable from the actual shares, without knowledge of the secret s .*

3.1 Doubly Replicated Secret Sharing

We define *Doubly Replicated Secret Sharing* (DRSS) that is widely used in our dOPRF protocol. DRSS is a variation of RSS that maintains the same access structure but features a different distribution of shares. Consequently, it inherently preserves the same security and privacy properties as RSS; specifically, a (t, n) -DRSS scheme satisfies t -privacy and t -security, just like RSS.

Definition 4 (Doubly replicated secret sharing). *The (t, n) -Doubly replicated secret sharing scheme among n servers is defined similarly to RSS, with the exception that the set $\mathcal{T}^2$ is used instead of $\mathcal{T}$ as the set of share indices.*

An element $x \in \mathbb{F}_p$ is shared by constructing an additive secret sharing of size $\#\mathcal{T}^2 = \binom{n}{t}^2$,

$$x = \sum_{T_1, T_2 \in \mathcal{T}} x_{T_1 T_2},$$

where all but one addends are sampled uniformly at random from $\mathbb{F}_p$, and the last one is selected so that the equation above is satisfied. The shares held by server S_i are exactly those $x_{T_1 T_2}$ where $i \notin T_1 \cup T_2$.

Each server S_i holds shares $\{x_{T_1 T_2} \mid T_1 \in \mathcal{T}_i, T_2 \in \mathcal{T}_i\}$, and that the share $x_{T_1 T_2}$ is held exactly by servers $\mathcal{S}_{T_1 T_2}$. In particular each $x_{T_1 T_2}$ is held by at least 1 server when $t < n/2$. Furthermore, as that $n - 2t \leq \#\mathcal{S}_{T_1 T_2}$ so when $t < n/3$, each share is held by at least $t + 1$ servers, in particular by at least 1 *honest* server. By abuse of notation, we denote with $\llbracket x \rrbracket_D := ((x_{T_1 T_2}^i)_{T_1, T_2 \in \mathcal{T}_i})_{i=1, \dots, n}$ the set of shares distributed to the servers by a (possibly corrupt) dealer. Each server S_i receives $(x_{T_1 T_2}^i)_{T_1 T_2 \in \mathcal{T}_i}$ where $x_{T_1 T_2}^i$ is the share associated to index $T_1 T_2$. We call $\llbracket x \rrbracket_D$ *valid* if $x_{T_1 T_2}^i = x_{T_1 T_2}^j$ for all $i, j \in \mathcal{S}_{T_1 T_2}$ and all $T_1, T_2 \in \mathcal{T}$. Then $\llbracket x \rrbracket_D$ is a doubly replicated secret sharing $\llbracket x \rrbracket_D$ of $x = \sum_{T_1, T_2 \in \mathcal{T}} x_{T_1 T_2}^i$. Otherwise the sharing is called *invalid*, and it does not correspond to a DRSS sharing of a field element.

4 Building Blocks

In this section, we present the building blocks (sub-protocols) for our full malicious dOPRF protocol in section 5. We use the following notation for share indices, $\dot{\mathcal{T}} \subseteq \{1, \dots, n\}$ (ASS), $\dot{\mathcal{T}} = \mathcal{T}$ (RSS) or $\dot{\mathcal{T}} = \mathcal{T}^2$ (DRSS), and the corresponding server set ($I \subseteq \dot{\mathcal{T}}$ for ASS or $\{1, \dots, n\}$ for RSS/DRSS). We assume all protocols design in a RSS- (t, n) malicious setting, where there are up to t servers denoted as $\mathcal{S}_{\text{Mal}}$ controlled by an adversary $\mathcal{A}$ that arbitrarily disrupts protocol execution, no less than $n - t$ servers denoted as $\mathcal{S}_{\text{Hon}}$ honestly follow protocol.

Generating random secret shares. Random secret shares are used both as common reference strings shared among different parties as well as to mask and hide data and secret shares. Due to page limit, we list those very basic primitives from which our protocols are built upon without showing their concrete construction. We assume all secret sharing are *Random secret sharing*; We

define $\mathcal{F}_{\text{GenRand}}$ as the ideal functionality that receives a command from a group of servers, and outputs corresponding random secret sharing of a random field element to all servers; we define $\mathcal{F}_{\text{GenZero}}$ as the ideal functionality that receives a command from a group of servers and outputs corresponding random secret sharing of zero to all servers.

RSS to RSS Multiplication. The first multiplication for computing the Legendre PRF occurs during the setup phase, where servers compute $\llbracket s \rrbracket_R, \llbracket s \rrbracket_R \mapsto \llbracket s^2 \rrbracket_R$, meaning that both the inputs and the output are RSS shares. We implement this using an optimization method based on [36]. We have included the detailed ideal functionality and its discussion in Appendix A.

4.1 RSS to RSS Multiplication with Opening

The second multiplication in Fig. 1 is computed in the online phase, where the servers compute $\llbracket x + k \rrbracket_R \llbracket s^2 \rrbracket_R$ and the client is expecting a result $(x + k)s^2$ revealed to him. We define the ideal functionality of multiplication with opening in Fig. 2 as $\mathcal{F}_{\text{MulOpen}}$, in which there are n servers $\{S_1, \dots, S_n\}$ perform multiply-and-open to a receiver $S_j, j \in \{1, \dots, n\}$. A naive way realizing $\mathcal{F}_{\text{MulOpen}}$ is to first invoke $\mathcal{F}_{\text{Mul}}$ which outputs $\llbracket (x + k)s^2 \rrbracket_R$, and then reveal it to receiver S_j . However, we emphasize that this is communication inefficient (large communication volume in 2 rounds). Instead, we propose a new protocol 1 that computes $\mathcal{F}_{\text{MulOpen}}$ in one round with fewer communication volumes.

In protocol 1, servers invoke $\mathcal{F}_{\text{GenZero}}$ to obtain a DRSS share of zero $\llbracket z \rrbracket_D = (z_{T_1 T_2})_{T_1, T_2 \in \mathcal{T}}$ to mask later computed local products, additionally. Note this step can also be pre-computed in the setup phase for better online efficiency. With that, for $i \in \{1, \dots, n\}$ each server i could locally compute the DRSS terms

$$o_{T_1 T_2} := a_{T_1} b_{T_2} + z_{T_1 T_2}, (T_1, T_2 \in \mathcal{T}_i)$$

for every $T_1, T_2 \in \mathcal{T}_i$. All these local terms from all servers in total constitute a random DRSS sharing of product ab , and they can be returned directly to receiver S_j . Upon receiving messages from all servers, the receiver S_j compares the consistency of $o_{T_1 T_2}^i$ for each $i \in \mathcal{S}_{T_1 T_2}, T_1 T_2 \in \mathcal{T}$. Only if all verification pass that S_j accepts the result $v = \sum_{T_1, T_2 \in \mathcal{T}} o_{T_1 T_2}$, otherwise it rejects.

Efficiency. Each server transmits $\binom{n-1}{t}^2$ elements to the receiver S_j . Let $D = \binom{n}{t}^2$ denote the number of Doubly Replicated Secret Sharing (DRSS) shares of zero in Π_{dOPRF} . Note that for one party holding $\binom{n-1}{t}^2$ DRSS shares, generating each DRSS share requires $D - 1$ PRG calls. Therefore, generating DRSS shares of zero requires a total of $n \binom{n-1}{t}^2 (D - 1)$ PRG calls.

Security. Informally, we demonstrate how protocol 1 ensures both *verifiability* and *privacy*. $\mathcal{S}_{\text{Hon}}$ has access to all RSS shares of a and b , enabling them to generate DRSS shares of ab for every share index pair $T_1, T_2 \in \mathcal{T}$. This redundancy allows for the detection of any additive attack from $\mathcal{S}_{\text{Mal}}$ (verifiability).

We now address the privacy of (a, b) with respect to the receiver S_j , assuming $S_j \in \mathcal{S}_{\text{Mal}}$. Assume in the worst case that $\mathcal{A}$ controls t servers, denote $\hat{s} :=$

$\{\alpha, \beta, \dots\}$ as the set contains all indices of servers from $\mathcal{S}_{\text{Mal}}$, note adversary $\mathcal{A}$ is able to learn all RSS shares of a or b except for the share index $\hat{T} := \hat{s}$. Thus, $\mathcal{A}$ learns nothing about DRSS shares of ab with those share indices in set

$$\hat{\mathcal{T}} = \{\hat{T}T_1, \dots, \hat{T}T_{\max}\} \cup \{T_1\hat{T}, \dots, T_{\max}\hat{T}\}.$$

Players: $S_1, \dots, S_n$ and the ideal functionality $\mathcal{F}_{\text{MulOpen}}$.
Input: $\mathcal{S}_{\text{Hon}}$ inputs their shares $(a_T, b_T)_{T \in \mathcal{T}_i}$ to $\mathcal{F}_{\text{MulOpen}}$, $\mathcal{S}_{\text{Mal}}$ gives input $d \in \mathbb{F}_p$ to $\mathcal{F}_{\text{MulOpen}}$.
Computation: $\mathcal{F}_{\text{MulOpen}}$ reconstructs a, b from received RSS shares of $\mathcal{S}_{\text{Hon}}$, computes $c = ab + d$.
Output: If $d = 0$, $\mathcal{F}_{\text{MulOpen}}$ outputs (c, accept) to S_j . Otherwise, it outputs (c, reject) to S_j .

Fig. 2: Ideal functionality $\mathcal{F}_{\text{MulOpen}}$ - RSS Multiplication with opening up to additive attack.

In another word, $\hat{\mathcal{T}}$ of size $2\binom{n}{t} - 1$ contains indices for which the corresponding DRSS terms that are known only to $\mathcal{S}_{\text{Hon}}$. Each DRSS share with index in $\hat{\mathcal{T}}$ is masked by a random number that is known only to $\mathcal{S}_{\text{Hon}}$, thus, $\mathcal{A}$ learns nothing about each individual product $a_{T_1}b_{T_2}$ for every $T_1, T_2 \in \hat{\mathcal{T}}$. Consequently, there is no leakage (*privacy*) in Protocol 1.

5 The Legendre dOPRF protocol

In the framework of Fig. 1, if a malicious client colludes with $t' \leq t < n/3$ servers, protocol 1 is not applicable because we cannot guarantee that the input RSS of x is valid; that is, each server in $\mathcal{S}_{\text{Hon}}$ must receive identical RSS shares corresponding to the same RSS index. For this reason, we first define the desired ideal functionality $\mathcal{F}_{\text{dOPRF}}$ in Fig. 3 and then present our adapted protocol 2 that securely computes $\mathcal{F}_{\text{dOPRF}}$ against an adversary $\mathcal{A}$ that controls both the client and $t' \leq t < n/3$ servers.

5.1 Ideal functionality in the malicious setting

We assume that an adversary $\mathcal{A}$ corrupts $t' \leq t < n/3$ servers and the client in Fig. 1, and $\mathcal{A}$ might act arbitrarily the way he desires. Specifically, the adversary $\mathcal{A}$ may compromise the client to submit inconsistent replicated secret shares (RSS) of x to the honest servers in order to extract k . Additionally, $\mathcal{A}$ may

Protocol 1 Verifiable RSS multiplication with opening in the $\mathcal{F}_{\text{GenZero}}$ -hybrid model

Players: Server $S_1, \dots, S_n$.**Input :** $\llbracket a \rrbracket_R, \llbracket b \rrbracket_R$ **Output:** (v, accept) or (v, reject) .

```

1 Servers invoke  $\mathcal{F}_{\text{GenZero}}$  to obtain  $\llbracket z \rrbracket_D$ .
2 for each server from  $S_1, \dots, S_n$  do
3   for  $T_1, T_2 \in \mathcal{T}_i$  do
4     | Compute  $o_{T_1 T_2}^i = a_{T_1}^i b_{T_2}^i + z_{T_1 T_2}$ .
5   return  $\{o_{T_1 T_2}^i\}_{T_1, T_2 \in \mathcal{T}_i}$  to receiver  $S_R$ .
6  $S_j$  proceeds once it receives all messages.
7 for  $T_1, T_2 \in \mathcal{T}$  do
8   for  $i \in \mathcal{S}_{T_1, T_2}$  do
9     | if  $o_{T_1, T_2}$  is not set then
10      | Set  $o_{T_1, T_2} = o_{T_1, T_2}^i$ .
11     | else
12      | if  $o_{T_1, T_2}^i \neq o_{T_1, T_2}$  then
13      |   Outputs  $(v, \text{reject})$  and aborts.
14 Outputs  $(v = \sum_{T_1, T_2 \in \mathcal{T}} o_{T_1, T_2}, \text{accept})$ .
```

corrupt t' servers to launch an additive attack during the multiply-and-open operations among the servers.

Thus, in Fig. 3 we are able to define the desired ideal functionality $\mathcal{F}_{\text{dOPRF}}$ in the $(\mathcal{F}_{\text{GenZero}}, \mathcal{F}_{\text{GenRand}})$ -hybrid model. $\mathcal{F}_{\text{dOPRF}}$ interacts with the client and all n servers in the online phase. It serves as an ideal functionality that inputs $x \in \mathbb{F}_p$ or $\perp$ from the client, honest RSS shares of k, s^2 from $\mathcal{S}_{\text{Hon}}$ and z from $\mathcal{A}$, and outputs to the client a computation result $(\alpha \in \{r \leftarrow \mathbb{F}_p, (x+k)s^2+d\}, \text{status} \in \{\text{accept}, \text{reject}\})$.

With the definition of the ideal functionality in Fig. 3, it is clear that when $\mathcal{A}$ inputs $\perp$ (modeling inconsistent RSS in real world) to $\mathcal{F}_{\text{dOPRF}}$, the adversary learns only a random element $r \in \mathbb{F}_p$ after computation, thus gains no information of k or s^2 . On the other hand, for $\mathcal{A}$ that controls t' servers, he cannot force an honest client to accept a computation result $c = (x+k)s^2+d$ when $d \neq 0$.

5.2 Malicious Legendre dOPRF protocol

A straightforward approach to implementing $\mathcal{F}_{\text{dOPRF}}$ involves first verifying the consistency of the client's inputs among $\mathcal{S}_{\text{Hon}}$. This verification process requires each participating server to confirm with every other server that holds the same RSS share. If any inconsistent RSS share of x is detected, the servers collectively output $[r]$, generated by $\mathcal{F}_{\text{GenRand}}$, and then issue a **reject** signal. Otherwise, $\mathcal{F}_{\text{MulOpen}}$ is invoked to compute $(x+k)s^2$. However, even realizing $\mathcal{F}_{\text{MulOpen}}$ with our communication optimized protocol 1, this method of verifying client inputs leads to still three communication rounds.

Players: Server S_i ; $i = 1, \dots, n$, the client and $\mathcal{F}_{\text{dOPRF}}$.
Input: Client inputs $x \in \mathbb{F}_p$ or $\perp$ to $\mathcal{F}_{\text{dOPRF}}$. Denotes s^2 as b , $\mathcal{S}_{\text{Hon}}$ inputs their shares $(k_T, b_T)_{T \in \mathcal{T}_i}$ to $\mathcal{F}_{\text{dOPRF}}$, $\mathcal{A}$ gives input $d \in \mathbb{F}_p$ to $\mathcal{F}_{\text{dOPRF}}$.
Computation: If the client inputs $x \in \mathbb{F}_p$, $\mathcal{F}_{\text{dOPRF}}$ reconstructs k, b from received RSS shares of $\mathcal{S}_{\text{Hon}}$, computes $c = (x + k)b + d$.
Output: If $x \in \mathbb{F}_p$, when $d = 0$, $\mathcal{F}_{\text{dOPRF}}$ outputs (c, accept) to client and (c, reject) otherwise. When $x = \perp$, set $c = r$ where r is uniformly randomly selected from $\mathbb{F}_p$, $\mathcal{F}_{\text{dOPRF}}$ outputs (r, reject) to the client.

Fig. 3: Ideal functionality $\mathcal{F}_{\text{dOPRF}}$ up to additive attack.

In this subsection, we adapt protocol 1 to protocol 2 that realize $\mathcal{F}_{\text{dOPRF}}$ in two rounds, which is also denoted as Π_{dOPRF} throughout. To check potential inconsistent RSS of x submitted to $\mathcal{S}_{\text{Hon}}$, we introduce a small change in protocol 2: for every $T_1 T_2 \in \mathcal{T}$, we have each server $i \in \mathcal{S}_{T_1 T_2}$ send $v_{T_1 T_2}^i = o_{T_1 T_2}^i + (x_{T_1}^i + r_{T_1}^i) \hat{z}_{T_1 T_2}^i$ to the client instead of $o_{T_1 T_2}^i$ as in protocol 1, where $[r]$ is a random ASS among all servers and $\hat{z}_{T_1 T_2}$ is an ASS shared among $\mathcal{S}_{T_1 T_2}$. With this change, only if all honest parties receive consistent shares of x can the client reconstruct $o_{T_1 T_2}$. Otherwise, the client receives random messages and learns nothing of k or s^2 . In other words, if the client acts maliciously and sends an invalid RSS of x to $\mathcal{S}_{\text{Hon}}$, the client will receive a random output, preventing the adversary from deducing any useful information. However, above change in protocol 2 breaks the verifiability by an honest client against additive attack. To remedy this loss, we have each server send additionally one single hash output h_i computed over $\{o_{T_1 T_2}^i\}_{T_1, T_2 \in \mathcal{T}_i}$ to the client. After an honest client reconstructs $\{o_{T_1 T_2}\}_{T_1, T_2 \in \mathcal{T}}$, for each $i \in [n]$ he recomputes his own hash output over $\{o_{T_1 T_2}\}_{T_1, T_2 \in \mathcal{T}_i}$ and compares it to h_i , he accepts $\{o_{T_1 T_2}\}_{T_1, T_2 \in \mathcal{T}}$ only if all checks pass. Here, altogether servers send $n \left(\binom{n-1}{t}^2 + 1 \right)$ field elements to the client. For the detailed security proof of protocol 2, please refer it in Appendix B.

Client Verifiability Consider a scenario where an honest client interacts with up to t servers that may be controlled by an adversary $\mathcal{A}$. In this setting, we assert the verification algorithm described in the reconstruction stage of protocol 2 is an efficient verification algorithm such that:

- If client inputs inconsistent RSS shares of x to $\mathcal{S}_{\text{Hon}}$, client receives (r, reject) ;
- Otherwise, if $\mathcal{S}_{\text{Mal}}$ inputs $d \in \mathbb{F}_p$ in the additive attack, client receives

$$c, \text{status} = \begin{cases} (x + k)s^2 + d, \text{reject} & \text{if } d \neq 0, \\ (x + k)s^2, \text{accept} & \text{otherwise.} \end{cases}$$

This verifiability ensures that an honest client can reliably verify whether the protocol execution was free from disruptions, in the presence of up to t corrupted servers.

Protocol 2 Legendre dOPRF in the $(\mathcal{F}_{\text{GenZero}}, \mathcal{F}_{\text{GenRand}})$ -hybrid model
 (Π_{dOPRF})

Players: Server S_i for $i = 1, \dots, n$ and a client.

Input : $x \in \mathbb{F}_p$ by client, $\llbracket s^2 \rrbracket_R$ and $\llbracket k \rrbracket_R$ by servers.

Output: (c, status) .

- 1 Client computes a RSS of $x = \sum_{T \in \mathcal{T}} x_T$, then sends $(x_T)_{T \in \mathcal{T}_i}$ to server S_i for each $i \in \{1, \dots, n\}$.
 - 2 Servers invoke $\mathcal{F}_{\text{GenZero}}$ to obtain $\llbracket z \rrbracket_D, [\hat{z}_{T_1 T_2}]$ among $\mathcal{S}_{T_1 T_2}$ for each $T_1, T_2 \in \mathcal{T}$; additionally, they invoke $\mathcal{F}_{\text{GenRand}}$ to obtain $\llbracket r \rrbracket_R$.
 - 3 Let $\llbracket a \rrbracket_R = \llbracket x + k \rrbracket_R, \llbracket b \rrbracket_R = \llbracket s^2 \rrbracket_R$.
 - 4 **for** each server i from $S_1, \dots, S_n$ **do**
 - 5 Set $H = \epsilon$.
 - 6 **for** $T_1, T_2 \in \mathcal{T}_i$ **do**
 - 7 Compute $o_{T_1 T_2}^i = a_{T_1}^i b_{T_2}^i + z_{T_1 T_2}$.
 - 8 Update $H = H \parallel o_{T_1 T_2}^i$.
 - 9 Compute $v_{T_1 T_2}^i = o_{T_1 T_2}^i + (x_{T_1}^i + r_{T_1}^i) \hat{z}_{T_1 T_2}^i$.
 - 10 Compute $h_i = \text{Hash}(H)$.
 - 11 **return** $(\{v_{T_1 T_2}^i\}_{T_1, T_2 \in \mathcal{T}_i}, h_i)$ to client.
 - 12 Once the client receives all messages, it proceeds as follows.
 - 13 Set $v = 0$.
 - 14 **for** $T_1, T_2 \in \mathcal{T}$ **do**
 - 15 Computes $v_{T_1, T_2} = \frac{1}{\#\mathcal{S}_{T_1, T_2}} \sum_{i \in \mathcal{S}_{T_1, T_2}} v_{T_1, T_2}^i$.
 - 16 Accumulates $v = v + v_{T_1, T_2}$.
 - 17 **for** $i = 1, \dots, n$ **do**
 - 18 Set $H = \epsilon$.
 - 19 **for** $T_1, T_2 \in \mathcal{T}_i$ **do**
 - 20 Update $H = H \parallel v_{T_1 T_2}$.
 - 21 If $h_i \neq \text{Hash}(H)$, outputs (v, reject) .
 - 22 Outputs (v, accept) .
-

5.3 Full security of the Legendre PRF

Theorem 2 establish that securely evaluating a single bit of the Legendre PRF in an oblivious manner is guaranteed, assuming up to $t < n/3$ malicious servers. This ensures that either the protocol runs honestly, or if there is a malicious party, no one gains any information about the secret values. As a result, the oblivious Legendre PRF effectively simulates an idealized Legendre PRF oracle $\mathcal{F}_{\text{dOPRF}}$. The security of the PRF, where an attacker has access to the Legendre PRF oracle $\mathcal{F}_{\text{dOPRF}}$, is based on the DSLSP Assumption. The most effective known attacks against the DSLSP Assumption are key-extraction attacks [24, 34, 35]. Given that an attacker can query the PRF M times, the classical attack requires $O(M^2 \log p)$ memory and has a runtime of $O(\frac{p \log p \log \log p}{M^2})$, while the quantum attack uses $\tilde{O}(M^2)$ classical memory with quantum random access and has a runtime of $\tilde{O}(\sqrt{\frac{p}{M^2}})$. Under these conditions, the protocol offers $\lambda_c = \lambda$ bits

of classical security and $\lambda_q = 2\lambda/3$ bits of quantum security when the underlying prime p is of size 2λ .

On partial obliviousness. A partially oblivious PRF (pOPRF) is an OPRF whose input is divided into a private and a public part $x = (x_{\text{pub}}, x_{\text{priv}})$. There are generic transformations ([13], Section 3.1) to construct a pOPRF from an OPRF, which can also be employed in our construction. Let $\{F_k\}$ be an OPRF, let $\{G_k\}$ be a PRF. Then, the function $F'_k(x_{\text{pub}}, x_{\text{priv}}) := F_{G_k(x_{\text{pub}})}(x_{\text{priv}})$ is a pOPRF. Given a fixed RSS secret sharing of a key $k = \sum_{T \in \mathcal{T}} k_T$, we define $k' = \sum_{T \in \mathcal{T}} G_{k_T}(x_{\text{pub}})$ to be an updated OPRF key, which we can also denote as $G'(k) = k'$. Here G' is a PRF function, which is not only a function on k , but on the randomness involved in splitting k into RSS shares. The new key k' can be computed from the public PRF G , the local shares k_T and the public part of the input x_{pub} . The partially oblivious Legendre PRF is defined to be the Legendre OPRF computed with the updated key k' , both in the semi-honest and malicious setting.

On adaptive adversaries. An adaptive adversary is an adversarial model in which the set of corrupted servers can change over time. However, the adversary stays constrained to control at most t corrupted servers at a time. Our protocol can be made adaptively secure by updating all long term secret data at regular time intervals. These include the key $\llbracket k \rrbracket_R$, as well as random secret sharing generated from $\mathcal{F}_{\text{GenZero}}$ and $\mathcal{F}_{\text{GenRand}}$.

6 Evaluation

We first conduct an asymptotic comparison of our protocol Π_{dOPRF} with state-of-the-art MPC protocols for computing the Legendre symbol. Subsequently, we implemented Π_{dOPRF} in C++ and compared it to other OPRF solutions, in our implementation we use the BLAKE3 [37] as a hash function and also as a PRF for generating randomness. Our code is available at GitHub³⁴. All our tests are on a single server with 32GB RAM, 12th Gen Intel(R) Core(TM) i7-12700K CPU model that ran Ubuntu 22.04 LTS. Finally, in the same context, we performed local benchmarking on all sub-procedures of Π_{dOPRF} . Due to page limitations, the results are not included here; interested readers are referred to the full version [33].

6.1 Comparison to State-of-the-art MPC

We compare our protocol Π_{dOPRF} to two state-of-the-art MPC solutions proposed by Dalskov et al. [15] and Chida et al. [14] and present our results in Table 2. Specifically, we consider the same but practical outsourced computation model with one client and a few servers, where one or two servers are corrupted. Note that we adapt the computation of $\mathcal{C}_{\text{PRF}}$ to fit our desired outsourced computation model for [14, 15].

³ <https://github.com/nkKolja/Legendre-dOPRF>

⁴ <https://github.com/nann-cheng/d-OPRF>

Table 2: Concrete efficiency comparison to State-of-the-art MPC.

Reference	Scheme	Comm.	No Inter-Server Comm.	Rounds
Our Π_{dOPRF}	RSS-(1,4)	52	✓	2
Dalskov19 [15]		36	✗	3
Chida23 [14]	RSS-(1,3)	201	✗	9
Our Π_{dOPRF}	RSS-(2,7)	1687	✓	2
Chida23 [14]		745	✗	9
Chida23 [14]		445	✗	9

Table 3: Comparison between our work and state-of-the-art OPRF solutions. Superscript * indicates estimated numbers adapted to our WAN setting.

Protocol	λ	Primitive	Comm.[KB]		Rounds	Runtime [ms]	
			Offline	Online		Offline	Online
PESTO [10]	128	Pairing, ECDSA	-	-	2	-	260*
GC-OPRF [21]		GC, AES-128	-	4634	4	-	1842
Legendre-OPRF [11]		OT, ZK	-	911	9	-	1049*
Our Π_{dOPRF}		RSS-(1,4)	432	144	2	204	215
Our Π_{dOPRF}		RSS-(2,7)	93900	6303	2	1076	815
GC-OPRF [21]	256	GC, AES-256	-	6628	4	-	2320
Our Π_{dOPRF}		RSS-(1,4)	1728	579	2	208	637

In Chida et al. [14], the authors propose a general MPC framework designed for settings with an honest majority in the presence of $t < n/2$ malicious adversaries. This allows more room for adversarial behaviours than our work, which allows less corruptions with $t < n/3$. Their framework is modular and applicable to any arbitrary arithmetic circuit, and it consists four stages: *Input Sharing*, *Circuit Emulation*, *Verification*, and *Output Reconstruction*. However, their solution is not as efficient when applied to our proposed dOPRF. More precisely, for computing the circuit $\mathcal{C}_{\text{PRF}}$, regardless of the underlying secret sharing scheme used, the framework requires 7 invocations of $\mathcal{F}_{\text{rand}}$, 6 invocations of $\mathcal{F}_{\text{mul}}$, and 8 invocations of $\mathcal{F}_{\text{open}}$ over 9 rounds. If using RSS-(t, n) where $(t, n) = (1, 3)$, assuming the underlying semi-honest sub-protocol $\mathcal{F}_{\text{mul}}$ is instantiated by a naive protocol consuming 24 elements in one round, $\mathcal{F}_{\text{open}}$ instantiated with 6 elements, this results in a total communication volume of 201 elements. When $(t, n) = (2, 5)$, assuming the underlying semi-honest sub-protocol $\mathcal{F}_{\text{mul}}$ is instantiated by the protocol in [17], which requires opening two masked elements among all parties, the communication volume would total 745 elements. Alternatively, if using Shamir Secret Sharing in the setting $(t, n) = (2, 5)$, communication-wise, the realization of $\mathcal{F}_{\text{mul}}$ requires 20 elements and $\mathcal{F}_{\text{open}}$ also requires 20 elements, resulting a total communication of 445 elements.

In the work of Dalskov et al. [15], they consider a setting of four parties where one party might be an active adversary, the authors propose a novel method performing multiplication as well as identifying cheating. This method incurs a total of 12 elements exchanged for each multiplication operation. Including the

communication cost for the input and output stages, the total communication volume amounts to 36 elements over 3 rounds.

In Tab. 2, with one corrupt server, we compared our protocol to the solution built from Fantastic Four [15] that used both RSS-(1,4) and the solution from [14] built from RSS-(1,3); with two corrupt servers, we compared our protocol instantiated from RSS-(2,7) to the solution from [14] built from two secret sharing schemes, *i.e.* RSS-(2,5) and SSS-(2,5). As clearly shown in Tab. 2, only our work Π_{dOPRF} offers the advantage of no inter-server communication. In the setting of one corrupt server, since computation and communication are less dominant than communication delay, our protocol Π_{dOPRF} exhibits a clear advantage over the state-of-the-art. With two corrupt servers, our protocol Π_{dOPRF} requires roughly double or triple the communication volume than in [14], regardless of this higher volume cost, we argue with much fewer rounds required in our work: 2 rounds compared to 9 rounds, our protocol Π_{dOPRF} would be more efficient than [14] in this application scenario.

6.2 Comparison to state-of-the-art OPRF

In Tab. 3, we compare our protocol against state-of-the-art works in terms of computation time and communication volume, covering both the offline and online phases for an output of 128 bits. All evaluations were performed in a simulated WAN environment (50 Mbits bandwidth and 100 ms delay) on a single machine. The compared works fall into two categories: two-party post-quantum OPRF protocols [11, 21] and distributed classical OPRF protocols [10].

For each protocol, λ indicates the security level, *Comm* represents the total communication volume, *Rounds* denotes the number of sequential communication rounds in the online phase, and *Runtime* measures evaluation time at various stages. In the offline phase, *Runtime* reflects the total time taken by all parties to complete their tasks. In the online phase, we measure the time from the client's request initiation to obtaining the final OPRF/dOPRF result. The results for [10] are estimates based on figures from Tab. 1 of [10] (2 servers) adjusted for our WAN setting (200 ms RTT), while the numbers for [11] are sourced from their estimations in Section 5.2 under our WAN conditions. For GC-OPRF [21], we executed their code in the simulated WAN setting to obtain corresponding metrics. Notably, for our protocol Π_{dOPRF} , we revised the results to reflect a real distributed server setup, as it is expected to perform better with distributed computations.

Among all protocols that achieve $\lambda = 128$ bit security, our RSS-(1,4) Π_{dOPRF} instantiation protocol demonstrated the best computational and communication efficiency. Under $\lambda = 256$, our RSS-(1,4) Π_{dOPRF} protocol also outperforms the GC-OPRF protocol from [21].

Acknowledgments. This work was partially supported by the Stepping Stone Grant "Privacy-preserving & Distributed Authentication" funded by the University of St. Gallen.

References

1. Secret sharing schemes realizing general access structures. In *Proc. IEEE Global Telecommunication Conf., Globecom 87*, Lecture Notes in Computer Science, 1987.
2. Unapproved IEEE Draft Standard for Specifications for Password Based Public Key Cryptographic Techniques. *IEEE Unapproved Std P1363.2 /D27*, 2007.
3. ISO: Information technology — security techniques — key management — part 4: Mechanisms based on weak secrets. ISO/IEC, International Organization for Standardization, 2017. <https://www.iso.org/standard/67933.html>16.
4. Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. Pasta: Password-based threshold authentication. Cryptology ePrint Archive, Paper 2018/885, 2018.
5. Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and fhe. Cryptology ePrint Archive, Paper 2023/232, 2023.
6. Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. Cryptology ePrint Archive, Paper 2019/1271, 2019.
7. Alessandro Baccarini, Marina Blanton, and Chen Yuan. Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. Cryptology ePrint Archive, Paper 2020/1577, 2020.
8. Andrea Basso. A post-quantum round-optimal oblivious prf from isogenies. Cryptology ePrint Archive, Paper 2023/225, 2023.
9. Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. Cryptanalysis of an oblivious prf from supersingular isogenies. In *Advances in Cryptology – ASIACRYPT 2021*.
10. Carsten Baum, Tore K. Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. Pesto: Proactively secure distributed single sign-on, or how to trust a hacked server. Cryptology ePrint Archive, Paper 2019/1470, 2019.
11. Ward Beullens, Lucas Dodgson, Sebastian Faller, and Julia Hesse. The 2hash oprf framework and efficient post-quantum instantiations. Cryptology ePrint Archive, Paper 2024/450, 2024.
12. Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. Cryptology ePrint Archive, Paper 2020/1532, 2020.
13. Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. Cryptology ePrint Archive, Paper 2022/302, 2022.
14. Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority mpc for malicious adversaries. *Journal of Cryptology*, 36(3):15, 2023.
15. Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: {Honest-Majority}{Four-Party} secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2183–2200, 2021.
16. Ivan Damgård. On the randomness of Legendre and Jacobi sequences. In *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO ’88, pages 163–172, London, UK, UK, 1990. Springer-Verlag.
17. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multi-party computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.
18. Poulami Das, Julia Hesse, and Anja Lehmann. Dpase: Distributed password-authenticated symmetric encryption. Cryptology ePrint Archive, Paper 2020/1443, 2020.

19. Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. Mpc-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*.
20. Lucas Dodgson, 2023. https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/Master_Thesis_Post_Quantum_Building_blocks_for_secure_computation.pdf.
21. Sebastian Faller, Astrid Ottenhues, and Johannes Ottenhues. Composable oblivious pseudo-random functions via garbled circuits. Cryptology ePrint Archive, Paper 2023/1176, 2023.
22. Dankard Feist. Legendre pseudo-random function, 2019. <https://legendreprf.org/bounties>.
23. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography*, pages 303–324, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
24. Paul Frixons and André Schrottenloher. Quantum security of the legendre prf. Cryptology ePrint Archive, Paper 2021/149, 2021.
25. Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, pages 430–443, New York, NY, USA, 2016. ACM.
26. Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.
27. Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. Cryptology ePrint Archive, Paper 2009/045, 2009.
28. Stanisław Jarecki, Ali Bagherzandi, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. Cryptology ePrint Archive, Paper 2010/561, 2010.
29. Stanisław Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. Cryptology ePrint Archive, Paper 2014/650, 2014.
30. Stanisław Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Toppss: Cost-minimal password-protected secret sharing based on threshold oprf. Cryptology ePrint Archive, Paper 2017/363, 2017.
31. Stanisław Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. 2009.
32. Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. 2010.
33. Novak Kaluderovic, Nan Cheng, and Katerina Mitrokotsa. A post-quantum distributed oprf from the legendre prf. *Cryptology ePrint Archive*, 2024.
34. Novak Kaluderović, Thorsten Kleinjung, and Dušan Kostić. Cryptanalysis of the generalised Legendre pseudorandom function. volume 4, pages 267–282, 2020.
35. Novak Kaluderović, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre prf. Cryptology ePrint Archive, Paper 2020/098, 2020.
36. Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154:370–381, 2006.
37. Jack O’Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O’Hearn. Blake3, one function fast everywhere, 2020.

Players: Server $\mathcal{S}_1, \dots, \mathcal{S}_n$ and the functionality $\mathcal{F}_{\text{Mul}}$.
Input: Every server $S_i \in \mathcal{S}_{\text{Hon}}$ inputs their shares $(a_T, b_T)_{T \in \mathcal{T}_i}$ to $\mathcal{F}_{\text{Mul}}$, while each malicious server inputs whatever input to $\mathcal{F}_{\text{Mul}}$.
Computation: $\mathcal{F}_{\text{Mul}}$ checks if every RSS share submitted by each $S_i \in \mathcal{S}_{\text{Mal}}$ is equal to the one corresponds to the one hold by $\mathcal{S}_{\text{Hon}}$, it outputs $\perp$ if there is any mismatch; otherwise $\mathcal{F}_{\text{Mul}}$ reconstructs a, b from RSS shares sent by $\mathcal{S}_{\text{Hon}}$ and computes $c = ab$, subsequently a random RSS of c as $\llbracket c \rrbracket_R$.
Output: $\mathcal{F}_{\text{Mul}}$ outputs corresponding RSS shares of $\llbracket c \rrbracket_R$ to all servers.

Fig. 4: Ideal functionality $\mathcal{F}_{\text{Mul}}$ - RSS Multiplication with abort.

A RSS to RSS Multiplication

We define the ideal functionality of $\mathcal{F}_{\text{Mul}}$ in Fig. 4. Maurer [36] in Protocol 6.2 provides maliciously secure multiplication in a similar fashion. They check for malicious behaviour by comparing the received shares and then opening the difference of all shares to confirm that parties indeed computed secret shares of the same product term $a_{T_1} b_{T_2}$. However, this approach takes three communication rounds and is communication costly.

We opt for a more efficient approach where the parties are assumed to hold a pre-shared DRSS random seed $\llbracket s \rrbracket_D$, and they use their seed share $s_{T_1 T_2}$ to compute the RSS sharing of each product term $\llbracket a_{T_1} b_{T_2} \rrbracket_R$. This way the parties can check for malicious behaviour without the need for additional communication. While this approach reduces the security to that of an underlying PRF function used for randomness generation, this trade-off is fine as the remainder of our protocol also relies on computational security (as opposed to information-theoretic in [36]). Specifically, assume that the servers hold a common seed distributed as a DRSS. In other words let $\llbracket s \rrbracket_D = (s_{T_1 T_2})_{T_1, T_2 \in \mathcal{T}}$ be held among the servers. Server i computes $c^{T_1 T_2} := a_{T_1} b_{T_2}$ for each $T_1, T_2 \in \mathcal{T}_i$, and then computes a secret sharing of c_i as $(c_T^{T_1 T_2})_{T \in \mathcal{T}} \leftarrow \text{RSS}(c^{T_1 T_2}; s_{T_1 T_2})$. Each server distributes the RSS shares they computed to all other servers. Upon receiving all shares, the servers verify that the shares received from different parties corresponding to the same shares T_1, T_2, T are all equal, if not they abort, otherwise they compute the RSS of the product as $(ab)_T = \sum_{j=1}^n c_{j,T}$. Due to page limit, a detailed description of the protocol in the malicious adversarial setting with $t < n/3$, along with its associated proof, can be found in the full version of this work [33].

B The Security of Π_{dOPRF}

Theorem 2. *Protocol 2 securely computes $\mathcal{F}_{\text{dOPRF}}$ in the $(\mathcal{F}_{\text{GenZero}}, \mathcal{F}_{\text{GenRand}})$ -hybrid model with adversary $\mathcal{A}$ controls $t' \leq t < n/3$ servers and the client.*

Proof. We need to prove there exists a PPT simulator Sim , it inputs only the received output (c, status) by the client in the ideal functionality $\mathcal{F}_{\text{dOPRF}}$, such that it's able to output a transcript whose distribution is indistinguishable to the one received by the client in protocol 2. Specifically, the goal is to simulate $v_{T_1 T_2}^i$ for every $i \in S_{\text{Hon}}, T_1, T_2 \in \hat{\mathcal{T}}$ and $\{h_i\}_{i \in S_{\text{Hon}}}$. w.l.o.g, assume $t' = t$, which indicates $|S_{\text{Hon}}| = n - t$, since each honest server holds RSS share index $\hat{T}$ and thus he holds $2\binom{t}{n-1}$ terms whose DRSS share indices are within $\hat{\mathcal{T}}$. Thus, there are in total $I = (n - t)(2\binom{t}{n-1})$ terms sent from S_{Hon} corresponds to $\hat{\mathcal{T}}$.

In the case when the client submits consistent RSS of x to S_{Hon} , to simulate their distribution in this case, let Sim draws randomly field elements $\{r_i\}_{i \in [I-1]}$ from $\mathbb{F}_p$, where $\bar{r}$ is denoted as their sum. Denote $\bar{v} = c - d - \sum_{i \in [I-1]} r_i - \bar{r}$, the list $L = \{r_1, \dots, r_{I-1}, \bar{v}\}$ of size I is exactly the generated simulation of $v_{T_1 T_2}^i$ for every $i \in S_{\text{Hon}}, T_1, T_2 \in \hat{\mathcal{T}}$, as the distribution of L is identical to what the client receives in the real world. Then, from L , the simulator is able to compute $v_{T_1 T_2}$ for every $T_1 T_2 \in \hat{\mathcal{T}}$ and hence perfectly computes $\hat{h}_i$ for every $i \in S_{\text{Hon}}$.

In contrast, when the client provides inconsistent RSS share of x to S_{Hon} , we analyze the message distribution in the real world. w.l.o.g, we assume that the client submits a valid $\llbracket x \rrbracket_R$ except at share index T_* . This means there are inconsistent values for the share x_{T_*} held by S_{T_*} . Formally, we define $\bar{x}_{T_*}^i = e^i + x_{T_*}$, where there exist $i, j \in S_{T_*}$ such that $e^i \neq e^j$. Here, e^i denotes the error in $\bar{x}_{T_*}^i$. Thus, $\bar{x}_{T_*}^i$ represents a potentially invalid share held by server S_i with share index T_* . Denote $\bar{\mathcal{T}} = \{T_* T_1, \dots, T_* T_{\max}\}$, for every $T_1 T_2 \in \bar{\mathcal{T}}$, observe that in protocol 2 the client reconstructs

$$v_{T_1 T_2} = \frac{1}{\#S_{T_1 T_2}} \left(\sum_{i \in S_{T_1 T_2}} a_{T_1}^i b_{T_2}^i \right) + e_{T_1 T_2}$$

where $e_{T_1 T_2} = \frac{1}{\#S_{T_1 T_2}} \sum_{i \in S_{T_1 T_2}} e_{T_1}^i \hat{z}_{T_1 T_2}^i$ is a linear combination result from random numbers that obeys a uniform distribution $\mathcal{U}$, thus the final reconstructed output $v_{T_1 T_2}$ is independent of secret a or b . Finally, the client recovers $c = V + E$, where V is related to k and s^2 and $E = \sum_{T_1 T_2 \in \bar{\mathcal{T}}} e_{T_1 T_2}$ is a random number.

From above analysis, we present the simulator construction in the following, where the simulated view is indistinguishable to that in the execution of protocol 2. To simulate the transcript in the real world from $\mathcal{F}_{\text{dOPRF}}$ output (r, reject) :

- Let Sim draw randomly I field elements from $\mathbb{F}_p$ as simulated view transcript for $v_{T_1 T_2}^i, T_1 T_2 \in \hat{\mathcal{T}}, i \in S_{\text{Hon}}$.
- For each $i \in S_{\text{Hon}}$, Let Sim draw a random element $\hat{h}_i$ from the output domain of Hash as simulated transcript of $\hat{h}_i$.

Finally, Sim is able to output $\{\{\hat{v}_i^{T_1 T_2}\}_{T_1, T_2 \in \hat{\mathcal{T}}}, \hat{h}_i\}_{i \in S_{\text{Hon}}}$. In the random oracle model, if we draw a random element $\hat{h}_i$ from the output domain of Hash , we argue that any PPT $\mathcal{A}$ cannot distinguish between $\hat{h}_i$ and the real execution output h_i . Thus, $\mathcal{A}$ simulates all transcripts sent from S_{Hon} that is computationally indistinguishable to what he receives in the real world.