A post-quantum Distributed OPRF from the Legendre PRF

Nan Cheng, Novak Kaluđerović, and Katerina Mitrokotsa

Cybersecurity and Applied Cryptography group, Universität St. Gallen, Switzerland
{nan.cheng,novak.kaluderovic, katerina.mitrokotsa}@unisg.ch

Abstract. A distributed OPRF allows a client to evaluate a pseudorandom function on an input chosen by the client using a distributed key shared among multiple servers. This primitive ensures that the servers learn nothing about the input nor the output, and the client learns nothing about the key. We present a post-quantum OPRF in a distributed server setting, which can be computed in a single round of communication between a client and the servers. The only server-to-server communication occurs during a precomputation phase. The algorithm is based on the Legendre PRF which can be computed from a single MPC multiplication among the servers. To this end we propose two MPC approaches to evaluate the Legendre PRF based on replicated and optimised secret sharing, respectively. Furthermore, we propose two methods that allows us to perform MPC multiplication in an efficient way that are of independent interest. By employing the latter, we are able to evaluate the Legendre OPRF in a fashion that is quantum secure, verifiable and secure against malicious adversaries under a threshold assumption, as well as computable in a single round of interaction. To the best of our knowledge, our proposed distributed OPRFs are the first post-quantum secure offering such properties. We also provide an implementation of our protocols, and benchmark it against existing OPRF constructions.

Keywords: OPRF · post-quantum · Legendre PRF · Distributed OPRF · MPC.

1 Introduction

An oblivious PRF (OPRF) is a two party 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. [FIPR05], and they found widespread application in numerous areas of cryptography. The inherent obliviousness of OPRFs facilitates the development of protocols that safeguard private data. A common design paradigm to leverage an OPRF is to let a client compute a high-entropy cryptographic object (*e.g.,* a token) from a low-entropy input (*e.g.,* a password). Furthermore, the fact that the computation is assisted by one or multiple servers allows for delegating computations to the servers and keeping a lightweight computation on the client side. The high-entropy cryptographic material can then be securely stored on the servers and does not need to be stored by the client, which only needs to remember a low-entropy value such as a password.

A variation of OPRFs are distributed OPRFs (dOPRFs), originally introduced by Jarecki et al. [JKKX17], which consider the setting where multiple servers are employed for the evaluation of the function. The secret key k is distributed among the servers and the servers and the client jointly evaluate the function $F_k(x)$ on the client's input x without revealing any information about x to the servers. Additionally, the distributed nature of the key provides additional protection from server breaches.

OPRFs have been demonstrated to be the fundamental building block for creating password-authenticated key exchange [Wu00, art07, art17], password secret sharing [JKK14, JBSL10, JKKX17, BFH$^+$19, DHL20], single sign-on protocols [BFH$^+$19, AMMM18], and private set intersection [HL09, KKRT16, JL09, JL10], among many other use cases. All existing practical OPRF constructions base their security on the difficulty of solving the discrete logarithm problem. This reliance makes such constructions vulnerable to attacks by quantum computers. Post-quantum OPRFs are an active area of research, and current constructions are either broken [BKW20, BKM$^+$], inefficient [ADDS19, Bas23], only presented in the semi-honest model [ADDG23, FOO23, DGH$^+$], or require many rounds of communication [BKW20, BDFH24, Dod23], and neither of them are distributed. Therefore, we address the following open question:

Is it possible to construct an efficient distributed OPRF that provides post-quantum security guarantees?

Our answer is affirmative and we introduce four different variants of distributed OPRFs that provide post-quantum security guarantees in different security models varying from semi-honest to fully malicious.

Contributions: Through the use of generic secure MPC techniques, any pseudorandom function can be computed in an oblivious fashion. This involves the client and the servers simulating a secure MPC protocol to compute $(x, k) \mapsto (F_k(x), \perp)$. However, these protocols have significant communication and computational overheads. We propose an alternative approach where the client shares their input secretly with the servers but does not participate in the PRF computation. The servers then perform the PRF computation in an MPC setting, as depicted in Figure 1.

We use the Legendre PRF as the PRF candidate due to its compatibility with MPC. In general it can be computed with just a few MPC multiplications. This, combined with the client-less MPC approach, allows us to compute the PRF in a single communication round between the client and the servers, eliminating the need for inter-server communication. This approach offers several benefits, including efficiency, reduced communication costs, and compatibility with classical OPRFs computed in the same communication model in a distributed setting.

We offer four different algorithms that provide security guarantees for different considered security models, ranging from the semi-honest to malicious. Two of these algorithms are based on replicated secret sharing (RSS), while the other two are based on optimised secret sharing (OSS), one for each security model. The cornerstone of our approach is a verifiable non-interactive maliciously secure MPC multiplication algorithm, which may be of independent interest.

Finally, we provide an implementation and benchmarks, comparing our algorithm against similar classical and post-quantum primitives.

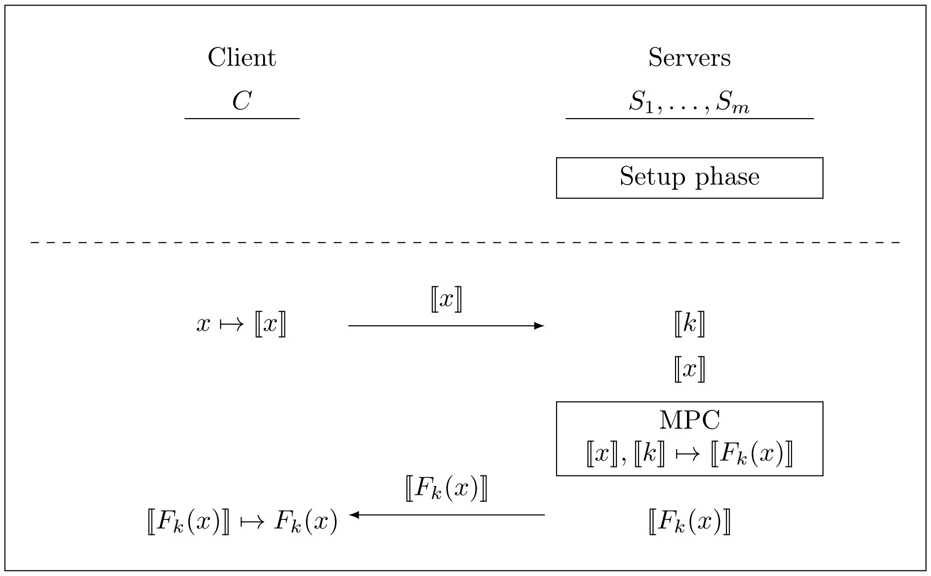

Fig. 1: MPC based distributed OPRF

Adversarial model: We provide different algorithms that are secure in the presence of diverse adversarial behaviours. We call an adversary *semi-honest* if they follow the protocol specification, but may try to learn more information than allowed from the protocol transcript. An adversary is called *malicious* if they have the full freedom to run any strategy available to them. The adversary is assumed to be probabilistic polynomial-time.

The main parties in the protocol is the client and n servers. We denote by t the threshold on the number of corrupted *servers*. The client may or may not be corrupted irrespective of t. We provide different protocols that are provably secure when $t < n/2$, $t < n/3$ and $t < n$ respectively. The non-corrupted servers are *honest*, meaning that they follow the protocol and do not collude with any corrupted parties.

Our algorithms provide *output verification*, meaning that the client can be assured of the output's correctness. We also introduce the concept of *input verification*, which is crucial in the new MPC setting. This protects the honest parties from the adversary in the event of malicious client behavior. Specifically, it ensures that the servers do not leak any sensitive data when the client behaves maliciously. In such cases, the client is given a random output.

We assume that the communication between the client and the servers, as well as the inter-server communication during the precomputation stage, occurs over a secure channel.

Lastly, the adversary is assumed to be static, meaning that the set of corrupted parties is determined at the beginning of the protocol.

Technical framework: The Legendre PRF can be computed easily with field arithmetic. Computing the Legendre PRF function $F_k(x)$ is information-theoretically equivalent to revealing a field value $(x+k)\cdot s^2$ where s is a random non-zero field element unknown to both parties. As such, the computation of the Legendre PRF boils down to performing one addition, one squaring, and one multiplication.

Our first two protocols employ replicated secret sharing (RSS) to compute this function. Each field element x, k, s is stored as an RSS share among the servers. Given the scheme's linearity, the computation of addition is inherently straightforward. For the remaining operations, we provide two multiplication algorithms: one to compute the product and another to compute the square.

In Section 3 we introduce *doubly replicated secret sharing* (DRSS), a secret sharing scheme which realises the same access structure as RSS, but acts as a useful tool for computing multiplications.

The product is computed with an online multiplication algorithm RSS, RSS $\mapsto$ DRSS by relying only on local operations. The DRSS output, together with some auxiliary data is shared with the client who then verifies the correctness of the computation. This procedure is presented in 3.1. The computation of the square is facilitated by a semi-honest RSS, RSS $\mapsto$ RSS multiplication algorithm, which uses the previous multiplication algorithm in a second stage to verify the correctness of the semi-honest protocol. This algorithm is provided in the Appendix A. Despite requiring a round of interaction, this algorithm does not create an overhead in the online phase since it is only used offline.

Both multiplication algorithms provide post-quantum computational security as the verification step relies on the use of a hash function. However, the secrecy itself is information-theoretic. On top of this, we build two algorithms that compute the Legendre OPRF in a semi-honest (Section 4.1) and malicious (Section 4.2) setting respectively. The latter two protocols use optimised secret sharing (OSS) and authenticated optimised secret sharing (AOSS) to compute the Legendre PRF in the semi-honest and malicious setting respectively. The algorithms are introduced in Sections 4.3 and 4.4.

Finally, in Section 5 we benchmark the first two protocols. We share our implementation and provide the run-time of the protocols.

Protocol	Threshold	Input	Output	Communication rounds
Section 4.1	$t < n/2$	RSS	ASS	1
Section 4.2	$t < n/3$	RSS	DRSS	1
Section 4.3	$t < n$	OSS	OSS	1+1
Section 4.4	$t < n$	AOSS	AOSS	1+1

Fig. 2: Comparison of our proposed dOPRFs.

Related work: Oblivious PRFs have been systematically studied by Casacuberta et al. [CHL22]. 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 Table 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). *Verifiabilty* allows the client to be convinced of the correctness of the received output. In our case, the verifiability stems from the threshold assumption of a (super)majority of honest servers. A *distributed* OPRF works in a multi-server setting (i.e., the secret key is distributed to multiple servers), and a *threshold* OPRF can be computed requiring a threshold τ our of the total number n of servers.

OPRFs that do not rely on the previously mentioned primitives are scarce. One such example is a lattice-based construction proposed by Albrecht et al. [ADDS19]. While this construction offers verifiability, it is notably inefficient, requiring more than 128GB of bandwidth for each evaluation. There are also two existing constructions based on isogenies. The first, proposed by Boneh et al. [BKW20], is built on the CSIDH [CLM$^+$18] framework. The same paper proposed an additional protocol based on SIDH [FJP11], [JAC$^+$17], which was found to be vulnerable to specific attacks. Basso [Bas23] proposed a countermeasure and an enhanced algorithm to address these vulnerabilities. This improved construction also provides the advantage of verifiability.

Recently, Dodgson [Dod23] modeled and proved the security of the Legendre OPRF in the Universal Composability (UC) model. They focused on the single server case, leading to security assumptions that are different from those in our scenario. The author implemented and benchmarked the OPRF using the MP-SPDZ [Kel20] framework and the MASCOT [KOS16] protocol. However, it is important to note that the implementations provided by the author only assure security guarantees against classical and static adversaries, and they are designed for a semi-honest setting. Furthermore they require more than 96 rounds of interaction between the client and the server.

Recent work by Beullens et al. [BDFH24] provided a practical OPRF construction based on the Legendre PRF, together with a proof of security in the UC framework. Their construction relies on oblivious transfer and vector oblivious linear evaluation. This approach makes the protocol somewhat practical, requiring 1MB of total communication and an estimated evaluation time of 0.57 seconds on a WAN network. However, the protocol requires sending 9 messages between the server and client, with the communication itself being the bottleneck. At the moment no distributed version of said protocol exists and the authors pose it as an open challenge for future research.

Faller et al. [FOO23] show how to construct efficient, composable and quantum secure OPRFS from generic techniques. They use garbled circuits (GC) to construct and prove the security of an OPRF in the UC model with malicious users and semi-honest servers. In addition to that they implement and benchmark two versions of their protocol, which we compare with in Table 1.

OPRF	PRF	Partially oblivious	Verifiable	Distributed	Threshold	Quantum resistant
[HL09, JKK14]	NR	✗	✓	✗	✗	✗
[JL09]	DY	✗	✓	✗	✗	✗
[JBSL10, JKKX17, BFH$^+$19]	2HashDH	✓	✗	✓	✗	✗
[BKW20, ADDS19]	Isogenies	✗	✓	✗	✗	✓
[FOO23]	GC	✗	✗	✗	✗	✓
[Dod23, BDFH24]	Legendre PRF	✗	✓	✗	✗	✓
Our work	Legendre PRF	✓	✓*	✓	✓	✓

Table 1: Comparison of some state of the art OPRFs.

On distributed security: We emphasise 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.

While this may seem like a strong security relaxation, we argue that the benefits overweight 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 user's inputs. In general users' inputs are passwords (or hashes thereof) which are not assumed to be high-entropy, and this attack would reveal any such inputs. A quantum computer would speed these attacks even further by means of a faster search algorithm such as Grover's search [Gro96].

A second important property to note is that in our protocol the users' inputs would be revealed *only if the attacker takes control of the servers'* ***during the protocol***. The same is not true of the keys, which are long-term secrets stored on the servers, and which would be revealed to an attacker in any case of server breach. Even though our approach suffers from these drawbacks (which we denote with a yellow check-mark in Table 1), it allows us to make a trade-off and create a very efficient protocol with only a single round of communication and very fast evaluation times.

2 Preliminaries

In this paper, we describe the notation and recall some basic definitions that will be used in the rest of the paper.

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 $[\![x]\!]$ a generic secret sharing of a field element $x \in \mathbb{F}_p$, i.e., an n-tuple $(x_1, \ldots, x_n)$ such that each share x_i is owned by party i.

Multi-Party Computation: Secure multi-party computation (or MPC) allows two or more parties to compute any function on private inputs without revealing anything but the output of the function. We consider only arithmetic MPC protocols, where the parties hold (t, n) secret sharings of some field elements, and their goal is to compute secret sharings of arithmetic functions of these elements, while preserving all security properties under the same threshold t.

In our setting (Figure 1), in addition to the n computing parties, there is a single client C which provides a part of the input to the computing parties, and receives the output of the computation. We only concentrate on the protocols which can efficiently compute field additions and multiplications. More precisely, we develop MPC techniques which, given some secret sharings $[\![x]\!]$, $[\![y]\!]$ compute $[\![x \cdot y]\!]$ and reveal $x \cdot y$ to the client, while preserving security properties even with malicious clients, and without the need for any inter-server communication.

Oblivious PRFs: A *Pseudorandom Function* (PRF) family is a collection of functions $\{F_k\}_{k \in \mathcal{K}}$ with the same domain and codomain, 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 codomain.

An *Oblivious Pseudorandom Function* (OPRF) is a two-party protocol between a client and a server. It enables the server to evaluate a pseudorandom function on an input chosen by the client using the server's key. The protocol ensures that the server cannot learn anything about the client's input or output, while also preventing the client from learning anything about the server's key.

An OPRF is called *distributed* if there are multiple servers taking the role of the PRF evaluator. The key is assumed to be distributed between the servers by means of secret sharing. In this scenario, multiple servers evaluate a PRF in a multi-party computation setting, ensuring the confidentiality of the key and input/output values. A threshold (t,n) OPRF is a sub-type of a distributed OPRF where a threshold $t+1$ out of the total n servers are required to evaluate the OPRF on the client's input and can withstand t malicious servers.

A *Verifiable Oblivious Pseudorandom Function* (VOPRF) is an OPRF where the client can verify the correctness of the output they receive.

2.1 The Legendre PRF

The usage of Legendre symbols in a pseudorandom function is an idea originally proposed by Damgård [Dam90]. Due to its impractical nature, being orders of magnitude slower than generic PRF counterparts, it did not gain much attention until a result by Grassi et al. [GRR$^+$16] sparked an interest in the Legendre PRF [Fei19] because it was found suitable as a multi-party computation friendly PRF. This is mainly due to the homomorphic property of the Legendre symbol and the possibility of evaluating it with only three modular multi-party multiplications, which makes it a very efficient MPC friendly PRF candidate.

Definition 1 (Legendre symbol). *We define the Legendre symbol by setting*

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

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 codomain. We will assume that* $x \neq 0$ *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. In particular, we assume*

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

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* $F_k : \mathbb{F}_p \to \{-1, +1\}$ *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 $(k_1, \ldots, k_\lambda) \in \mathcal{K}^\lambda$ defining

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

Multi-party Legendre symbol computation: The Legendre symbol $\left(\frac{a}{p}\right)$ of a secret shared input $[\![a]\!]$ can be computed in an MPC setting as shown in Algorithm 1.

Algorithm 1 MPC Legendre symbol computation

```
1  Sample a random $[\![s]\!]$ and compute $[\![s^2]\!]$
2  Compute $[\![a]\!], [\![s^2]\!] \mapsto [\![as^2]\!]$ and reveal the outcome
3  Compute the Legendre symbol $\left(\frac{as^2}{p}\right) = \left(\frac{a}{p}\right)$
```

Note that revealing as^2 is equivalent to revealing $\left(\frac{a}{p}\right)$, since masking a with a random non-zero square gives us the same information as the Legendre symbol of a.

Multi-party Legendre PRF computation: Since the Legendre PRF is defined as $\left(\frac{x+k}{p}\right)$ where x is the client's input and k the servers' key. After the client shares their input as $[\![x]\!]$, the servers compute $[\![a]\!] = [\![x+k]\!]$ and reveal the value as^2 only to the client.

For a λ-bit output Protocol 1 is repeated for each bit of the output. In particular λ keys $k_1, \ldots, k_\lambda$ and λ random values $s_1, \ldots, s_\lambda$ are computed for each of the λ output bits. This procedure can be parallelised, and a high level overview presented in Protocol 2.

The actual protocol described in Section 4 differs from the high level overview, described in Algorithm 2, in Steps 4. and 6. The multiplication in Step 6. yields a secret sharing $[\![\cdot]\!]'$ of a different format then the input $[\![\cdot]\!]$. Both the semi-honest and the malicious algorithm employ different types of secret sharings for $[\![\cdot]\!]'$. Moreover, additional precomputed data (in particular secret shares of zero) is used in order to compute that multiplication in a privacy preserving manner, along with a hash computation. The result of this hash computation is provided to the client for verification. In Step 6, besides reconstructing the output, the client also carries out a verification step in the malicious algorithm.

The security of the Legendre PRF itself is not subject of this paper. This topic has been explored in other studies [vDH00, Kho19, BBUV19, KKK20a, KKK20b, SHB21, FS21]. The current consensus suggests that the Legendre PRF is presumed to be secure against quantum adversaries who have classical access to the PRF oracle. This assumption is consistent with our scenario.

Algorithm 2 Distributed oblivious Legendre PRF

```
Client Input  : $x \in \mathbb{F}_p$
Servers Input : $(\llbracket k_i \rrbracket, \llbracket s_i^2 \rrbracket)_{i=1,\dots,\lambda}$ pre-computed and secret shared,
                where $k_i, s_i \in \mathbb{F}_p$ for each $i = 1, \dots, \lambda$.
Output        : $(\left(\frac{x+k_1}{p}\right), \dots, \left(\frac{x+k_\lambda}{p}\right))$
1 Client: Share $\llbracket x \rrbracket$ to servers
2 Servers: Compute $\llbracket a \rrbracket = \llbracket x \rrbracket + \llbracket k_i \rrbracket$
3 Servers: Compute $\llbracket b \rrbracket = \llbracket s^2 \rrbracket$
4 Servers: Compute $\llbracket a \rrbracket, \llbracket b \rrbracket \mapsto \llbracket a \cdot b \rrbracket'$
5 Servers: Send shares of $\llbracket a \cdot b \rrbracket'$ to the client
6 Client: Reconstruct $((x + k_i)s_i^2)_{i=1,\dots,\lambda}$
7 Client: Compute $(\left(\frac{x+k_i}{p}\right))_{i=1,\dots,\lambda}$
```

3 Arithmetic MPC and Secret Sharing

In this section, we recall the definitions of some basic secret sharing methods and arithmetic protocols. The parties possessing the shares of a secret shares are referred to as servers. We use the terms *server* and *server index* interchangeably to refer to a server $S_1, \dots, S_n$ or the specific index $i = 1, \dots, n$ assigned to that server. We also use the term *share* and *share index* interchangeably to refer to a share of a secret sharing held by some server, and the index of the said share. For more detailed information the reader is advised to refer to [CDI05, Mau06, Esc22].

Additive secret sharing: Additive secret sharing (ASS) is a method to share 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.

Definition 3 (Additive secret sharing). *For $I \subseteq \{1, \dots, n\}$, $\#I > 1$, an element $x \in \mathbb{F}_p$ is additively secret shared among servers S_i for $i \in I$ as follows: Let $\mathcal{T} = I$ be the set of share indices. Select any share index $T_0 \in \mathcal{T}$ and then for each $T \in \mathcal{T} \setminus \{T_0\}$ sample x_T uniformly at random from $\mathbb{F}_p$, and set $x_{T_0} = x - \sum_{T \in \mathcal{T} \setminus \{T_0\}} x_T$. Then*

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

and the share of server S_i is the term x_T where $T = i$. We denote an ASS secret sharing of x with $[x]$.

Replicated secret sharing: Replicated secret sharing (RSS) [ss-87] is a generalisation of additive secret sharing, in which the same additive secret shares of an element are provided to multiple involved parties. This approach provides more freedom in choosing the threshold $1 \leq t \leq n-1$, which in its own part allows for more functionality. We assume that $t < \frac{n}{2}$ which in particular allows for communicationless multiplication. We follow the notation of [BBY20].

Definition 4 (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 $x \in F_p$ 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 $[\![x]\!]_R$.

We define $\mathcal{T}_i \subseteq \mathcal{T}$ to be the set of shares held by party S_i, i.e.,

$$\mathcal{T}_i := \{T \in \mathcal{T} \mid i \notin T\}.$$

We further define 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.

We define $\mathcal{S}_T \subseteq \{1, \dots, n\}$ to be 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 $[\![x]\!]_R$ the set of shares distributed to the servers by a (possibly corrupt) dealer.

$$[\![x]\!]_R = ((x_T^i)_{T \in \mathcal{T}_i})_{i=1,\dots,n}.$$

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 $[\![x]\!]_R$ *valid* if $x_T^i = x_T^j$ for all $i, j \in \mathcal{S}_T$ and all $T \in \mathcal{T}$. Then $[\![x]\!]_R$ is a replicated secret sharing $[\![x]\!]_R$ 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 $[\![x]\!]_R$ be an RSS of x distributed to n parties. For any subset of parties T, with $|T| \leq t$, the following conditions hold:*

1. *The view of the parties in T reveals no information about the secret x. Formally, for any secret x and any possible shares $s_1, s_2, \ldots, s_t$ held by the parties in T,*
$$\Pr[x|(x^1_T)_{T\in\mathcal{T}_1}, (x^2_T)_{T\in\mathcal{T}_2}, \ldots, (x^t_T)_{T\in\mathcal{T}_t}] = \Pr[x].$$
2. *The shares of the parties in x can be perfectly simulated using random numbers. That is, there exists a simulator Sim that generates shares $(\hat{x}^1_T)_{T\in\mathcal{T}_1}$, $(\hat{x}^2_T)_{T\in\mathcal{T}_2}$, …, $(\hat{x}^t_T)_{T\in\mathcal{T}_t}$ indistinguishable from the actual shares, without knowledge of the secret s.*

Doubly replicated secret sharing: In this section, we introduce doubly replicated secret sharing (DRSS), a variation of RSS with the same access structure but a different distribution of the shares, so it naturally satisfies the same security and privacy properties of RSS, in particular the (t, n)-DRSS satisfies t-privacy and t-security just like RSS.

Definition 5 (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 $\#\mathcal{T}^2 = \binom{n}{t}^2$,
$$x = \sum_{T_1,T_2\in\mathcal{T}} x_{T_1T_2},$$
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 $x_{T_1T_2}$ where $i \notin T_1 \cup T_2$.

Using the same notation from Section as in RSS we can see that each server S_i holds shares $\{T_1T_2 \,|\, T_1 \in \mathcal{T}_i, T_2 \in \mathcal{T}_i\}$, and that the share $x_{T_1T_2}$ is held exactly by servers $\mathcal{S}_{T_1T_2}$. In particular each $x_{T_1T_2}$ is held by at least 1 server when $t < n/2$. Furthermore, as that $n - 2t \leq \#\mathcal{S}_{T_1T_2}$ 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 $[\![x]\!]_D$ the set of shares distributed to the servers by a (possibly corrupt) dealer.
$$[\![x]\!]_D = ((x^i_{T_1T_2})_{T_1,T_2\in\mathcal{T}_i})_{i=1,\ldots,n}.$$
Each server S_i receives $(x^i_{T_1T_2})_{T_1T_2\in\mathcal{T}_i}$ where $x^i_{T_1T_2}$ is the share associated to index T_1T_2. We call $[\![x]\!]_D$ *valid* if $x^i_{T_1T_2} = x^j_{T_1T_2}$ for all $i, j \in \mathcal{S}_{T_1T_2}$ and all $T_1, T_2 \in \mathcal{T}$. Then $[\![x]\!]_D$ is a doubly replicated secret sharing $[\![x]\!]_D$ of $x = \sum_{T_1,T_2\in\mathcal{T}} x^i_{T_1T_2}$. Otherwise the sharing is called *invalid*, and it does not correspond to a DRSS sharing of a field element.

Optimised secret sharing: For a given secret $x \in \mathbb{F}_p$, denote the Optimised secret sharing (OSS)[BENO19] of x as $\langle x \rangle$ which is defined by sampling an ASS $[r_x]$ of a random field element $r_x \in \mathbb{F}_p$, computing $\delta_x = x + r_x$ and setting

$$\langle x \rangle = (\delta_x, [r_x])$$

where δ_x is a public value known to all parties, and $[r_x]$ is an ASS share of a random value $r_x \leftarrow_{\$} \mathbb{F}_p$; $r_x = \sum_{i=1}^{n} r_{x,i}$. The share of server S_i is $(\delta_x, r_{x,i})$.

Authenticated optimised secret sharing: Using the idea from SPDZ [DPSZ11], where an authenticated secret key α is negotiated among servers in the setup phase, we define authenticated optimised secret sharing in a similar fashion. The authenticated additive secret sharing (AASS) of a field element $x \in \mathbb{F}_p$ is defined as

$$[\![x]\!]_A := ([x], [\alpha x]),$$

where each server S_i holds additive secret shares x_i and $(\alpha x)_i$ of $[x]$ and $[\alpha x]$ respectively. The authenticated optimised secret sharing (AOSS) of a field element $x \in \mathbb{F}_p$ is defined by sampling $r_x \leftarrow_{\$} \mathbb{F}_p$ a random field element, setting $\delta_x = x + r_x$ and defining

$$\langle\!\langle x \rangle\!\rangle := (\delta_x, [\alpha x], [\alpha r_x]).$$

Each server S_i holds shares $(\delta_x, (\alpha x)_i, (\alpha r_x)_i)$.

3.1 Secret share functionalities

Secret share addition: All previously mentioned secret sharing schemes are linear. The computation of $[\![a + b]\!]$ from $[\![a]\!]$ and $[\![b]\!]$ can be computed with local operations at each server by adding the corresponding shares of the addends to obtain the share of the sum.

Secret share shares of zero: For any set of share indices $\dot{\mathcal{T}} \subseteq \{1, \ldots, n\}$ (ASS), $\dot{\mathcal{T}} = \mathcal{T}$ (RSS) or $\dot{\mathcal{T}} = \mathcal{T}^2$ (DRSS), and corresponding server set ($I = \dot{\mathcal{T}}$ for ASS or $\{1, \ldots, n\}$ for RSS/DRSS), we denote with $(r_{\dot{T}})_{\dot{T} \in \dot{\mathcal{T}}}$ an additive secret sharing of zero. In particular we have $\sum_{\dot{T} \in \dot{\mathcal{T}}} r_{\dot{T}} = 0$.

Secret share shares of constants: When a constant c is known to all the involved parties, a secret sharing $[\![c]\!]$ can be computed from a secret sharing of zero $(r_{\dot{T}})_{\dot{T} \in \dot{\mathcal{T}}}$ by setting

$$c_{\dot{T}} = r_{\dot{T}} + \tfrac{c}{\#\dot{\mathcal{T}}} \text{ for all } \dot{T} \in \dot{\mathcal{T}}.$$

It follows that $\sum_{\dot{T} \in \dot{\mathcal{T}}} c_{\dot{T}} = c$, and the secret sharing of the constant can be computed from a secret sharing of zero with only local operations by each server.

Secret shares of zero are useful for re-randomising secret shares held by servers by $[\![x]\!] + [\![0]\!] = [\![x]\!]$. Secret shares of zero can be computed during the precomputation stage and saved for later use, or computed on the go from secret shares of some secret elements, as explained in Appendix A.3.

Secret sharings of zero can be used to mask a reveal of a secret sharing. If the servers wish to reveal x to the client, without revealing the shares $x_{\dot{T}}$, they will compute an additive secret sharing of zero, and send $r_{\dot{T}} + x_{\dot{T}}$ to the client, who still obtains an additive secret sharing of x, but independent of the original shares.

RSS and DRSS share verification: The RSS and DRSS can be verified at opening under threshold assumptions. The only setting relevant to our construction is the one in which the servers reveal a (D)RSS by sending their shares to the client. The client receives the shares, checks if they are valid by comparing shares with the same index received from different servers. If there is a discrepancy, the client aborts. Otherwise, it accepts and reconstructs the secret by adding all the shares.

In RSS, when the threshold is bounded by $t < n/2$, for each share T there is an honest server holding that share. In particular, for $t < n/2$ the verification process is secure.

In DRSS, when the threshold is bounded by $t < n/3$, for each share pair T_1, T_2, there is an honest server holding that pair of shares. In particular, for $t < n/3$ the verification process is sound.

Masked opening of product: Let I be a set of server indices, and assume that the server S_i holds the values a^i and b^i. The servers wish to reveal the values $o^i = a^i \cdot b^i$ to the client under the condition that all the a^i are equal among themselves and all the b^i are equal among themselves. If they are not equal, the client receives a random value.

To do that, the clients are assumed to have a precomputed (or computed on-the-go) three additive secret sharing of 0:

$$(r_i^a)_{i\in I},\ \sum_{i\in I} r_i^a = 0;\ r_i^a \in S_i,$$
$$(r_i^b)_{i\in I},\ \sum_{i\in I} r_i^b = 0;\ r_i^b \in S_i,$$
$$(r_i)_{i\in I},\ \sum_{i\in I} r_i = 0;\ r_i \in S_i,$$

as well as $\#I$ public hash functions Hash_i. Server S_i computes

$$v_i = a^i \cdot b^i + a^i \cdot r_i^a + b^i \cdot r_i^b + r_i,$$

and $h^i = \mathrm{Hash}_i(a^i \cdot b^i)$, and sends $S_i \to (v_i, h^i)$ to the client. The client computes $v = \frac{1}{\#I}\sum_{i\in I} v_i$. They compare $\mathrm{Hash}_i(v)$ to h^i for all $i \in I$, and abort in case of inconsistencies. Otherwise the client accepts v.

Multiplication of RSS secret shares: The Legendre PRF protocol is reduced to computing a single addition and a single multiplication of RSS secret shares. Due to the nature of our requirements, the multiplication algorithm does not produce another RSS share, but instead a different type of secret share. We propose two algorithms, one which multiplies two RSS secret shares to obtain an ASS secret share, and the second which outputs a DRSS secret share.

We assume that n servers hold two RSS secret shares $[\![a]\!]_R$ and $[\![b]\!]_R$. They wish to compute

$$ab = \sum_{T_1 \in \mathcal{T}} a_{T_1} \sum_{T_2 \in \mathcal{T}} b_{T_2} = \sum_{T_1, T_2 \in \mathcal{T}} a_{T_1} b_{T_2}.$$

The servers will compute this by computing all terms of the product that they have access to.

RSS to ASS multiplication algorithm Each server S_i holds shares $(a_T, b_T)_{T \in \mathcal{T}_i}$. The protocol proceeds by the servers multiplying all terms of the product that they have access to, normalised by an appropriate factor. In particular server S_i computes

$$o_i := \sum_{T_1, T_2 \in \mathcal{T}_i} \frac{1}{\# S_{T_1 T_2}} a_{T_1} b_{T_2}.$$

This procedure is denoted with $[\![a]\!], [\![b]\!] \mapsto [a \cdot b]$.

The opening of the product proceeds by masking all the shares with a secret sharing of zero $(r_i)_{i=1}^n$ $(o_i \mapsto o_i + r_i)$ and revealing the shares:

$$\begin{aligned}\sum_{i=1}^n o_i + r_i &= \sum_{i=1}^n \sum_{T_1, T_2 \in \mathcal{T}_i} \frac{1}{\# S_{T_1 T_2}} a_{T_1} b_{T_2} \\ &= \sum_{T_1, T_2 \in \mathcal{T}} \sum_{i \in \mathcal{S}_{T_1 T_2}} \frac{1}{\# S_{T_1 T_2}} a_{T_1} b_{T_2} \\ &= \sum_{T_1, T_2 \in \mathcal{T}} a_{T_1} b_{T_2} = a \cdot b.\end{aligned}$$

RSS to DRSS multiplication algorithm Each server S_i holds shares $(a_T, b_T)_{T \in \mathcal{T}_i}$. The protocol proceeds by the servers multiplying all terms of the product that they have access to.

$$o_{T_1 T_2} = a_{T_1} b_{T_2} \text{ for each } T_1, T_2 \in \mathcal{T}_i.$$

Then the $(o_{T_1 T_2})_{T_1, T_2 \in \mathcal{T}}$ form a DRSS of $a \cdot b$. In fact $o_{T_1 T_2}$ are equal for each $i \in \mathcal{S}_{T_1 T_2}$, and

$$\sum_{T_1, T_2 \in \mathcal{T}} o_{T_1 T_2} = \sum_{T_1, T_2 \in \mathcal{T}} a_{T_1} b_{T_2} = a \cdot b.$$

We also denote this procedure with $[\![a]\!]_R, [\![b]\!]_R \mapsto [\![a \cdot b]\!]_D$.

An opening of a DRSS product proceeds by masking with a DRSS secret sharing of zero each term of this product $o_{T_1 T_2} \mapsto o_{T_1 T_2} + r_{T_1 T_2}$. Then, each term of the DRSS is revealed with a *masked opening of product*. This procedure ensures that the product is revealed only if all the terms of the RSS shares $[\![a]\!]_R$ and $[\![b]\!]_R$ are valid.

OSS to OSS multiplication algorithm Assume that parties participating in secret sharing hold $\langle x\rangle = (\delta_x, [r_x])$, $\langle y\rangle = (\delta_y, [r_y])$ as well as an additive secret sharing of the product $[r_x r_y]$. Then, they can compute the product $\langle xy\rangle$ by only using local operations as follows:

$$\begin{aligned}\delta_{xy} &:= \delta_x \cdot \delta_y\\ r_{xy} &= xy - \delta_{xy} = \delta_x r_y + \delta_y r_x - r_x r_y\\ [r_{xy}] &= \delta_x[r_y] + \delta_y[r_x] - [r_x r_y],\\ \langle xy\rangle &= (\delta_{xy}, [r_{xy}]).\end{aligned}$$

In particular server S_i computes $(r_{xy})_i = \delta_x(r_y)_i + \delta_y(r_x)_i - (r_x r_y)_i$.

4 The Legendre dOPRF protocol

We provide four different protocols for computing the distributed Legendre OPRF in two different adversarial settings. Two protocols are based on replicated secret sharing, and the other two on optimised secret sharing. The RSS protocols follow the communication structure presented in Figure 1. The OSS protocols follow the same communication structure, with the addition of a client-server communication round in the setup phase. Prior to starting the protocol, the servers share a random OSS with the client which is then used for masking the client's input in the on-line phase. The protocols are presented for a single bit output. For a more general λ-bit output, the setup, evaluation and reconstruction phase are executed in parallel λ times with the same input.

4.1 Semi-honest RSS protocol

In this subsection, we present the protocol which computes the Legendre OPRF in a distributed setting without inter-server communication under a semi-honest adversary assumption.

We assume $t < n/2$, at most t semi-honest servers and a semi-honest client that may be colluding. The servers and clients follow the steps of the protocol, but cannot gather any information apart from the output due to the use of secure secret sharing and multiparty computation.

Let $f : (x, k, s^2) \mapsto (x + k)s^2$ be the function we want to securely compute. Before giving our detailed protocol description, we model the ideal functionalities first. We define an ideal functionality $\mathcal{F}_{\mathsf{Setup}}$ in the following that interacts with each server $\mathcal{S}_i, i = 1, \dots, n$ in the pre-computation phase.

- **Input**: $\mathcal{F}_{\mathsf{Setup}}$ receives a start command from the servers.
- **Computation**: $\mathcal{F}_{\mathsf{Setup}}$ samples $s, k \leftarrow_{\$} \mathbb{F}_p$, computes $[\![k]\!]_R, [\![s^2]\!]_R$ and an n-party ASS $[r]$ where $r = 0$.
- **Output**: $\mathcal{F}_{\mathsf{Setup}}$ distributes $[\![k]\!]_R, [\![s^2]\!]_R$ and $[r]$ to the servers.

The ideal functionality $\mathcal{F}_{\mathsf{RSS-OPRF}}$, which interacts with the client and all n servers is defined in the evaluation phase as follows:

- **Input**: $\mathcal{F}_{\mathsf{RSS-OPRF}}$ inputs $x \in \mathbb{F}_p$ from the client, and RSS $[\![k]\!]_R, [\![s^2]\!]_R$ from n servers where $k, s \in \mathbb{F}_p$.
- **Computation**: $\mathcal{F}_{\mathsf{RSS-OPRF}}$ reconstructs x from $[\![x]\!]_R$, k from $[\![k]\!]_R$, s^2 from $[\![s^2]\!]_R$, and computes $y = f(x, k, s^2)$.
- **Output**: $\mathcal{F}_{\mathsf{RSS-OPRF}}$ outputs y to the client.

We present our full protocol details in algorithm 3, which includes setup, input, evaluation and reconstruction stages.

Setup stage Let $\mathcal{T}$ be the set of shares defined as in Section 3. In the setup (precomputation) stage, the servers are assumed to hold shares of the following values

- An RSS sharing of the key, $[\![k]\!]_R$,
- An RSS sharing of a random square, $[\![s^2]\!]_R$,
- An n-party ASS sharing of 0, $[r]$,

These values are assumed to be precomputed and verified for consistency by the clients. While the key $[\![k]\!]_R$ and the masking square $[\![s^2]\!]_R$ are always assumed to have been precomputed, the secret sharing of 0 can also be computed in the evaluation phase, as explained in the Appendix A.

In total, each server holds $\binom{n-1}{t}$ field elements for k and s^2 each, and a single field element for the ASS share of 0, totaling to $2\binom{n-1}{t} + 1$ field elements.

Input stage In the input stage, the client shares an input $x \in \mathbb{F}_p$ as a (t, n)-RSS to the servers. The client computes $[\![x]\!]_R$ and sends the corresponding shares to the servers:

$$x = \sum_{T \in \mathcal{T}} x_T, \; S_i \text{ given } (x_T)_{T \in \mathcal{T}_i}.$$

Evaluation stage After receiving the inputs, the servers compute RSS of $a = x + k$ and $b = s^2$ by setting $a^i_T = x^i_K + k_T$ and $b^i_T = s^2_T$, and proceed to compute $[\![a]\!]_R, [\![b]\!]_R \mapsto [a \cdot b]$. The server S_i computes the sum of all the term-wise products of $a \cdot b$ they have access to, and masks it with the ASS of zero:

$$o^i := \Big(\sum_{T_1, T_2 \in \mathcal{T}_i} \frac{1}{\#\mathcal{S}_{T_1 T_2}} a^i_{T_1} b^i_{T_2} \Big) + r_i.$$

The server S_i returns o^i to the client, in total sending a single field element.

Reconstruction stage The client computes $o = \sum_{i=1}^n o^i = a \cdot b$.

Algorithm 3 Legendre dOPRF in the semi-honest setting

```
Input stage: Client
Input : $x \in \mathbb{F}_p$
1  Compute a RSS of $x = \sum_{T \in \mathcal{T}} x_T$
2  return $(x_T)_{T \in \mathcal{T}_i}$ to server $S_i$

Evaluation stage: Server $S_i$; $i = 1, \ldots, n$
Input : $(x^i_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of input provided by the client
        For each $j = 1, \ldots, \lambda$:
          $(k^j_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of the secret keys
          $(s^{2,j}_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of the square terms
          $r^j_i \in \mathbb{F}_p$ share of ASS sharing of 0
3  for $j = 1, \ldots, \lambda$ do
4      for $T \in \mathcal{T}_i$ do
5          Set $a^j_T = x^i_T + k^j_T$ for $T \in \mathcal{T}_i$
6          Set $b^j_T = s^{2,j}_T$ for $T \in \mathcal{T}_i$
7      Compute $o^j_i = r^j_i + \sum_{T_1, T_2 \in \mathcal{T}_i} \frac{1}{\# \mathcal{S}_{T_1 T_2}} a^j_{T_1} b^j_{T_2}$
8  return $(o^j_i)^{\lambda}_{j=1}$ to client.

Reconstruction stage: Client
Input : $(o^j_i)^{\lambda}_{j=1} \subseteq \mathbb{F}^{\lambda}_p$ outputs from servers $i = 1, \ldots, n$
9  for $j = 1, \ldots, \lambda$ do
10     $v^j = \sum_{i=1}^{n} o^j_i$
11 return $\left(\left(\frac{v^j}{p}\right)\right)^{\lambda}_{j=1} = \left(\left(\frac{x+k_i}{p}\right)\right)^{\lambda}_{j=1}$
```

Security in the semi-honest setting In the following proof as well as all provided proofs in the paper, we consider that for all protocols, we have two scenarios: the *real-world execution* and the *ideal-world execution.* In both scenarios, an adversary, denoted as $\mathcal{A}$, may control certain participants - this could be the client, some servers, or even a combination of the client and several servers. The adversary $\mathcal{A}$ has access to the input/output information of the parties under its control, potentially including intermediate random tapes or the ultimate output of the protocol.

However, the ideal-world execution scenario is deemed perfectly secure even in the presence of the adversary $\mathcal{A}$. We characterize a protocol as perfectly (or statistically) secure if, for any given adversary $\mathcal{A}$, there exists a simulator capable of generating an output distribution, using only the view transcripts available in the ideal-world execution, that is perfectly (statistically) indistinguishable from the real output distribution in the ideal-world execution.

Definition 6. *(Security of algorithm 3) In the $\mathcal{F}_{\mathsf{Setup}}$-hybrid model, a simulator* Sim *exists for all $x, k, s \in \mathbb{F}_p$ and the function $f : (x, k, s^2) \mapsto (x + k)s^2$,* Sim *realizes $\mathcal{F}_{\mathsf{RSS-OPRF}}$ in a way that ensures the simulated view transcript is perfectly indistinguishable from the real protocol execution in 3 when facing a semi-honest adversary $\mathcal{A}$. This adversary may control either a semi-honest client or up to t semi-honest servers, or any collaborative arrangement between these entities.*

Theorem 2. *Algorithm 3 securely realizes $\mathcal{F}_{\mathsf{RSS-OPRF}}$.*

Proof. We distinguish three scenarios based on the adversary $\mathcal{A}$'s control, each requires a distinct simulator construction.

- *$\mathcal{A}$ controls only the client: We construct a simulator* Sim *that inputs the received output result y from the oracle $\mathcal{F}_{\mathsf{RSS-OPRF}}$ that can be queried by $\mathcal{A}$. To simulate the view transcript $\{o^i\}_{i=1^n}$ transmitted from the servers to the client, it suffices for* Sim *to generate a random ASS of y, denoted as $[y]$, that is identical to $\{o^i\}_{i=1^n}$ as that in the real world. Thus,* Sim *perfectly simulates the behaviors of the honest parties.*
- *$\mathcal{A}$ controls at most t servers: We construct a simulator* Sim*, which operates without any external inputs. The primary objective of* Sim *is to simulate the view transcript that is transmitted from the client to the t servers. According to Theorem 1, all shares held by these t servers can be perfectly simulated.*
- *$\mathcal{A}$ controls at most t servers and the client: We construct a simulator* Sim *that accepts x from $\mathcal{A}$ as input, as well as the received output result y from the oracle $\mathcal{F}_{\mathsf{RSS-OPRF}}$ that can be queried by $\mathcal{A}$, as well as all information held by those t malicious servers. Firstly, to simulate the view transcript transmitted from the client to these t servers, we generate the RSS of x, and the simulator outputs the corresponding shares to t servers which are identical to that in the real protocol execution; Secondly, to simulate the view transcript $\{o^i\}_{i=1,\dots,n-t}$ (assume these are the index of honest servers) transmitted from the honest servers to the client, it suffices for* Sim *to generate a random $(n-t)$-party ASS of $\bar{r}$, denoted as $[\bar{r}]$, that is perfectly indistinguishable to $\{o^i\}_{i=1,\dots,n-t}$ in the real world.*

4.2 Malicious RSS protocol

In this subsection, we present the main algorithm which computes the Legendre OPRF in a distributed setting without inter-server communication under a malicious adversary assumption.

We assume $t < n/3$, at most t malicious servers and a potentially malicious client that may be colluding.

If the client acts maliciously and sends an invalid RSS of the input to the honest clients, then they will receive a random output, and the adversary will not be able to deduce any information from the received data.

If the client acts honestly, then they will be able to notice any malicious activity by the adversary, and in that case abort. Otherwise, they receive the correct output, and no secret information is leaked.

Under this scenario, we model the ideal functionalities slightly differently. We define an ideal functionality $\mathcal{F}_{\mathsf{Setup}^*}$ in the following that interacts with each server $\mathcal{S}_i, i = 1, \ldots, n$ in the pre-computation phase.

- **Input**: $\mathcal{F}_{\mathsf{Setup}^*}$ receives a start command from the servers.
- **Computation**: $\mathcal{F}^*_{\mathsf{Setup}}$ samples $s, k \leftarrow_\$ \mathbb{F}_p$, computes $[\![k]\!]_R$ and $[\![s^2]\!]_R$, a DRSS of $[\![r]\!]_D$ where $r = 0$, and $\#S_{T_1T_2}$-party ASS of $[t^{T_1T_2}], [\bar{t}^{T_1T_2}], [\hat{t}^{T_1T_2}]$ for each $T_1, T_2 \in \mathcal{T}$ where $t^{T_1T_2} = \bar{t}^{T_1T_2} = \hat{t}^{T_1T_2} = 0$.
- **Output**: $\mathcal{F}_{\mathsf{Setup}^*}$ distributes the RSS $[\![k]\!]_R$ and $[\![s^2]\!]_R$, along with the n-party ASS $[r]$, to the servers. Additionally, for each pair of thresholds T_1, T_2 in the set $\mathcal{T}$, it provides every party in $\mathcal{S}_{T_1T_2}$ with the ASS $[t^{T_1T_2}], [\bar{t}^{T_1T_2}]$, and $[\hat{t}^{T_1T_2}]$.

In cases where an adversary $\mathcal{A}$ manipulates the entire computation, either by supplying invalid RSS or deviating from the specified protocol, we modify the ideal functionality $\mathcal{F}_{\mathsf{RSS-OPRF}}$, originally designed for the semi-honest setting, to $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ as detailed below.

- **Input**: $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ inputs $[\![x]\!]_R$ from the client where $x \in \mathbb{F}_p$, and $[\![k]\!]_R, [\![s^2]\!]_R$ from each server $\mathcal{S}_i, i = 1, \ldots, n$.
- **Computation**: $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ checks the validity of all three RSSs, in the case all RSSs are *valid* $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ reconstructs x, k, s^2 and then computes $y = f(x, k, s^2)$.
- **Output**: In the case all three RSSs are *valid*, $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ outputs y to the client; otherwise it outputs $(\perp, r)$ to the client where $r \leftarrow_\$ \mathbb{F}_p$.

$\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ interacts with the client and each server $\mathcal{S}_i, i = 1, \ldots, n$ in the online phase, and the adversary $\mathcal{A}$ is parameterized by a public known function $f : (x, k, s^2) \mapsto (x + k)s^2$.

We present the protocol for a single bit output. For a more general λ-bit output, the setup, evaluation and reconstruction phase are executed in parallel λ times with the same input.

Setup stage Let $\mathcal{T}$ be the set of shares defined as in Section 3. In the setup (precomputation) stage, the servers are assumed to hold the following values:

- An RSS sharing of the key, $[\![k]\!]_R$,
- An RSS sharing of a random square, $[\![s^2]\!]_R$,
- A DRSS sharing of 0, $[\![r]\!]_D$,
- An ASS sharing of 0, $[t^{T_1T_2}]$ shared among $S_{T_1T_2}$, for each $T_1, T_2 \in \mathcal{T}$.
- An ASS sharing of 0, $[\bar{t}^{T_1T_2}]$ shared among $S_{T_1T_2}$, for each $T_1, T_2 \in \mathcal{T}$.
- An ASS sharing of 0, $[\hat{t}^{T_1T_2}]$ shared among $S_{T_1T_2}$, for each $T_1, T_2 \in \mathcal{T}$.

These values are assumed to be precomputed and verified for consistency by the clients. While the key $[\![k]\!]_R$ and the masking square $[\![s^2]\!]_R$ are always assumed to have been precomputed, the secret sharings of 0 can also be computed in the evaluation phase, as explained in Appendix A.3.

In total, each server holds $\binom{n-1}{t}$ field elements for k and s^2 each, $\binom{n-1}{t}^2$ field elements for the DRSS share of 0, and three field element for each of the $\binom{n-1}{t}^2$ ASS shares of 0, totaling to $4\binom{n-1}{t}^2 + 2\binom{n-1}{t}$ field elements.

Input stage In the input stage the client shares an input $x \in \mathbb{F}_p$ as a (t, n)-RSS to the servers. The client computes $[\![x]\!]_R$ and sends the corresponding shares to the servers:

$$x = \sum_{T \in \mathcal{T}} x_T,\ S_i \text{ given } (x_T^i)_{T \in \mathcal{T}_i} \text{ where } x_T^i = x_T.$$

We use notation x_T^i to cover the case in which a malicious client might send inconsistent shares to the servers.

Evaluation stage After receiving the inputs, the servers compute RSS of $a = x + k$ and $b = s^2$ by setting $a_T^i = x_T^i + k_T^i$ and $b_T^i = (s^2)_T^i$, and proceed to compute $[\![a]\!]_R, [\![b]\!]_R \mapsto [\![a \cdot b]\!]_D$.

The server S_i computes all the term-wise products of $a \cdot b$ they have access to, and masks them with the DRSS of zero:

$$o_{T_1T_2}^i := a_{T_1}^i b_{T_2}^i + r_{T_1T_2}.$$

At this point all servers indexed in $\mathcal{S}_{T_1T_2}$ should have the same values $o_{T_1T_2}^i$, assuming they received a valid input from the client. They proceed by revealing each of the $o_{T_1T_2}^i$ with the masked opening method from Section 3.1.

For each $T_1, T_2 \in \mathcal{T}_i$, each server uses three precomputed secret shares of 0 shared among $\mathcal{S}_{T_1T_2}$, $(t_i^{T_1T_2})_{i \in \mathcal{S}_{T_1T_2}}$, $(\bar{t}_i^{T_1T_2})_{i \in \mathcal{S}_{T_1T_2}}$, $(\hat{t}_i^{T_1T_2})_{i \in \mathcal{S}_{T_1T_2}}$ to compute

$$v_i^{T_1T_2} := \frac{1}{\mathcal{S}_{T_1T_2}} o_{T_1T_2}^i + a_{T_1}^i t_i^{T_1T_2} + b_{T_2}^i \bar{t}_i^{T_1T_2} + \hat{t}_i^{T_1T_2},$$

as well as $h_{T_1T_2}^i := \text{Hash}_i(o_i^{T_1T_2})$, where Hash_i is a different public hash function for each $i = 1, \ldots, n$. They send the following values to the client:

$$S_i \to (v_i^{T_1T_2}, h_{T_1T_2}^i)_{T_1, T_2 \in \mathcal{T}_i},$$

In total, being $\binom{n-1}{t}^2$ field and hash values.

Reconstruction stage The client can verify the computation was done correctly by firstly computing $v_{T_1T_2} = \sum_{i \in \mathcal{S}_{T_1T_2}} v_i^{T_1T_2}$ for each $T_1, T_2 \in \mathcal{T}$. Then, the client compares $\text{Hash}_i(v_i^{T_1T_2})$ with $h_{T_1T_2}^i$ for all $i \in \mathcal{S}_{T_1T_2}$ and aborts in case of inequality. Otherwise, they finish the protocol by computing:

$$\sum_{T_1, T_2 \in \mathcal{T}} v_{T_1T_2} = \sum_{T_1, T_2 \in \mathcal{T}} a_{T_1} b_{T_2} + r_{T_1T_2} = ab.$$

The pseudocode for the full protocol for a λ-bit output is given in algorithm 4.

Algorithm 4 Legendre dOPRF in the malicious setting

```
Input stage: Client
Input : $x \in \mathbb{F}_p$

1  Compute a RSS of $x = \sum_{T \in \mathcal{T}} x_T$
2  return $(x_T)_{T \in \mathcal{T}_i}$ to server $S_i$

Evaluation stage: Server $S_i$; $i = 1, \dots, n$
Input : $(x^i_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of input provided by the client
        For each $j = 1, \dots, \lambda$:
        $(k^j_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of the secret keys
        $(s^{2,j}_T)_{T \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of the square terms
        $(r^j_{T_1T_2})_{T_1,T_2 \in \mathcal{T}_i} \subseteq \mathbb{F}_p$ shares of the DRSS sharings of 0
        $t^{T_1T_2,j}_i \in \mathbb{F}_p$ share of ASS sharing of 0 (for each $T_1, T_2 \in \mathcal{T}_i$)
        $\bar{t}^{T_1T_2,j}_i \in \mathbb{F}_p$ share of ASS sharing of 0 (for each $T_1, T_2 \in \mathcal{T}_i$)
        $\hat{t}^{T_1T_2,j}_i \in \mathbb{F}_p$ share of ASS sharing of 0 (for each $T_1, T_2 \in \mathcal{T}_i$)

3  for $j = 1, \dots, \lambda$ do
4      for $T \in \mathcal{T}_i$ do
5          Set $a^j_T = x^i_T + k^j_T$ for $T \in \mathcal{T}_i$
6          Set $b^j_T = s^{2,j}_T$ for $T \in \mathcal{T}_i$
7      for $T_1, T_2 \in \mathcal{T}_i$ do
8          Compute $o^j_{T_1T_2} = a^j_{T_1} b^j_{T_2} + r^j_{T_1T_2}$
9          Compute $v^{T_1T_2,j}_i = o^j_{T_1T_2} + a^j_{T_1} \cdot t^{T_1T_2,j}_i + b^j_{T_2} \cdot \bar{t}^{T_1T_2,j}_i + \hat{t}^{T_1T_2,j}_i$
10         Compute $h^{i,j}_{T_1T_2} = \mathrm{Hash}_i(o^j_{T_1T_2})$
11 return $((v^{T_1T_2,j}_i, h^{i,j}_{T_1T_2})_{T_1,T_2 \in \mathcal{T}_i})^{\lambda}_{j=1}$ to client.

Reconstruction stage: Client
Input : $((v^{T_1T_2,j}_i, h^{i,j}_{T_1T_2})_{T_1,T_2 \in \mathcal{T}_i})^{\lambda}_{j=1} \subseteq \mathbb{F}^2_p$ outputs from $\mathcal{S}_i$, $i = 1, \dots, n$

12 for $j = 1, \dots, \lambda$ do
13     for $T_1, T_2 \in \mathcal{T}$ do
14         $v^j_{T_1T_2} = \frac{1}{\#\mathcal{S}_{T_1T_2}} \sum_{i \in \mathcal{S}_{T_1T_2}} v^{T_1T_2,j}_i$
15         for $i \in P_{T_1T_2}$ do
16             if $\mathrm{Hash}_i(v^j_{T_1T_2}) \neq h^{i,j}_{T_1T_2}$ then $\perp$;
17     Compute $v^j = \sum_{T_1,T_2 \in \mathcal{T}} v^j_{T_1T_2}$
18 return $\left(\left(\frac{v^j}{p}\right)\right)^{\lambda}_{j=1} = \left(\left(\frac{x+k_i}{p}\right)\right)^{\lambda}_{j=1}$
```

Client Verifiability Consider a scenario where an honest client interacts with up to t may be controlled by an adversary $\mathcal{A}$. In this setting, we assert the verification algorithm described in the reconstruction stage of algorithm 4 is an efficient verification algorithm such that:

- It outputs an correct computation of $(x+k)\cdot s^2$ only if all servers, including those potentially under adversarial control, adhere strictly to the protocol specifications.
- Otherwise, it outputs $\perp$ if any of the servers, up to a maximum of t, deviate or disrupt the protocol procedure.

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 adversarial servers.

Algorithm 5 Designing a Simulator for Handling valid RSS Provided by a Malicious Client

```
Input  : A valid $[\![x]\!]_R$ from the client, $y$ from $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$.
Output: $\{(\hat{v}_i^{T_1T_2}, \hat{h}^i_{T_1T_2})_{T_1,T_2\in\mathcal{T}_i}\}_{i=1}^n$
1 Samples $\hat{o}^{T_1T_2} \leftarrow_\$ \mathbb{F}_p$ for every $T_1, T_2 \in \mathcal{T}$ except for $\hat{o}_n^{T_{\mathsf{Max}}T_{\mathsf{Max}}}$, here $T_{\mathsf{Max}}$
    denotes the last share index in $\mathcal{T}$.
2 Denote $\hat{o}$ as the sum of these $I-1$ randomly generated values, then
    computes $\hat{o}^{T_{\mathsf{Max}}T_{\mathsf{Max}}} = y - \hat{o}$.
3 Computes $\hat{h}^i_{T_1T_2} = \mathrm{Hash}_i(\hat{o}^{T_1T_2})$ for every $T_1, T_2 \in \mathcal{T}_i, i = 1, \dots, n$.
4 Generates a random ASS sharing $[\hat{o}^{T_1T_2}]$ for every $T_1, T_2 \in \mathcal{T}$, denote
    $\hat{v}_i^{T_1T_2}$ as the share of $[\hat{o}^{T_1T_2}]$ held by server $S_i$ where $i \in \mathcal{S}_{T_1T_2}$.
5 return
        $\{(\hat{v}_i^{T_1T_2}, \hat{h}^i_{T_1T_2})_{T_1,T_2\in\mathcal{T}_i}\}_{i=1}^n$.
```

Security in the malicious setting

Definition 7. *(Security of algorithm 4) In the $\mathcal{F}^*_{\mathsf{Setup}}$-hybrid model, there exists a simulator* Sim *that $\forall x, k, s \in \mathbb{F}_p$ and function $f : (x, k, s^2) \mapsto (x+k)s^2$, it realizes the ideal functionality $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$ such that its simulated view transcript is computationally indistinguishable from a real world execution in algorithm 4 in the presence of a malicious adversary $\mathcal{A}$, which controls either a semi-honest client or at most t semi-honest servers, or any collusive arrangement between these two entities.*

Theorem 3. *The algorithm 4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$.*

Proof. *From the definition of DRSS, we know there are in total $I = \binom{n}{t}^2$ shares constituting a DRSS sharing, where $I - 1$ is exactly the amount of independent random numbers needed for generating the DRSS sharing of a secret. And $\sum_{T_1,T_2\in\mathcal{T}}(\#\mathcal{S}_{T_1T_2} - 1)$ is exactly the amount of independent random numbers needed when generating the ASS sharing of 1 in the algorithm 4. Thus, in total, $\hat{I}$ independent random numbers are needed in algorithm 4, where:*

$$\begin{aligned}\hat{I} &= I - 1 + \sum_{T_1,T_2\in\mathcal{T}}(\#\mathcal{S}_{T_1T_2} - 1)\\ &= I - 1 + \sum_{T_1,T_2\in\mathcal{T}}(\#\mathcal{S}_{T_1T_2}) - I\\ &= n \cdot \binom{n-1}{t}^2 - 1\end{aligned}$$

We define three scenarios based on the adversary $\mathcal{A}$'s control. Each scenario requires a distinct simulator construction.

– *$\mathcal{A}$ controls the client: We construct a simulator* Sim *that inputs $[\![x]\!]_R$ from the client, as well as the output y or $(\perp, r)$ from the oracle queried by $\mathcal{A}$, and aim to simulate the view transcript $(v_i^{T_1T_2}, h^i_{T_1T_2})_{T_1,T_2\in\mathcal{T}_i}$ sent from each $\mathcal{S}_i$ in the execution of Protocol 2 (detailed in algorithm 4). Here, we further differentiate between two scenarios based on the validity of the input $[\![x]\!]_R$:*
  - *If $[\![x]\!]_R$ is* valid, *let* Sim *samples $\hat{o}^{T_1T_2} \leftarrow_\$ \mathbb{F}_p$ for every $T_1, T_2 \in \mathcal{T}$ where $T_1 \neq T_{\mathsf{Max}}, T_2 \neq T_{\mathsf{Max}}$, here T_{Max} is defined as the last one in $\mathcal{T}$. Let $\hat{o}$ be the sum of those $I - 1$ randomly generated values. Then, the simulator* Sim *computes $\hat{o}^{T_{\mathsf{Max}}T_{\mathsf{Max}}} = y - \hat{o}$. Further, let* Sim *generate a random additive secret sharing for each $\hat{o}^{T_1T_2}, T_1, T_2 \in \mathcal{T}$, which is denoted as $\hat{v}_i^{T_1T_2}, i \in \mathcal{S}_{T_1T_2}$. Meanwhile, let* Sim *compute $\hat{h}^i_{T_1T_2} = \text{Hash}_i(\hat{o}^{T_1T_2})$ for every $T_1, T_2 \in \mathcal{T}_i, i = 1, \ldots, n$. Finally,* Sim *outputs:*

$$\{(\hat{v}_i^{T_1T_2}, \hat{h}^i_{T_1T_2})_{T_1,T_2\in\mathcal{T}_i}\}_{i=1}^n.$$

The construction of Sim *is also formally presented in Algorithm 5. When comparing the simulated view above to the execution of Protocol 2 (outlined in algorithm 4), we define* View *as the aggregated view of all $S_i, i = 1, \ldots, n$. This view encompasses* View_v *and* View_h*, where* View_v *is a list of size $\hat{I} + 1$ that combines the first element of each term received from the servers, and* View_h *the list that combines every second element. We derive similar definitions* $\mathsf{View}^*, \mathsf{View}_{v^*}, \mathsf{View}_{h^*}$ *from our simulated transcripts. Thus,* Sim *perfectly simulates the behaviors of the honest parties.*
We see that both View_v *and* View_{v^*} *are determined by $\hat{I}$ uniformly random independent numbers and the value y. Thus,* $\mathsf{View}_v \overset{d}{=} \mathsf{View}_{v^*}$*. With the fact that* $\mathsf{View}_h/\mathsf{View}_{h^*}$ *are derived from* $\mathsf{View}_v/\mathsf{View}_{v^*}$*, further we have* $\mathsf{View}_h \overset{d}{=} \mathsf{View}_{h^*}$*.*

- *Otherwise, assume a submitted $[\![x]\!]_R$ from the client is* valid *except for one share index T^*, i.e., there are inconsistent values of share x_{T_*} held by servers $\mathcal{S}_{T_*}$. Formally, we define $\bar{x}^i_{T_*} = e^i + x_{T_*}$ where $\exists, i, j \in \mathcal{S}^e_{T_*}$ that $e^i \neq e^j$, i.e., e^i denotes the error on $\bar{x}^i_{T_*}$. Here $\bar{x}^i_{T_*}$ denotes a potential invalid share distributed to server $\mathcal{S}_i$ with share index T^* from the client. Let us analyse the final output distribution in algorithm 4 under this case. Given a pair of share indices $T_*, T_2 \in \mathcal{T}$, for every $T_2 \in \mathcal{T}$ the client is able to reconstruct $o^{T_*T_2} = a_{T_*}b_{T_2} + r_{T_*T_2} + e_{T_*T_2}$ for $i \in \mathcal{S}_{T_*T_2}$ where $e_{T_*T_2} = b_{T_2} \cdot \sum_{i \in \mathcal{S}_{T_*T_2}} e^i + \sum_{i \in \mathcal{S}_{T_*T_2}} (e^i t_i^{T_*T_2})$, observe that $e_{T_*T_2} \simeq \mathcal{U}$. Thus, the final value $o^{T_*T_2}$ is independent to $a_{T_*}b_{T_2} + r_{T_*T_2}$, and with the masking of $\hat{t}_i^{T_1T_2,j}$ from the evaluation stage (detailed in Algorithm 4), $\{v^i_{T_*T_2}\}_{i \in \mathcal{S}_{T_*T_2}}$ constitutes a random ASS of $o^{T_*T_2}$.*
  Note that every $\mathcal{S}_i \in \mathcal{S}_{T_1T_2}$ computes $h^i_{T_1T_2} = \mathrm{Hash}_i(o^i_{T_1T_2})$, where $o^i_{T_1T_2} = a_{T_*}b_{T_2} + r_{T_*T_2} + e^i \cdot b_{T_2}$ is masked by a random number $r_{T_*T_2}$. Thus, in the random oracle model, if we draw a random element denoted as $\hat{h}^i_{T_1T_2}$ from the output domain of Hash_i, we argue that any PPT $\mathcal{A}$ cannot distinguish between $\hat{h}^i_{T_1T_2}$ and the real execution output $h^i_{T_1T_2}$.
  Finally, the client is able to recover $y^* = (x+k) \cdot s^2 + E$, where $E = \sum_{T_2 \in \mathcal{T}} e_{T_*T_2}$ is the final error that equals the sum of $\#\mathcal{T}$ random independent numbers. Thus, $y^* \simeq \mathcal{U}$. And the distribution of the final real world protocol output y^* doesn't change when the client submits multiple invalid shares to the servers.
  From above analysis, we present the final simulator construction in algorithm 6, wherever the simulated view is indistinguishable to that in the corresponding real world execution.

– *$\mathcal{A}$ controls at most t servers: We need to simulate the view transcript from the honest client to these t servers. Following 1, let $\mathcal{A}$ randomly selecting $\binom{n}{t} - 1$ elements from $\mathbb{F}_p$, which we argue are identical to all those $\binom{n}{t} - 1$ shares of x send by the client in the real protocol execution.*
  For any RSS share x_{T_*}, k_{T_*}, $s^2_{T_*}$ held by $\mathcal{A}$ where $T_* \in \mathcal{T}$, suppose there is an addition of non-constant error to at least one of x_{T_*}, k_{T_*}, or $s^2_{T_*}$ which is subsequently utilized in the evaluation phase of Algorithm 4.
  In this scenario, we argue that the final real-world output distribution will be identical to the one of real-world output distribution observed in the earlier case, where the client submits an invalid RSS of x to honest servers, i.e., this is a distribution determined by a random number from $\mathbb{F}_p$. Moreover, using simulator description in algorithm 6, we could output a simulated real-world output distribution. This means the real-world output distribution is identical to what in the ideal functionality $\mathcal{F}_{\mathsf{RSS-OPRF}^*}$.
  Thus, we argue by using only invalid RSS that $\mathcal{A}$ couldn't disrupt the final output distribution to make it related with any input x, k, or s^2.
– *$\mathcal{A}$ controls at most t servers and the client: We construct a simulator* Sim *that accepts $[\![x]\!]_R$ from $\mathcal{A}$ as input, the received output result y or $(\perp, r)$ from the oracle queried by $\mathcal{A}$, as well as all information held by those t malicious servers. We denote $\mathcal{S}_{\mathsf{Hon}}$ as the set of those $n-t$ honest servers, $\mathcal{S}_{\mathsf{Mal}}$ as the*

set of those t servers controlled by $\mathcal{A}$. We aim to simulate view transcript from the $\mathcal{S}_{\mathsf{Hon}}$ to the client.
Under this scenario, with the information $y = (x + k) \cdot s^2$, the $\mathcal{A}$ is able to compute $u = y - \sum_{T_1,T_2 \in \mathcal{T}_{\mathcal{S}_{\mathsf{Mal}}}} a_{T_1} b_{T_2}$ and provide it to the simulator.
We construct a simulator that works similarly as described in Algorithm 6. To argue about the privacy of the secret inputs k or s^2, it is sufficient for the simulator to simulate only the transcript view $\{(\hat{v}_i^{\bar{T}T_2}, \hat{h}^i_{\bar{T}T_2})_{T_2 \in \mathcal{T}_i}\}_{i \in \mathcal{S}_{\mathsf{Hon}}}$ and $\{(\hat{v}_i^{T_1\bar{T}}, \hat{h}^i_{T_1\bar{T}})_{T_1 \in \mathcal{T}_i}\}_{i \in \mathcal{S}_{\mathsf{Hon}}}$ where $\bar{T}$ refers to the special share index held only by each of the server in $\mathcal{S}_{\mathsf{Hon}}$. Let $Y = \{(\hat{v}^{\bar{T}T_2})_{T_2 \in \mathcal{T}_i}\}_{i \in \mathcal{S}_{\mathsf{Hon}}} \bigcup (\hat{v}^{T_1\bar{T}})_{T_1 \in \mathcal{T}_i}\}_{i \in \mathcal{S}_{\mathsf{Hon}}}$, if we generate a random $(2\binom{n}{t} - 1)$-party ASS of u and obtain $[u]$, we argue that this $[u]$ is perfectly indistinguishable to Y.

Algorithm 6 Designing a Simulator for Handling invalid RSS Provided by a Malicious Client

```
Input  : An invalid $[\![x]\!]_R$ from the client, $(\perp, r)$ from $\mathcal{F}_{\mathsf{RSS-OPRF^*}}$.
Output: $\{(\hat{v}_i^{T_1T_2}, \hat{h}_{T_1T_2})_{T_1,T_2 \in \mathcal{T}_i}\}_{i=1}^n$.
1 Samples $\hat{v}_i^{T_1T_2} \leftarrow_\$ \mathcal{U}$ for every $i \in \mathcal{S}_{T_1T_2}$ and every $T_1, T_2 \in \mathcal{T}$.
2 Computes $\hat{o}^{T_1T_2} = \sum_{i \in \mathcal{S}_{T_1T_2}} \hat{v}_i^{T_1T_2}$ for every $T_1, T_2 \in \mathcal{T}$ where $T_1 \neq T_*$.
3 Computes $\hat{h}^i_{T_1T_2} = \mathrm{Hash}_i(\hat{o}^{T_1T_2})$ for every $T_1, T_2 \in \mathcal{T}, T_1 \neq T_*$.
4 For every $T_2 \in \mathcal{T}$, samples a random element $\hat{h}^i_{T_*T_2}$ from the output
    domain of $\mathrm{Hash}_i$ for every $i \in \mathcal{S}_{T_*T_2}$.
5 return
    $\{(\hat{v}_i^{T_1T_2}, \hat{h}_{T_1T_2})_{T_1,T_2 \in \mathcal{T}_i}\}_{i=1}^n$.
```

4.3 Semi-honest OSS protocol

In this section, we describe one more protocol (algorithm 7) for a distributed OPRF based on optimised secret sharing (OSS). It provides an improvement in terms of threshold - i.e., it allows for a threshold $t < n$. However, it comes at a cost of requiring another round of communication between the servers and the client, which we set in the setup phase. In particular, the servers need to share a random value with the client before the client shares their input, which is r_x as shown in algorithm 7. In algorithm 7, for the setup stage we rely on the ideal functionality $\mathcal{F}_{\mathsf{OSS-Setup}}$ that provide required correlated randomness to enable our protocol design, particularly $\mathcal{F}_{\mathsf{OSS-Setup}}$ can be realized using any post-quantum crypto primitive that generate beaver's triple.

Round Efficiency Algorithm 7 takes two client-server communication rounds. No communication required among servers in the online phase.

Algorithm 7 OSS-based Legendre dOPRF in the semi-honest setting

```
Setup stage: Client; Server$S_i$, $i = 1, \dots, n$.
Input   : Servers input $[k]$ where $k \in \mathbb{F}_p$.
1 Servers run $\mathcal{F}_{\mathsf{OSS-Setup}}$ and obtain $[r_x]$, $[r_x]$, $[\hat{r}]$, $[s^2]$, $[(r_k + r_x)\hat{r}]$ where
  $r_k, r_x, s, \hat{r}$ are randomly selected from $\mathbb{F}_p$.
2 Servers reconstruct $[k + r_k]$ and $[s^2 + \hat{r}]$, hence obtain
      $\langle k \rangle = (k + r_k, [r_k]), [r_x], \langle s^2 \rangle = (s^2 + \hat{r}, [\hat{r}]), [(r_k + r_x)\hat{r}]$.
3 Servers distributes $[r_x]$ to the client.

Input stage: Client
Input   : $x \in \mathbb{F}_p$
4 Computes $\delta_x = x + \sum_{i=1}^{n} [r_x]_i$
5 return $\delta_x$ to server $S_i, i = 1, \dots, n$

Evaluation stage: Server $S_i$; $i = 1, \dots, n$
6 Computes
      $\langle x \rangle + \langle k \rangle = (x + r_x + k + r_k, [r_x + r_k])$
7 Let $\delta_{x+k} = x + r_x + k + r_k$ and $\delta_{s^2} = s^2 + \hat{r}$, computes
      $[(x + k) \cdot s^2] = (\delta_{x+k} - [r_x + r_k]) \cdot (\delta_{s^2} - [\hat{r}])$
                    $= \delta_{x+k}\delta_{s^2} - \delta_{s^2}[r_x + r_k] - \delta_{x+k}[\hat{r}] + [(r_k + r_x)\hat{r}]$
8 return $[(x + k) \cdot s^2]$ to client.

Reconstruction stage: Client
9 return $(x + k) \cdot s^2 = \sum_{i=1}^{n} [(x + k) \cdot s^2]_i$
```

Security Let $\mathcal{F}_{\mathsf{OSS-OPRF}}$ be the ideal functionality that inputs x from the client, s^2 and k from the server; outputs $(x+k)s^2$ to the client. In algorithm 7, note that either the client's transcript view δ_x or the additive secret shares of $(x + k)s^2$, obtained from up to $n - 1$ servers, adhere to a uniform distribution. This means a simulator can easily draw a random element from the corresponding domain, which would be perfectly indistinguishable to what in the real world execution. Thus, we claim in the $\mathcal{F}_{\mathsf{OSS-Setup}}$-hybrid model, a simulator Sim exists for all $x, k, s \in \mathbb{F}_p$ and the function $f : (x, k, s^2) \mapsto (x + k)s^2$, Sim realizes $\mathcal{F}_{\mathsf{OSS-OPRF}}$ in a way that ensures the simulated view transcript is perfectly indistinguishable from the real protocol execution in algorithm 7, when facing a semi-honest adversary $\mathcal{A}$ that controls up to $n - 1$ servers and the client.

4.4 Malicious OSS protocol

In a setting with up to $n-1$ malicious servers, which might not follow the protocol description, in order to be able to verify the correctness of the final protocol output, we need a different ideal functionality. We define $\mathcal{F}_{\mathsf{OSS-OPRF}^*}$ as the ideal functionality that inputs x from the client, s^2, k and auxiliary data $z \in \mathbb{Z}_N$ from the server. Here z indicates the servers' additional input in the real world protocol. We denote by W a PPT algorithm that gives an output depending on z and $(x+k)s^2$. The functionality $\mathcal{F}_{\mathsf{OSS-OPRF}^*}$ outputs $((x+k)s^2, \psi = 0)$ if $z = 0$ and $(W(z,(x+k)s^2), \psi = 1)$ otherwise when $z > 0$.

We employ the idea from the SPDZ framework [DPSZ11] that uses an ASS of an authenticated secret key α to authenticate every secret sharing managed by the servers, along with the authenticated optimized secret sharing that we have designed algorithm 8 for computing $\mathcal{F}^*_{\mathsf{OSS-OPRF}}$. In algorithm 8, for the setup stage we rely on the ideal functionality $\mathcal{F}_{\mathsf{OSS-Setup}^*}$ that provides more correlated randomness compared to $\mathcal{F}_{\mathsf{OSS-Setup}}$, however it can also be realized using only any post-quantum crypto primitive that generate beaver's triple.

Here, we clarify the notations used in algorithm 8. We define $[\![x]\!]_A$ of a secret x for which it holds:

$$[\![x]\!]_A := ([x], [\alpha x]),$$

$\langle\!\langle x \rangle\!\rangle$ denotes the authenticated optimized SS of a secret x for which it holds:

$$\langle\!\langle x \rangle\!\rangle := (x + r_x, [\alpha x], [\alpha r_x]).$$

In a scenario where each server $\mathcal{S}_i$ holds $[\![y]\!]_{A,i} = ([y]_i, [\alpha y]_i)$ for $i = 1, \ldots, n$, and the authenticated secret y is revealed either to one of the servers or a third-party client, the output receiver could verify the correctness of y by verifying if it holds:

$$\sum [\alpha]_i \cdot \sum [y]_i = \sum [\alpha y]_i \tag{2}$$

This requires each server $\mathcal{S}_i$ to return $[\alpha]_i, [y]_i, [\alpha y]_i$ to the output receiver.

In a slight different scenario, where if the servers hold multiple authenticated optimized SS $\langle\!\langle x_1 \rangle\!\rangle, \langle\!\langle x_2 \rangle\!\rangle, \cdots, \langle\!\langle x_m \rangle\!\rangle$, to verify the correctness of the public part within each $\langle\!\langle x_j \rangle\!\rangle, j \in 1, \cdots, m$, each server $\mathcal{S}_i$ could locally track the secret sharing of a proof Δ_i.

$$\Delta_i = [\alpha]_i \sum_{j=1}^{m} R_j \cdot \delta_j - \sum_{j=1}^{m} R_j \cdot [\alpha \cdot \delta_j] \tag{3}$$

where $R_1, \cdots, R_m$ are publicly random integers negotiated among servers.

Round Efficiency The communication pattern in 8 follows exactly the same structure as in algorithm 7, though more computations are required.

Algorithm 8 OSS-based Legendre dOPRF in the semi-honest client but malicious server setting

```
Setup stage: Client; Server$S_i$, $i = 1, \dots, n$.
Input : Servers input $[k], [\alpha]$ where $k, \alpha \in \mathbb{F}_p$.
1  Servers run $\mathcal{F}_{\mathsf{OSS-Setup}^*}$ with input $[k]$ and $[\alpha]$, and obtain
       $$[\![k]\!]_A, [\![r_k]\!]_A, [\![s^2]\!]_A, [\![\hat{r}]\!]_A, [\![r_0]\!]_A, [\![r_x]\!]_A, [\![(r_k + r_x)\hat{r}]\!]_A$$
   where $r_0, r_k, r_x, s, \hat{r}$ are randomly selected from $\mathbb{F}_p$.
2  Servers reconstruct $[k + r_k]$ and $[s^2 + \hat{r}]$, hence obtain
       $$\langle\!\langle k \rangle\!\rangle = (k + r_k, [\alpha k], [\alpha r_k]), \langle\!\langle s^2 \rangle\!\rangle = (s^2 + \hat{r}, [\alpha s^2], [\alpha \hat{r}]),$$
3  Servers distributes $[r_0], [r_x]$ to the client.
Input stage: Client
Input : $x \in \mathbb{F}_p$
4  Computes $\delta_0 = x + \sum [r_0]_i, \delta_x = x + \sum_{i=1}^n [r_x]_i$
5  return $\delta_0, \delta_x$ to server $S_i, i = 1, \dots, n$
Evaluation stage: Server $S_i$; $i = 1, \dots, n$
6  Computes $[\alpha x] = \delta_0[\alpha] - [\alpha r_0]$
7  Let $\langle\!\langle x \rangle\!\rangle = (x + r_x, [\alpha x], [\alpha r_x])$, then computes $\langle\!\langle x + k \rangle\!\rangle = \langle\!\langle x \rangle\!\rangle + \langle\!\langle k \rangle\!\rangle$.
8  Computes
       $$[\Delta] = [\alpha](R_0(x+k+r_x+r_k)+R_1(s^2+\hat{r}))-R_0([\alpha(x+k+r_x+r_k)])-R_1([s^2+\hat{r}])$$
9  Let $\delta_{x+k} = x + r_x + k + r_k$ and $\delta_{s^2} = s^2 + \hat{r}$, computes
       $$\begin{aligned}[(x + k)s^2] &= (\delta_{x+k} - [r_x + r_k]) \cdot (\delta_{s^2} - [\hat{r}]) \\ &= \delta_{x+k}\delta_{s^2} - \delta_{s^2}[r_x + r_k] - \delta_{x+k}[\hat{r}] + [(r_k + r_x)\hat{r}], \\ [\alpha(x + k)s^2] &= \delta_{x+k}\delta_{s^2}[\alpha] - \delta_{s^2}[\alpha(r_x + r_k)] - \delta_{x+k}[\alpha\hat{r}] + [\alpha(r_k + r_x)\hat{r}].\end{aligned}$$
10 return $[(x + k)s^2], [\alpha(x + k)s^2 + \Delta], [\alpha]$ to client.
Reconstruction stage: Client
11 Computes $y = \sum_{i=1}^n [(x + k)s^2]_i$, $\alpha = \sum_{i=1}^n [\alpha]_i$, and
   $\psi = \sum_{i=1}^n [\alpha((x + k)s^2) + \Delta]_i$.
12 return $y$ if $y\alpha = \psi$, otherwise returns $\perp$.
```

Client Verifiability In algorithm 8, if all involved parties honestly follow the protocol description, the correctness is obvious; However, in case of up to $n-1$ malicious servers controlled by an adversary $\mathcal{A}$, when $\mathcal{A}$ returns manipulated $[r_0]$ to the client, this error would finally result in a wrong computation of $[\alpha x]$ in the online phase; when $\mathcal{A}$ returns manipulated $[r_x]$ to the client or mishandles any computation in the evaluation phase, the final protocol output cannot be verified by the client except with probability $1-1/q$ where q is the size of $\mathbb{F}_p$. This is shown by the combination of equations 2, 3.

Security We claim that algorithm 8 securely realizes $\mathcal{F}_{\mathsf{OSS-OPRF}^*}$ in the $\mathcal{F}_{\mathsf{OSS-Setup}^*}$-hybrid model. This is equivalent to say there exists an simulator which inputs what provided by $\mathcal{A}$ and possibly ideal functionality output, that outputs a view transcript which is indistinguishable to that in real world execution. For this end, note that in algorithm 8, in each run of the protocol a new independent secret key α is used, thus, statistically, with protocol output $(x+k)s^2$ as the simulator input, if the simulator generates a random simulated $\hat{\alpha} \leftarrow_\$ \mathbb{F}_p$ and then generates a n-party ASS $[(x+k)s^2]^*$, subsequentially compute $[\alpha^*(x+k)s^2]$, we argue the simulated view $([(x+k)s^2]^*, [\alpha^*(x+k)s^2])$ is perfectly indistinguishable to the real world protocol output. On the other hand, notice that a honest client distributes δ_0, δ_x to the servers in the real world protocol, which are both masked values of same secret x, thus, this real world view transcript is perfect indistinguishable to a simulated view including two random numbers $\delta_0^*, \delta_x^* \leftarrow_\$ \mathbb{F}_p$.

4.5 On partial obliviousness

A partially oblivious PRF 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 ([CHL22], Section 3.1) to construct a partially oblivious PRF (pOPRF) from an OPRF and could also be employed to transform also our proposed dOPRF constructions. 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'.

The drawbacks of this approach is that it is dependent on the secret sharing of the long-term key k, and a different secret sharing $[\![k]\!]_R + [\![0]\!]_R$ leads to a different function.

5 Evaluation

We implemented and benchmarked protocols from Section 4.1 and Section 4.2. Our implementation can be found on

https://github.com/For-anonymous-submissions/Legendre-dOPRF.

Our tests were conducted on an Apple M2 ARM processor with a clock speed of 3.49 GHz. The protocol was developed in C language and compiled using gcc with the `-O3` optimization level. We offer two versions of the implementation: a standard one in C and a high-speed version with field arithmetic coded in assembly. The implementation ensures constant time and memory access, and it does not involve any branching.

Our results are listed in Table 2, where the upper half is the result for the semi-honest protocol in Algorithm 3 and the bottom half for the malicious protocol in Algorithm 4. The procedure of calculating the Legendre symbol, denoted as `leg_symbols`, was benchmarked separately from the reconstruction stage. Both implementations fare well for post-quantum standards, but scale exponentially in t and therefore n when the memory use becomes a bottleneck.

p	t	n	`input`	`evaluation`	`reconstruction`	`leg_symbols`
128 bits	1	3	0,001 ms	0,011 ms	0,001 ms	0,20 ms
	2	5	0,005 ms	0,112 ms	0,002 ms	
	3	7	0,020 ms	1,500 ms	0,003 ms	
192 bits	1	3	0,001 ms	0,033 ms	0,002 ms	0,75 ms
	2	5	0,005 ms	0,300 ms	0,006 ms	
	3	7	0,020 ms	3,300 ms	0,010 ms	
256 bits	1	3	0,001 ms	0,075 ms	0,005 ms	2,22 ms
	2	5	0,005 ms	0,650 ms	0,011 ms	
	3	7	0,020 ms	7,280 ms	0,017 ms	
p	t	n	`input`	`evaluation`	`reconstruction`	`leg_symbols`
128 bits	1	4	0,003 ms	0,047 ms	0,042 ms	0,20 ms
	2	7	0,012 ms	0,960 ms	1,700 ms	
	3	10	0,068 ms	29,312 ms	90,88 ms	
192 bits	1	4	0,002 ms	0,106 ms	0,120 ms	0,75 ms
	2	7	0,013 ms	2,592 ms	4,400 ms	
	3	10	0,069 ms	85,248 ms	211,77 ms	
256 bits	1	4	0,002 ms	0,259 ms	0,238 ms	2,20 ms
	2	7	0,013 ms	6,688 ms	8,304 ms	
	3	10	0,70 ms	202,496 ms	366,33 ms	

Table 2: Benchmarks for Legendre dOPRF, Algorithm 3 and 4.

Bibliography

[ADDG23] 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. `https://eprint.iacr.org/2023/232`. (page 2)

[ADDS19] 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. `https://eprint.iacr.org/2019/1271`. (page 2, 5, 6)

[AMMM18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. Pasta: Password-based threshold authentication. Cryptology ePrint Archive, Paper 2018/885, 2018. `https://eprint.iacr.org/2018/885`. (page 2)

[art07] Unapproved IEEE Draft Standard for Specifications for Password Based Public Key Cryptographic Techniques. *IEEE Unapproved Std P1363.2 /D27*, 2007. (page 2)

[art17] 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.html16`. (page 2)

[Bas23] Andrea Basso. A post-quantum round-optimal oblivious prf from isogenies. Cryptology ePrint Archive, Paper 2023/225, 2023. `https://eprint.iacr.org/2023/225`. (page 2, 5)

[BBUV19] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. Cryptanalysis of the legendre prf and generalizations. Cryptology ePrint Archive, Report 2019/1357, 2019. `https://eprint.iacr.org/2019/1357`. (page 9)

[BBY20] 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. `https://eprint.iacr.org/2020/1577`. (page 11, 36)

[BDFH24] 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. `https://eprint.iacr.org/2024/450`. (page 2, 5, 6)

[BENO19] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: double your online spdz! improving spdz using function dependent preprocessing. In *International Conference on Applied Cryptography and Network Security*, pages 530–549. Springer, 2019. (page 13)

[BFH$^+$19] 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. `https://eprint.iacr.org/2019/1470`. (page 2, 6)

[BKM$^+$] Andrea Basso, Pêter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. Cryptanalysis of an oblivious prf from supersingular isogenies. In *Advances in Cryptology – ASIACRYPT 2021*. (page 2)

[BKW20] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. Cryptology ePrint Archive, Paper 2020/1532, 2020. `https://eprint.iacr.org/2020/1532`. (page 2, 5, 6)

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005. (page 10)

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. Cryptology ePrint Archive, Paper 2022/302, 2022. `https://eprint.iacr.org/2022/302`. (page 5, 30)

[CLM$^+$18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: An efficient post-quantum commutative group action. Cryptology ePrint Archive, Paper 2018/383, 2018. `https://eprint.iacr.org/2018/383`. (page 5)

[Dam90] 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. (page 8)

[DGH$^+$] 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*. (page 2)

[DHL20] Poulami Das, Julia Hesse, and Anja Lehmann. Dpase: Distributed password-authenticated symmetric encryption. Cryptology ePrint Archive, Paper 2020/1443, 2020. `https://eprint.iacr.org/2020/1443`. (page 2)

[Dod23] 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`. (page 2, 5, 6)

[DPSZ11] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Paper 2011/535, 2011. `https://eprint.iacr.org/2011/535`. (page 13, 28)

[Esc22] Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Paper 2022/062, 2022. https://eprint.iacr.org/2022/062. (page 10)

[Fei19] Dankard Feist. Legendre pseudo-random function, 2019. https://legendreprf.org/bounties. (page 8)

[FIPR05] 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. (page 1)

[FJP11] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Paper 2011/506, 2011. https://eprint.iacr.org/2011/506. (page 5)

[FOO23] Sebastian Faller, Astrid Ottenhues, and Johannes Ottenhues. Composable oblivious pseudo-random functions via garbled circuits. Cryptology ePrint Archive, Paper 2023/1176, 2023. https://eprint.iacr.org/2023/1176. (page 2, 6)

[FS21] Paul Frixons and André Schrottenloher. Quantum security of the legendre prf. Cryptology ePrint Archive, Paper 2021/149, 2021. https://eprint.iacr.org/2021/149. (page 9)

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996. (page 6)

[GRR$^+$16] 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. (page 8)

[HL09] 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. https://eprint.iacr.org/2009/045. (page 2, 6)

[JAC$^+$17] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation, 2017. https://sike.org/. (page 5)

[JBSL10] Stanislaw Jarecki, Ali Bagherzandi, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. Cryptology ePrint Archive, Paper 2010/561, 2010. https://eprint.iacr.org/2010/561. (page 2, 6)

[JKK14] Stanislaw 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. https://eprint.iacr.org/2014/650. (page 2, 6)

[JKKX17] Stanislaw 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. `https://eprint.iacr.org/2017/363`. (page 2, 6)

[JL09] Stanisław Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. 2009. (page 2, 6)

[JL10] Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. 2010. (page 2)

[Kel20] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. Cryptology ePrint Archive, Paper 2020/521, 2020. `https://eprint.iacr.org/2020/521`. (page 5)

[Kho19] Dmitry Khovratovich. Key recovery attacks on the legendre prfs within the birthday bound. Cryptology ePrint Archive, Paper 2019/862, 2019. `https://eprint.iacr.org/2019/862`. (page 9)

[KKK20a] Novak Kaluđerović, Thorsten Kleinjung, and Dusan Kostic. Improved key recovery on the legendre prf. Cryptology ePrint Archive, Paper 2020/098, 2020. `https://eprint.iacr.org/2020/098`. (page 9)

[KKK20b] Novak Kaluđerović, Thorsten Kleinjung, and Dušan Kostić. Cryptanalysis of the generalised Legendre pseudorandom function. ANTS XIV, 2020. `https://msp.org/obs/2020/4-1/p17.xhtml`. (page 9)

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. Cryptology ePrint Archive, Paper 2016/799, 2016. `https://eprint.iacr.org/2016/799`. (page 2)

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. Cryptology ePrint Archive, Paper 2016/505, 2016. `https://eprint.iacr.org/2016/505`. (page 5)

[Mau06] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154:370–381, 2006. (page 10)

[SHB21] István András Seres, Máté Horváth, and Péter Burcsi. The legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications. Cryptology ePrint Archive, Paper 2021/182, 2021. `https://eprint.iacr.org/2021/182`. (page 9)

[ss-87] Secret sharing schemes realizing general access structures. In *Proc. IEEE Global Telecommunication Conf., Globecom 87*, Lecture Notes in Computer Science, 1987. `https://archiv.infsec.ethz.ch/education/as09/secsem/papers/ItSaNi87.pdf`. (page 10)

[vDH00] Wim van Dam and Sean Hallgren. Efficient quantum algorithms for shifted quadratic character problems, 2000. (page 9)

[Wu00] Thomas Wu. The SRP authentication and key exchange system. RFC, Internet Engineering Task Force, 2000. `https://www.rfc-editor.org/rfc/rfc2945`. (page 2)

A The setup phase for protocol 3 and 4

In the setup phase of the protocol, the servers compute values that are used later on in the online phase. We show how these values are computed. In particular the servers need to compute

- A random RSS key $[\![k]\!]_R$,
- A random RSS square $[\![s^2]\!]_R$,
- Various secret sharings of zero $[r], [\![r]\!]_R, [\![r]\!]_D$.

A.1 Generating random values

A random additive secret share $[s]$ can be generated by each party generating a random value locally. For RSS (DRSS) the setting is a bit different because shares s_T have to be equal for all parties in $\mathcal{S}_T$ (respectively for DRSS). This can be achieved either through communication, or on-the-go by using a pseudorandom function.

Assume that the parties hold a pre-computed (D)RSS key K_T and agree on a PRF F. Then, they can compute (D)RSS shares by evaluating F on the same input

$$s_T = F_{K_T}(x),$$

where x is some input known by all servers. In particular, from a single shared value x the parties can generate as many random RSS shares $[\![r^i]\!]_R$ as they wish by computing $(s^i_T)_{T\in\mathcal{T}} = (F_{K_T}(x||i))_{T\in\mathcal{T}}$.

A.2 Generating an RSS square

The usage of the square term s^2 in hiding the output of the Legendre PRF is essential because it masks the value of the output. The term s^2 is always used as an RSS sharing, and to that end we show how to compute a multiplication of two RSS values in a secure and verifiable manner.

Let $[\![a]\!]_R, [\![b]\!]_R$ be two RSS shares. This protocol is done in two steps. First, the parties compute an RSS share $[\![c]\!]_R$ of $a \cdot b$ with a protocol which only provides semi-honest security ([BBY20], Protocol 1), and confirm the validity of the share. Then they show that $c = a \cdot b$ by generating an RSS share of 1, computing $[\![c]\!]_R, [\![1]\!]_R \mapsto [\![c \cdot 1]\!]_D$ and $[\![a]\!]_R, [\![b]\!]_R \mapsto [\![a \cdot b]\!]_D$. They verify that $[\![c - ab]\!]_D = [\![c \cdot 1]\!]_D - [\![a \cdot b]\!]_D$ is indeed the DRSS share of zero by opening.

A.3 Computing secret sharings of zero

For completeness recall methods for computing ASS/RSS/DRSS secret sharings of zero. Let $\dot{\mathcal{T}} \subseteq \{1, \ldots, n\}$ (ASS) or $\dot{\mathcal{T}} = \mathcal{T}$ (RSS) or $\dot{\mathcal{T}} = \mathcal{T}^2$ (DRSS) be a set of share indices.

Lemma 1 (Secret sharings of 0 from a random element). *Assume that for each pair of shares $\dot{T}_1 \neq \dot{T}_2 \in \dot{\mathcal{T}}$ the servers holding $\dot{T}_1$ or $\dot{T}_2$ hold a 2-party additive secret share of zero known only to them, $r_{\dot{T}_1\dot{T}_2}, r_{\dot{T}_2\dot{T}_1} \in \mathbb{F}_p$ such that*

$$r_{\dot{T}_1\dot{T}_2} + r_{\dot{T}_2\dot{T}_1} = 0.$$

Then they can compute a valid secret sharing of zero.

Proof. *The servers compute*

$$r_{\dot{T}} := \sum_{\dot{T}_2 \in \dot{\mathcal{T}} \setminus \{\dot{T}\}} r_{\dot{T}\dot{T}_2}.$$

Furthermore

$$\sum_{\dot{T} \in \dot{\mathcal{T}}} r_{\dot{T}} = \sum_{\dot{T} \in \dot{\mathcal{T}}} \sum_{\dot{T}_2 \in \dot{\mathcal{T}} \setminus \{\dot{T}\}} r_{\dot{T}\dot{T}_2} = \frac{1}{2} \sum_{\dot{T} \neq \dot{T}_2 \in \dot{\mathcal{T}}} r_{\dot{T}\dot{T}_2} + r_{\dot{T}\dot{T}_2} = 0.$$

Therefore the secret sharing $(r_{\dot{T}})_{\dot{T} \in \dot{\mathcal{T}}}$ is a secret sharing of 0.

The servers that wish to compute a secret sharing of zero on-the-go in a comunicationless manner can do so assuming that they hold a pre-shared random string $s_{\{\dot{T}_1,\dot{T}_2\}}$ for each $\dot{T}_1 \neq \dot{T}_2 \in \dot{\mathcal{T}}$, known by each server holding $\dot{T}_1$ or $\dot{T}_2$. These strings can be obtained via public key infrastructure from distributed public-secret keypairs $(pk_{\dot{T}}, sk_{\dot{T}})_{\dot{T} \in \dot{\mathcal{T}}}$ (again relying on a shared random string and broadcasted public keys) distributed as a secret sharing to the servers. Once a set of pre-shared random strings is established, the servers can agree on a public pseudorandom function $F_k(\cdot)$ and use it to generate random elements (equiv. random 2-party additive secret sharings) as

$$r_{\dot{T}_1\dot{T}_2} = (-1)^{(\dot{T}_1 < \dot{T}_2)} F_{s_{\{\dot{T}_1,\dot{T}_2\}}}(a),$$

where a is any input, and "<" any strict total order on $\dot{\mathcal{T}}$. This approach, while communicationless, reduces the security from information theoretic to computational due to the use of the pseudorandom function.