Threshold Oblivious Pseudorandom Functions from Isogeny Group Actions

Robi Pedersen

Technical University of Denmark
robi.pedersen@protonmail.com

Abstract. We present a new verifiable oblivious pseudorandom function (VOPRF) from isogeny group actions. Our construction is twice as fast as the previous state of the art of Delpech de Saint Guilhem and Pedersen at a slightly higher communication cost. One major contribution is the realization of a new proof protocol that is integrated as a two-party computation into the OPRF protocol, making the output verifiable. The main design choice behind our construction and this new proof system is to enable an easy transformation into a threshold protocol, something previous designs have not achieved. To this end, we present our VOPRF in a modular way based on different subroutines. We show how to replace these subroutines with their threshold counterparts, using simulation-based arguments. This results in the first threshold VOPRF from isogenies and one of the first threshold VOPRFs in the post-quantum literature. In contrast to other post-quantum threshold VOPRF designs, our construction has input and output size independent of the number of server parties and furthermore is robust, while other designs rely on aborts in the presence of malicious parties.

Keywords: Oblvious pseudo-random functions, threshold schemes, isogeny-based cryptography, effective group actions

Acknowledgments

The author would like to thank Madhi Sedaghat for bringing the topic of threshold OPRFs to his attention, and the anonymous reviewers for their constructive comments on earlier versions of this paper. The author is thankful to Carsten Baum for his invaluable help with the security proofs in this paper. This work was supported by Research grant VIL53029 from VILLUM FONDEN.

This article is the full version of a paper accepted at PKC 2026.

Table of Contents

Threshold Oblivious Pseudorandom Functions from Isogeny Group Actions 1
Robi Pedersen
1 Introduction . . . 3
1.1 Technical overview and contributions . . . 4
1.2 Related works . . . 6
2 Background . . . 10
2.1 Effective group actions . . . 10
2.2 Verifiable secret sharing . . . 11
2.3 Distributed group action computation . . . 13
3 New (verifiable) oblivious pseudorandom function . . . 14
3.1 Ideal functionality . . . 14
3.2 The pseudorandom function . . . 15
3.3 Preprocessing phase . . . 17
3.4 Proving correct output . . . 18
3.5 Subalgorithms . . . 19
3.6 (Verifiable) Oblivious Pseudorandom Function . . . 20
3.7 Security proof of our VOPRF . . . 20
4 Thresholdization . . . 26
4.1 Malicious threshold VOPRF . . . 28
4.2 Proof in the malicious threshold . . . 34
4.3 A note on verifiability and trust . . . 37
5 Instantiation and costs . . . 38
5.1 Initial considerations . . . 38
5.2 Honest-client instantiation from CSI-FiSh . . . 38
5.3 Instantiation with prime order acting groups . . . 39
5.4 Cost overview . . . 43
A Preliminaries on distributed protocols from isogenies . . . 54
A.1 The threshold zero-knowledge proof from Π . . . 54
A.2 Structured Piecewise Verifiable Proofs . . . 55
B Semi-honest threshold OPRF . . . 56

1 Introduction

The idea of deterministic functions that generate outputs indistinguishable from random goes back to Goldreich, Goldwasser and Micali [55], who first described the concept of *pseudorandom functions* (PRF). In the public-key setting, pseudorandom functions are protocols jointly executed by a client and a server, in which the client sends an input m to the server, which evaluates a keyed pseudorandom function F_k on m and returns the output $y \leftarrow F_k(m)$ to the client. Security of such a PRF relies on the idea that clients should not be able to generate outputs of F_k for values not queried to the server. While pseudorandom functions by themselves find many useful applications in cryptography, endowing them with extra properties really opens up a world of possibilities.

One particular property suggesting itself in a two-party setting is that of *verifiability*, which allows the client to check whether the PRF output has been computed correctly. This is generally achieved by the server publishing some commitment (public key) to the secret key k that can be used by the client to verify the output. Additionally, this also allows clients to check whether the same key has been used in multiple evaluations of the function. Such functions are called verifiable random functions (VRF). VRFs have first been introduced by Micali, Rabin, and Vadhan [74] and have found applications in DNSSEC [54], key escrow [64], randomness beacons [32, 84] as well as in blockchain consensus algorithms [30, 40].

Another interesting property is that of *obliviousness*, in which the server does not learn any information about the client's input m, as well as the actual output y of the evaluation. Such *oblivious pseudorandom functions* (OPRF) were first presented by Naor and Reingold [75] and later formalized by Freedman, Ishai, Pinkas and Reingold [51]. Although introduced in the latter as a means to allow private information retrieval, OPRFs have been found to be building blocks of so many different applications that they have deservedly become a staple of modern cryptography. Examples range from private set intersection [56, 63, 68] through password-authenticated key exchange [59, 62] all the way to privacy-preserving pattern matching [48, 56], to name only a few. In many of these contexts, verifiability of the OPRF outputs, similar to VRFs, is also desirable or even necessary. A systematization of knowledge treating OPRFs has been drafted a bit over two years ago [28].

Threshold (V)OPRFs. More recently, more exotic properties of pseudorandom functions have found their way into the literature. One such property that we are going to explore throughout this work is the *thresholdization* of the server-side execution. The idea behind threshold protocols has its foundation in the idea of mitigating the risks with respect to key leakage. In a threshold system for a given threshold t, the secret key sk is held by multiple parties in the form of secret shares, in such a way that more than t parties need to combine their shares in order to learn the secret key sk or perform operations that use it, while shares from t or less parties would not be enough to learn anything about sk. Thus, an attacker corrupting up to t of these parties would still not be able to evaluate

the underlying PRF. Threshold OPRFs were initially introduced and formalized in [46, 60, 61] and find applications in distributed protocols, such as threshold private set intersection [72], password-based threshold authentication [1] and distributed password-authenticated encryption [39], as well as password-protected secret sharing [14,39,60]. We will discuss some more recent constructions as part of the related works in Section 1.2.

Post-quantum security. A different property of great interest for oblivious pseudorandom functions in a public-key setting is post-quantum security. With the advance of ever growing quantum computers able to break classical cryptographic problems, such as the integer factorization and discrete logarithm problems [88], it is of great interest to find constructions that elude these attacks. Approaches to post-quantum secure OPRFs are rather new and are constrained to about the last 5 years in research. Post-quantum secure OPRFs have been proposed from MPC assumptions [3, 21, 47, 92] as well as lattices [4, 5, 45]. But also isogenies have been explored with the same goal: [15] presents an OPRF based on the previously broken [24] in the general isogeny setting. In the more flexible (but slower) group action setting, [24] and [57] proposed a Naor-Reingold type OPRF. Later, [44] proposed a much more compact way to realize OPRFs from group actions than Naor-Reingold based approaches do.

1.1 Technical overview and contributions

In this paper, we present a post-quantum secure verifiable oblivious pseudorandom function construction from isogeny group actions. Our constructions are based on the framework first introduced in [44] in which a client and a server jointly evaluate a function "in the exponent" of a group action. Let

$$[\cdot] : \mathcal{G} \times \mathcal{E} \to \mathcal{E}$$

define a group action, where $\mathcal{G}$ is an additive group. The idea of [44] was to build a pseudorandom function in which a server holds a secret monovariate polynomial $f(X)$ with coefficients in $\mathcal{G}$ and the client holds an input message m, also in $\mathcal{G}$. Jointly, the client and server then evaluate the group action $[f(m)]E_0$. The degree d of the polynomial $f(X)$ ultimately depends on the query access we provide to the functionality. The authors in [44] argue that linear polynomials already fulfill the necessary pseudorandomness guarantees, but that if provided quantum access to such a functionality, an attacker could recover the secret polynomial. By using permutation polynomials of degree 3, one gets the same pseudorandomness guarantees as in the linear case and mitigates this attack vector, but at higher computation and communication costs. The quadratic case also prevents quantum superposition attacks, but uses more heuristic arguments for pseudorandomness, at a lower cost than $d = 3$. Initially designed as an OPRF, the same underlying pseudorandom function was also later used in [71] to construct a VRF.

To realize the OPRF in [44], the authors make use of Beaver-style triples [17], allowing the two parties to combine their shares multiplicatively in an additive

group. The authors let a trusted third party (TTP) generate these tuples, but also discuss how to remove the TTP by employing standard ways to generate such Beaver triples in a maliciously secure environment. To allow the verification of OPRF outputs, the TTP additionally has to generate commitments for the server-side triple elements e.g. $[a]E$ as a commitment to an element a. Although this achieves verifiability in a simple fashion, removing the TTP is not as straightforward anymore, and is also not investigated in [44]. We note that the VRF construction in [71] does not need such a TTP.

As our first main contribution, we introduce a new PRF construction that can be seen as a simplified variant of the polynomial evaluation PRF underlying [71] and [44]. Instead of polynomial evaluation, our PRF simply performs a multiplication in the exponent of an elliptic curve, e.g. $[\mathsf{sk} \cdot m]E$ for a server held secret key sk and a client's input m. We then build an OPRF on top of this multiplicative PRF with security relying on a one-more type assumption. In order to make the outputs of the OPRF verifiable, we introduce a new two-party proof system to convince the client that the output is produced correctly. Unlike [44], this proof system does not rely on commitments generated by a trusted third party. In our setting, commitments to Beaver triple values would actually allow an attacker to break one-more unpredictability of our scheme. Furthermore, getting rid of these commitments has another important upside: the setup now only needs standard Beaver-type triples. As a consequence, the TTP can be removed in the verifiable case using standard techniques, such as MASCOT [67] and OT extension [58, 66], which is not achieved by the construction in [44]. As an additional boon, our construction is also twice as fast. The downside of this modification is that our OPRF has a limit on the number of allowed queries per key. We present trade-offs between parameter sizes and the number of available queries. We prove security of our VOPRF in the universal composability (UC) framework [27].

We present our VOPRF in a very modular way by using placeholder algorithms for sampling elements from the group $\mathcal{G}$, computing linear combinations of elements in $\mathcal{G}$ and for the computation of the group action. We can easily instantiate these algorithms in the single-party case, but ultimately, we take this approach as it allows us to replace these subalgorithms with their distributed counterparts in a threshold setting. As our second main contribution (Section 4), we introduce threshold algorithms to instantiate these subalgorithms in the presence of malicious adversaries (we also outline the semi-honest case in Appendix B). We show that these algorithms correctly *realize* the single-party functionalities. We prove this by having a simulator $\mathcal{S}$ interact with the ideal single-party functionality and then engaging with an adversary in such a way that the multi-party protocol results in the same output as the single-party protocol, without the adversary knowing whether it is interacting with the simulator or engaging with the real protocol. Using such a simulation-based approach, we can port the properties realized by the single-party case to the multi-party setting.

To truly realize a single-party functionality, our threshold protocols need to be robust, i.e. have guaranteed correct outputs. In the semi-honest case, this is

a given, and so we can simply use Shamir secret sharing to sample elements in a threshold way and are certain to always reconstruct them or compute group actions correctly. In the malicious case, however, while robustness of reconstruction and group action computation is possible in this case (see e.g. [10, 20]), it generally needs commitments to the shares held by the individual parties. Unfortunately, this would bring us back to needing trusted commitments for the triple elements. Since these cannot be efficiently removed, we instead use sharing from bivariate polynomials as introduced in [18]. As a stepping stone towards our goal, we design a new bivariate verifiable secret sharing (VSS) protocol, which combines techniques from the bivariate VSS by Backes, Kate and Patra [11] and the distributed zero-knowledge proof system Π by Baghery [12], optimized to our setting. In particular, our VSS allows the linear combination of secret shared elements and their robust reconstruction. We use a similar technique to extend the distributed group action computation from [10] to the case of linear combinations of secrets. We note that the robust reconstruction of linear combinations of secrets in the presence of malicious adversaries bounds the threshold to a honest supermajority, i.e. $t < n/3$, in the absence of commitment schemes, due to byzantine faults, while our semi-honest construction is secure under a honest-majority assumption $t < n/2$.

1.2 Related works

We first compare our base (V)OPRF to other existing constructions from isogenies. Given that we propose the first threshold (V)OPRF from isogenies, we will compare our threshold construction to other quantum-secure threshold (V)OPRFs present in the literature.

OPRFs from isogenies. The construction of OPRFs from isogenies has been initiated by Boneh, Kogan and Woo [24], who introduced two OPRF constructions, one from SIDH-based assumptions, and one in the group-action setting based on a Naor-Reingold type pseudorandom function. The former one was broken by the SIDH attacks [29, 73, 85], but subsequently fixed by Basso [15] using a masked torsion point approach [50] and further protection against the active attacks outlined in [16]. The result is the currently only (verifiable) OPRF in the general (i.e., not group action) setting of isogenies. On the group action side of things, Heimberger et al. [57] generalized the construction from [24] to the restricted effective group action setting and removed the need of a trusted setup, while also reducing the cost. In the same year, Delpech de Saint Guilhem and Pedersen [44] designed an OPRF on a new underlying group-action based PRF. This resulted in a much more efficient evaluation and much more compact outputs, although at the cost of reintroducing a setup phase, and the need for effective group actions. Their construction is also the first one from group actions that can be made verifiable. We summarize existing constructions in the group action setting in Table 1 and compare them along different metrics. We note that a direct comparison to Basso's OPRF in the general setting is less straightforward, in particular in terms of computational costs, which are not explicitly

provided in [15]. However, the communication cost at NIST-I security amounts to 8.7 MB of data transferred between the server and client, according to [15]. As a comparison, at the same security level using a KLaPoTi [80] instantiation, our OPRF needs 124 kB of communication.

Table 1: Overview of existing OPRFs from isogeny group actions in terms of number of rounds, group action computations and total communication between both parties. We compare semi-honest (◐) and malicious (●) clients as well as trusted (◐) and verifiable (●) server outputs. We also indicate whether the OPRF needs a setup phase or an effective group action (EGA). For [44], we only consider the (cheaper) linear variant as it gives the same security guarantees as our protocol. The constant c in the communication costs indicates whether elliptic curves are oriented by Frobenius ($c = 1$) or more generally ($c = 6$), while $q = |\mathcal{G}|$ is the class number of the acting group.

	Setup Needed	EGA Needed	Model Client-Server	Online Rounds	Group Actions	Communication
[57]	No	No	◐-◐	$2\lambda + 2$	$3\lambda + 3$	$(3\lambda + 2)\log p$
[24]	Yes	Yes	◐-◐	2	$5\lambda + 2$	$(2\lambda + 1)\log p + 2\lambda^2$
			●-◐	4	$11\lambda + 2$	$(5\lambda + 1)\log p + 5\lambda^2$
[44]	Yes	Yes	●-◐	2	2	$2\log q + c\log p$
			●-●	2	$4\lambda + 2$	$2\lambda\log q + O(\lambda)$
Ours	Yes	Yes	●-◐	2	1	$2\log q + c\log p$
			●-●	2	$2\lambda + 1$	$\frac{7}{2}\lambda\log q + O(\lambda)$

Post-quantum secure threshold OPRFs. We compare our results with other threshold post-quantum secure OPRFs. Currently, to our knowledge, there are exactly three constructions that fit this bill, two of which are based on lattice assumptions [5, 89] and one on the Legendre PRF [65]. We discuss these constructions below and emphasize key differences and trade-offs between them and our design.

We start with the TOPRF from [89], which uses threshold fully homomorphic encryption (FHE) to evaluate AES as a PRF. Their construction is secure only in a semi-honest setting with corruptions of up to $n - 1$ parties and does not verify the outputs.[1] For 128-bit classical security, the communication between the server parties scales linearly with the threshold t as $(2.25t + 0.63)$MB.

[1] We note that verifiability in a threshold setting is not strictly necessary as it can be implied by the server assumptions, i.e. one could assume that when a protocol succeeds, that the output is correct based on the server-side protocol guarantees. We discuss this assumption further in Section 4.3.

The OPRF from [65] is based on the Legendre PRF [37]. The authors present constructions for semi-honest ($t < n/2$) as well as malicious ($t < n/3$) adversarial assumptions. A key particularity of these constructions is that the client secret shares their input m among the server parties. Each server then computes a part of the output and sends it back to the client, who recombines the shares into an output, which it then verifies. Verification is performed on the reconstructed value, and the client aborts if this fails. There are only two rounds of communication and no interaction between the servers. An interesting consequence of the way the input is shared is that in case of a break (i.e. where more than t server parties are corrupted), server parties would not only leak the shared key, but also additionally the client's input and verifiability of the outputs would also not be guaranteed anymore (the client would believe to have a correct output even when it is not). This peculiarity is not present in the other designs mentioned here, including our own.

Finally, the authors from [5] build upon the recent RLWE-based OPRF from [4] and show that a full-threshold TOPRF immediately follows from the key-homomorphic properties of the underlying VOPRF. To achieve a threshold t lower than n, they output $\binom{n}{t+1}$ replicated copies of a full-threshold. In the latter case, they also need to assume a trusted setup which generates these shares using a rejection sampling approach. Leveraging the homomorphic properties, the servers also do not need to communicate during executions and reconstruction is simply done on the client side. We note that [5] has a limit on the number of queries, depending on the parameters, and their scheme also only achieves security with abort.

We point out the main contrasts with our construction and give an overview of the main differences in Table 2. We first note that while the client-server communication is round-optimal in our TOPRF, we fundamentally need interaction between the server parties, and the number of rounds on the server side scale with the number of parties. This is due to the fact that we cannot combine elliptic curves in a meaningful way and is a fundamental limitation of isogeny-based cryptography (see also [35] on this topic). On the plus side, however, both the computation cost of the client and the input and output sizes are independent of the server parties, while in the other designs, the client communicates with each server individually and aggregates output from n servers. We note that the OPRF input and output messages scale linearly with the security parameter λ while the verifiable version scales with λ^2, which is considerably lower than the other designs. We directly compare communication costs between our scheme and [5, 65] in Table 3. We provide more details on costs in Section 5, where we also give estimates of the server-side internal communication. Our construction has bounds on corruptions inherited from Shamir ($t < n/2$ in the semi-honest setting) and bivariate secret sharing ($t < n/3$ in the malicious case). A key property resulting from these sharings is that our design achieves robustness, while [5, 65] have to rely on security with abort. We also note that all three schemes need some form of setup.

Table 2: Comparison of different aspects of our verifiable threshold OPRF with the constructions in [5, 65], assuming malicious corruptions in a (t, n)-sharing.

	Security	Setup phase	Query limit	Sharing	Server rounds	Verifier cost	Verifiability
[65]	Decisional Shifted Legendre Symbol Problem	Yes	**No**	Full or Replicated ($t < n/3$)	**1** (client-side aggregation)	$O(n)$	Abort
[5]	Decision Ring Learning With Errors	Yes (needs trust for $t < n - 1$)	Yes	Full or Replicated	**1** (client-side aggregation)	$O(n)$	Abort
Ours	Problem 3	Yes	Yes	Bivariate Polynomials ($t < n/3$)	n (round-robin)	$\mathbf{O(1)}$	**Robust**

Table 3: Communication costs between client and server(s) of our threshold VOPRF, compared to its Legendre [65] and Lattice [5] counterparts. For the Legendre OPRF, we use the 128-bit classical (85-bit quantum) secure numbers from [65, Table 4], whereas for the lattice instantiation, we consider the 100- and 167-bit instantiations with supported queries $Q = 2^4$ and $Q = 2^{64}$ from [5, Table 2], respectively. For the latter, we give the numbers of the full-threshold and (t, n)-threshold, which scale linearly in the number of parties (*: we omit a constant term of 1.8 kB in the threshold case). The numbers for our VOPRF are instantiated with KLaPoTi parameters (see the discussion in Section 5), for 128-bit classical security and both an optimistic and a conservative variant of NIST-I security. Note that our communication cost is independent of the number of server parties.

	Client → Server (Input)	Server → Client (Output)
[65], $(t, n) = (1, 4)$	96 B	144 kB
[65], $(t, n) = (2, 7)$	480 B	6.3 MB
[65], $(t, n) = (3, 10)$	2.6 kB	248 MB
[5], $Q = 2^4$	$\{n, 2t\}\times$(141.7 kB)	$\{n, t\}\times$(42.8 kB)*
[5], $Q = 2^{64}$	$\{n, 2t\}\times$(257.6 kB)	$\{n, t\}\times$(58.3 kB)*
Ours, classical	4.4 kB	10.9 kB
Ours, NIST-I (optimistic)	16.5 kB	45.7 kB
Ours, NIST-I (conservative)	33.0 kB	91.4 kB

2 Background

We write $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ for the ring of integers modulo q and $\mathbb{Z}_q[X]_d$ ($\mathbb{Z}_q[X,Y]_{(d,d)}$) for the ring of monovariate (bivariate) polynomials of degree d (in each variable) with coefficients in $\mathbb{Z}_q$. We write λ for the security parameter and use the underline notation $\underline{S}$ to define vectors $(S_1, \ldots, S_\lambda)$ of fixed length λ.

2.1 Effective group actions

The cryptographic framework used throughout this work are *effective group actions* (EGA). EGAs were initially introduced in [34, 86] and further examined in [2] as means to abstract isogeny computations in Schreier graphs. We start with their definition.

Definition 1 (Effective group action (EGA) [2, Definition 3.4]). *A group $(\mathcal{G}, \cdot)$ acting on a set $\mathcal{E}$ via $\star : \mathcal{G} \times \mathcal{E} \to \mathcal{E}$ is called an* effective group action*, if both $\mathcal{G}$ and $\mathcal{E}$ are finite and the following hold.*

1. *There exist efficient PPT algorithms for membership testing, equality testing, sampling, inversion and multiplication of elements in $\mathcal{G}$.*
2. *There exist efficient algorithms for membership testing and canonical representation for elements in $\mathcal{E}$.*
3. *There exists a distinguished element $E_0 \in \mathcal{E}$ called the* origin *with known representation.*
4. *There exists an efficient algorithm that on input $\mathfrak{a} \in \mathcal{G}$ and $E \in \mathcal{E}$ outputs the group action $\mathfrak{a} \star E$.*

We denote such group actions as $(\mathcal{G}, \mathcal{E}, \star)$. We will only work with group actions that are commutative and *regular*.

Definition 2 (Regular group action). *A group action $(\mathcal{G}, \mathcal{E}, \star)$ is* regular*, if it satisfies the following two properties.*

- *Freeness: if $\mathfrak{a} \star E = E$ for some $E \in \mathcal{E}$, then $\mathfrak{a}$ is the neutral element of $\mathcal{G}$.*
- *Transitivity: for any two elements $E, E' \in \mathcal{E}$ there exists $\mathfrak{a} \in \mathcal{G}$, such that $E' = \mathfrak{a} \star E$.*

For cryptographic EGAs, we assume the following problems (the EGA counterparts of discrete logarithm and computational Diffie-Hellman problems, respectively) to be hard.

Problem 1 (Vectorization [34, Problem 6]). Given $E, E' \in \mathcal{E}$, find $\mathfrak{a} \in \mathcal{G}$, such that $E' = \mathfrak{a} \star E$.

Problem 2 (Parallelization [34, Problem 7]). Given $E_0, E_1, F_0 \in \mathcal{E}$ with $E_1 = \mathfrak{a} \star E_0$, compute the unique $F_1 = \mathfrak{a} \star F_0$.

Effective group actions can be instantiated from isogenies between supersingular elliptic curves, which has been the focus of many research papers in the past few years [6, 22, 31, 38, 41, 80]. Throughout the following sections, we will make black-box usage of effective group actions, and postpone discussions about concrete instantiations to Section 5.

In this work, the acting group will always be cyclic with known generator $\mathfrak{g}$. In this case, we can express other group elements as powers of $\mathfrak{g}$. This allows us to work in the group of exponents of $\mathfrak{g}$, $\mathbb{Z}/q\mathbb{Z}$ where $q = |\mathcal{G}|$, rather than in $\mathcal{G}$ itself. This idea goes back to CSI-FiSh [22], who introduced the notation

$$[\cdot] : \mathbb{Z}/q\mathbb{Z} \times \mathcal{E} \to \mathcal{E}\,, \quad \text{where} \quad [a]E = \mathfrak{g}^a \star E\,. \tag{1}$$

Note that consecutive group actions result in additions in $\mathbb{Z}/q\mathbb{Z}$, since

$$[a]([b]E) = \mathfrak{g}^a \star (\mathfrak{g}^b \star E) = (\mathfrak{g}^a\mathfrak{g}^b) \star E = \mathfrak{g}^{a+b} \star E = [a+b]E\,. \tag{2}$$

Throughout this paper, we will assume by default that the group $\mathcal{G}$ has prime order q. This simplifies our exposition, although this is a non-trivial assumption as ideal-class groups of quadratic imaginary orders do not generally have this property. However, we show in Section 5 how this can be achieved constructively. Further, we will outline the main differences to the case where q is composite in more detail in the same section.

2.2 Verifiable secret sharing

A secret sharing of a value s in a (t, n)-threshold setting is the splitting of s into n pieces called shares, such that the knowledge of any t or fewer shares contains no information about s, and the knowledge of any $t+1$ or more shares allows the efficient and unique reconstruction of s. The sharing of such a secret can either be performed by a trusted dealer or in a distributed way among the parties. In the latter case, it is generally desirable that the parties can verify that they have indeed received correct shares.

Definition 3 (Verifiable secret sharing). *A* verifiable secret sharing *(VSS) scheme is a tuple of algorithms* (Share, Reconstruct)*, invoked by a set* $T \subseteq \{1, \ldots, n\}$ *of parties, where*

- Share *samples a secret and distributes a secret share to each party in* T*,*
- Reconstruct *takes as input the shares of parties in* T *and outputs the secret.*

A VSS is secure in a (t, n)*-threshold setting if it satisfies the following conditions.*

- ***Secrecy:*** *There exists a PPT simulator* Sim *interacting with an adversary* $\mathcal{A}$ *corrupting up to* t *parties, which can simulate the view of* $\mathcal{A}$ *during the execution of* Share*, such that it is indistinguishable from the real execution.*
- ***Correctness:*** *At the end of* Share*, all honest parties hold a valid* (t, n)*-sharing of a given secret* s*.*
- ***Robustness:*** *The* Reconstruct *routine always ends with correct reconstruction of the secret, even in the presence of up to* t *corruptions.*

2.2.1 Shamir secret sharing and Lagrange interpolation. One possible way of achieving a (t, n)-sharing was designed by Shamir [87], who proposed to encode the secret into the constant term of a degree t monovariate polynomial $f(X)$. The secret is $s = f(0)$, while the individual shares are evaluations of $f(X)$ at different points, usually $f(1), \ldots, f(n)$. These shares can then be distributed among n parties, who have to join forces to reconstruct $f(0)$. In fact, less than $t + 1$ shares yield no information about $f(0)$, but reconstruction of the secret can be performed efficiently using Lagrange interpolation from any $t+1$ or more shares. Let $T \subseteq \{1, \ldots, n\}$ be a subset of size $|T| > t$, then we can recover

$$f(X) = \sum_{i \in T} L_i^T(X) f(i)\,, \quad \text{where} \quad L_i^T(X) = \prod_{j \in T \setminus \{i\}} \frac{X - j}{i - j} \tag{3}$$

are the Lagrange interpolation polynomials. In particular, this allows us to recover $f(0)$ by using the Lagrange polynomials evaluated at 0, $L_i^T := L_i^T(0)$. To indicate that a value is shared in a Shamir secret sharing, we write $\langle s \rangle := \{f(1), \ldots, f(n)\}$, with the individual shares $f(i)$ held by different parties. We use indices to define subsets of this sharing, in particular $\langle s \rangle_T = \{f(i) \mid i \in T\}$ and $\langle s \rangle_i = f(i)$.

To allow verifiability of shares, especially when shares are generated in a distributed fashion, Shamir secret sharing usually employs commitment schemes. Schemes with classical security usually rely on verifying their shares using homomorphic commitments, such as e.g. the schemes by Feldman [49] or Pedersen [81], where the share dealer commits to the polynomial coefficients, and verifiers can use the homomorphic property to verify that their share is indeed a correct evaluation of the polynomial at their position i. Post-quantum security can for example be generically achieved from commitment schemes and zero-knowledge proofs as presented in [10, 12]. We revisit the proof system of a correct Shamir secret sharing, Π [12], in Appendix A.1.

2.2.2 Bivariate secret sharing. Shamir secret sharing can be extended from monovariate to bivariate polynomials, as introduced by Ben-Or, Goldwasser and Widgerson [18]. Here, secrets are encoded as the constant term of a bivariate polynomial $F(X, Y)$ with degree t in each of its variables, while each party holds an evaluation of this polynomial in one of its variable, e.g. party P_i holds $(F(X, i), F(i, Y))$, both degree t polynomials. Reconstruction can then again be achieved by parties doing Lagrange interpolation using the shares $F(i, 0)$ and is only possible if at least $t + 1$ parties cooperate.

The original idea of [18] to verify these shares is for parties to engage in pairwise sub-protocols, where party P_i and P_j check whether they have matching evaluations of $F(i, j)$ and $F(j, i)$. This allows maliciously secure verification of shares and reconstruction of the secret in the presence of up to t malicious parties, without the need for commitments, thus achieving perfect secrecy. However, in the malicious setting, the threshold of corruptions is reduced to $t < n/3$ due to Byzantine faults. A key advantage of bivariate secret sharing over Shamir secret

sharing is that every party P_i not only possesses shares $F(i,0)$ of the secret $F(0,0)$, but via the knowledge of $F(j,i)$, also shares of each other party P_j's share $F(j,0)$ of the secret. Such a share can be reconstructed by interpolating the second variable to 0. This property is particularly useful when malicious parties misbehave in protocols and their share needs to be reconstructed by other parties, without revealing the shared secret. Looking ahead, this property will allow us to achieve robustness in the malicious setting.

A way to reduce the allowed corruptions to $t < n/2$ was achieved by Backes, Kate and Patra [11], who employ (generic) commitment schemes to prove correctness of parties' shares in a publicly verifiable way. To this end, provers publish a matrix of commitments

$$\begin{pmatrix} \mathsf{C}(F(1,1), r_{11}) & \dots & \mathsf{C}(F(1,n), r_{1n}) \\ \mathsf{C}(F(2,1), r_{21}) & \dots & \mathsf{C}(F(2,n), r_{2n}) \\ \vdots & \vdots & \vdots \\ \mathsf{C}(F(n,1), r_{n1}) & \dots & \mathsf{C}(F(n,n), r_{nn}) \end{pmatrix},$$

and send the commitment openers $\{r_{ij}, r_{ji}\}_{j \in \{1,\dots,n\}}$ to each party P_i, which can then verify row i and column i. The matching of shares between different (honest) parties is then guaranteed by agreeing to this publicly available data, without the possibility of Byzantine attacks as in [18]. Another modification that the authors perform is to work with bivariate polynomials that are *symmetric*, i.e. where it holds that $F(X,Y) = F(Y,X)$, an idea that goes back to [36].

Throughout this work, we will only consider symmetric bivariate polynomials. We write such a symmetric bivariate sharing of a secret $s = F(0,0)$ as $[\![s]\!]$, where $[\![s]\!]_T$ indicates the subset of shares held by parties in T and by $[\![s]\!]_i = F(i,Y)$ the share held by party P_i.

2.3 Distributed group action computation

The distributed computation of group actions has first been explored in [42] in the semi-honest setting. The idea behind computing the action of a secret s using only its shares is to use the additive property outlined in equation (2), which allows us to do Lagrange interpolation in the exponent. In a round-robin, the different parties holding shares $\langle s \rangle_i = f(i)$ of the secret $s = f(0)$ can perform Lagrange interpolation by subsequently applying their share weighted with the correct Lagrange coefficient. Assume for simplicity that $T = \{1, \dots, m\}$, then this approach yields

$$[L_m^T f(m)] \cdots [L_1^T f(1)] E_0 \overset{(2)}{=} \Big[\sum_{i=1}^{m} L_i^T f(i) \Big] E_0 = [f(0)] E_0 \,.$$

The security properties expected from this distributed group action computations are the same as of a distributed key generation, i.e. that the output is correct, and that furthermore, the adversary does not learn anything about the secret used or about the output before the computation. Additionally, we

again impose that the computation is robust. We use the definitions from CSI-RAShi [20, Definitions 3 and 4] to this end.

Definition 4 (Distributed group action). *A* distributed group action *(DGA) algorithm* GroupAction *takes as input a curve $E \in \mathcal{E}$ and the shares of parties in a set $T \subseteq \{1, \ldots, n\}$ with $|T| > t$ defining a secret s, and outputs the curve $[s]E$. A DGA is secure in a (t, n)-threshold setting if it satisfies the following conditions.*

- ***Secrecy:*** *There exists a PPT simulator* Sim*, which on input $E' \leftarrow \mathcal{E}$ simulates the honest parties interacting with the adversary corrupting up to t parties, such that the protocol ends with E' as an output.*
- ***Robust correctness:*** *The* GroupAction *routine always ends with the correct computation of $[s]E$, even in the presence of up to t corruptions.*

In the malicious setting, these properties are usually achieved by introducing zero-knowledge proofs of correct execution at each step of this round-robin [9,20]. Since we will be working with a bivariate sharing in the malicious setting, we can make use of the structured piecewise verifiable proofs introduced in [9], which we revisit in Appendix A.2.

3 New (verifiable) oblivious pseudorandom function

We introduce our new OPRF construction in this section. We start by defining the ideal functionality on which our construction relies, then describe the high level idea of our construction and how to make the output verifiable. We finally present our construction in Section 3.6.

3.1 Ideal functionality

We define the ideal functionality $\mathcal{F}_{\mathsf{VOPRF}}$ of a verifiable OPRF in Figure 1. We note that there are many ways of defining such functionalities and point the interested reader to a systematization outlined in [28]. Our functionality draws inspiration from [24, Def. 45], but with some modifications that we outline below.

The functionality outputs random bitstrings of length ℓ depending on the input, such that the same input consistently gives the same output. This is realized similar to a random oracle, where inputs are lazily sampled and a list of input-output pairs is kept, and captures the fact that the OPRF output is a random value of ℓ bits.

On prompt Init, the functionality initiates a new server by giving it a function descriptor Π. This Π can be interpreted as the server's identity, allowing us to realize a keyed pseudorandom function. The functionality also keeps track of server identities in a parameter list `params`. For simplicity, we can assume that each server registers only a single function, treating multiple descriptors as belonging to different servers. Furthermore, we model multiple concurrent

queries by a single client simply as single evaluation queries by different concurrent users. Similarly to [24], these modifications allow us to avoid session and sub-session identifiers in the syntax of the functionality.

A client can request the evaluation of an input m by a server S using the query (EVAL, S, m). The functionality stores the request (C, S, m) and notifies the adversary via (EVAL, C, S), by telling it that C has queried EVAL from S, but without leaking information about the client's input. If the server agrees to the evaluation, it sends the message SERVERCOMPLETE to $\mathcal{F}_{\mathsf{VOPRF}}$.

If the server accepts, the adversary can decide whether or not to suppress the server's output. If it allows the request, it sends $(\text{CLIENTCOMPLETE}, C, \Pi)$ to $\mathcal{F}_{\mathsf{VOPRF}}$. Upon receiving this message, $\mathcal{F}_{\mathsf{VOPRF}}$ verifies whether an entry (C, S, m) exists. If it does, $\mathcal{F}_{\mathsf{VOPRF}}$ outputs the correct evaluation to C and removes this entry. The fact that this evaluation is correct represents the verifiability property of the OPRF, i.e. if the server's response doesn't verify correctly, then there is no correct output to be constructed.

If no such entry exists, the functionality aborts. The idea behind the functionality keeping track of these tuples is so that a client can only learn the evaluation of a value m that it actually queried with an EVAL message. This captures the one-more security of an OPRF, i.e. the idea that a client cannot produce an OPRF output itself for something that has not been queried before. Since in our protocol, the client's input will be extractable, we don't need the ticket mechanism used in other isogeny OPRF constructions [15, 24]. The ticket mechanism was introduced in [59] to circumvent the need of online extractability, which our protocol provides. In these constructions, for each EVAL request, the functionality increments a counter and for each CLIENTCOMPLETE decreases the counter again. If the counter is zero and an output is requested, the functionality aborts, preventing the client from getting more outputs than requested inputs.

3.2 The pseudorandom function

The pseudorandom function underlying our OPRF is based on the idea that the server holds a secret key sk and the client an input m, and the output of the functionality (before hashing it) is the elliptic curve $[\mathsf{sk} \cdot m]E_0$, where E_0 is a fixed starting curve. We note that via equation (2), sequential group actions are additive, thus multiplying two values would technically only be possible if a party would know both elements sk and m. We can circumvent this issue by working with correlated randomness tuples, similar to Beaver's trick [17]. As a result, our OPRF will rely on a preprocessing phase that we outline in Section 3.3 below.

The one-more security of our PRF relies on the assumption that the following problem is hard.

Problem 3 (One-more unpredictability). Let $\mathcal{O}_s$ be an oracle, which on input m, produces $[sm]E_0$. Given (classical) oracle access to $\mathcal{O}_s$, produce a pair $(m^*, [sm^*]E_0)$, where m^* has not been queried to $\mathcal{O}_s$ before.

As we will discuss in Section 5, this problem will impose important restrictions on the parameters we can use in a secure instantiation.

Definition 5 (Ideal functionality $\mathcal{F}_{\mathsf{VOPRF}}$).
The functionality maintains a random function F that maps key-input pairs $(\Pi, m) \in \{0,1\}^*$ to ℓ-bit strings. Whenever a value $F(\Pi, m)$ is referenced, if it is undefined, then $\mathcal{F}_{\mathsf{VOPRF}}$ assigns $F(\Pi, m) \leftarrow \{0,1\}^\ell$.

Initialization

- On message Init from S, send (Init, S) to $\mathcal{A}$.
- On message (Params, S, Π) by $\mathcal{A}$, if $\mathsf{param}(S)$ is undefined, set $\mathsf{params}(S) = \Pi$. Send (Params, Π) to S.

Evaluation

- On message (Eval, S, m) by client C, record (C, S, m) and send (Eval, C, S) to $\mathcal{A}$.
- On message ServerComplete from S, send $(\text{ServerComplete}, S)$ to $\mathcal{A}$.
- On message $(\text{ClientComplete}, C, \Pi)$ from $\mathcal{A}$, for $\Pi = \mathsf{params}(S)$:
  - If no record (C, S, m) exists, abort.
  - Retrieve and delete (C, S, m) from the record.
  - Send $(\text{Eval}, \Pi, F(\Pi, m))$ to C.

Fig. 1: Ideal functionality $\mathcal{F}_{\mathsf{VOPRF}}$

We note that our PRF can be seen as a modification of the PRF originally proposed in [44], in which the outputs are polynomial evaluations on the input message m, i.e. $[f(m)]E_0$. Their one-more security relies a similar problem to Problem 3, but where the oracle and the target output are polynomial evaluations of m, instead of multiplications. In particular the authors look at different polynomial degrees, noting that linear polynomials are insecure when the attacker is allowed to make quantum superposition queries to the oracle. In that case, the reduction from [53] would allow an attacker to extract the server's secret polynomial and break the scheme.

Although such an attack would be applicable in the setting of Problem 3, we note that since these PRFs are keyed functions evaluated by a server, we can simply mitigate it by disallowing superposition queries. Actually, the converse case would even be highly non-trivial in a verifiable setting: allowing quantum superposition queries to a *verifiable* function, would imply that the server somehow provides correctness proofs in these superposition states. This issue is only exacerbated when the server side is distributed and the servers also have to manipulate quantum states and convince each other in zero-knowledge that they have executed their advertised functionality correctly. As a consequence, we can safely restrict ourselves to the case of classical access to the PRF functionality and where servers provide classical proofs to each other and to the client.

Going from polynomial evaluation to a scalar multiplication PRF has both computational advantages and lower communication, which will both prove to be particularly significant in the multi-party case. Furthermore, the simplicity of the statement allows us to construct a much more efficient, but also flexible proof system to make our OPRF verifiable, which we discuss in detail Section 3.4. The

main motivation behind this new proof system is that it will allow us to remove the need for the preprocessing phase to rely on a trusted setup, as opposed to the design in [44]. We discuss the preprocessing phase first in Section 3.3.

3.3 Preprocessing phase

Our OPRF starts with a preprocessing phase in which a functionality generates triples of the type (u, v, w), where $w = u \cdot v$. The client and the server then each receive u and v, respectively, as well as an additive share of w, such that in the end, the client holds (u, w^C) and the server (v, w^S), such that $w^C + w^S = w = u \cdot v$. Holding these pairs, the idea is that both parties can exchange maskings of the message $\alpha = m - u$ and secret key $\beta = \mathsf{sk} - v$ and then jointly (sequentially) compute

$$[\mathsf{sk} \cdot m]E_0 = [(\beta + v)(\alpha + u)]E_0 = [w^C + \beta m][w^S + \alpha v]E_0\,, \tag{4}$$

where $w^C + \beta m$ is known to the client and $w^S + \alpha v$ to the server. We will end up needing multiple such triples in our protocol, so we endow each of these triples with unique identifiers id. We assume that there is an external mechanism, through which the server and client decide on the identifiers of the tuples that they are working with. We capture this preprocessing phase by defining a tuple generation functionality in Definition 6.

Definition 6 (Tuple generation functionality $\mathcal{F}_{\mathsf{TG}}$).

1. On the first activation, learn the corrupted parties from the adversary.
2. On input $(\textsc{Tuple}, \mathsf{id}, C, S)$ by a party $P \in \{C, S\}$, if there is no recorded entry for id, sample $u, v, w^S, w^C \leftarrow \mathbb{Z}_q$ (using the knowledge of the public parameters pp) with $w^C + w^S = uv$ and record $(C, S, \mathsf{id}, u, v, w^S, w^C)$. Otherwise, retrieve $(C, S, \mathsf{id}, u, v, w^S, w^C)$. Then,
   - If P is the client, send (u, w^C) to C and send $(\textsc{Output}, C)$ to $\mathcal{A}$.
   - If P is the server, send (v, w^S) to S and send $(\textsc{Output}, S)$ to $\mathcal{A}$.

Such a functionality can be realized either by a trusted third party, which simply distributes these elements between the client and the server, or can be built from triple generation functionalities. For example, our tuples are a special case of Beaver triples, which can be generated solely from oblivious transfer techniques, as for example outlined in MASCOT [67, Section 5]. Oblivious transfer instantiations from isogeny group actions assumptions have for example been explored in [70,78]. Furthermore, after a one-time setup, many more such tuples can be generated using OT extension techniques [58,66] purely from symmetric primitives, leading to very low amortized costs. We note that in the *verifiable* OPRF construction from [44], the functionality further generates commitments

of the type $([v]E_0, [w^S]E_0)$, for which, to our knowledge, no efficient (and amortizable) setup phase readily exists in the literature and is also not discussed in their paper.[2]

3.4 Proving correct output

We now discuss how outputs of the OPRF are made verifiable. Looking at equation (4) we note that it would suffice for the server to convince the client that it computed the action $E_S = [w^S + \alpha v]E_0$ correctly, i.e. using the received elements w^S and v. Indeed, a conceptually similar approach is taken in [44], which leverages the aforementioned commitments $([v]E_0, [w^S]E_0)$.

The proof system we introduce takes a very different approach. Rather than proving that the server-side group action has been computed using the correct group elements, our proof system instead verifies the OPRF output directly, after it has been computed on the client side. Although semantically this implies the same result, the server side statement and thus the proof statement become fundamentally different. In fact, for a given public key $\mathsf{pk} = [\mathsf{sk}]E_0$, the server now convinces the client that the output it computed is $E_m = [\mathsf{sk} \cdot m]E_0$. This is exactly the statement that

$$((E_0, \mathsf{pk}), (E_0, E_m, m)) \in \mathcal{L}_{\mathsf{Scal}}\,, \tag{5}$$

where

$$\mathcal{L}_{\mathsf{Scal}} = \left\{ (E_0, E_1), (F_0, F_1', a) \mid \exists s \in \mathcal{G} : E_1 = [s]E_0 \wedge F_1' = [a \cdot s]F_0 \right\}$$

is the language of scalar multiplications in the exponent introduced in [13]. For public m, [13] provides a zero-knowledge proof for this language. The key difference in our case, however, is that the server doesn't actually know the value m for which it has to produce the statement. As a consequence, our proof system becomes a two-party protocol, where both inputs m and sk are held by the client and the server, respectively, without revealing them to each other. Each party only receives a masked version of these values as part of the protocol.

We outline the way our proof system realizes this below and visualize the steps in Figure 2. While we try to stay general with respect to $\mathcal{L}_{\mathsf{scal}}$, note that in the OPRF, we will have $F_0 = E_0$ and $F_1' = [\mathsf{sk} \cdot m]F_0$, which is the output curve of the OPRF. As shown in (4), the verifier can compute the latter from $F_1 = [w^S + \alpha v]F_0$ received by the prover.

Our proof system has binary challenge space and will need to be repeated λ times to achieve soundness. At each repetition step $j = 1, \ldots, \lambda$, the prover samples randomizers $b_j \leftarrow \mathbb{Z}_q$ and queries $\mathcal{F}_{\mathsf{TG}}$ to get its half (v_j, w_j^S) of a triple. Then it commits to both $\hat{E}_j = [b_j]E_0$ and a curve $\hat{F}_j$, which the verifier can

[2] Note that in our setting, revealing $[w^S]E_0$ would actually allow the client to compute $[\beta u + w^C][w^S]E_0 = [\mathsf{sk} \cdot u]E_0$, which is a valid OPRF output for input u, known by the client. This would void the one-more security of our construction. In [44], such attacks are mitigated through the constant term of the polynomial.

later transform into $\hat{F}'_j = [b_j m]F_0$ using the other half (u_j, w_j^C) of the triple. The verifier can now check (for challenge $c_j = 1$) whether a response $r_j = b_j - \mathsf{sk}$ satisfies both the conditions that $[r_j]E_1 = [b_j]E_0$ and $[r_j m]F'_1 = [b_j m]F_0$. For $c_j = 0$, the client verifies that the commitments have been computed correctly using the response $r_j = (v_j, w_j^S)$.

We visualize this in the diagram below. Note that the prover provides F_1 and $\hat{F}_j$, from which the verifier can derive F'_1 and $\hat{F}'_j$. Note that the curve $\hat{F}'_j$ itself does not have to be computed, but can remain implicit, reducing the cost on the verifier side. In fact, for $c_j = 1$, the verifier combines the two isogenies $F'_1 \to \hat{F}'_j$ and $\hat{F_j}' \to \hat{F}_j$ on the right side to a single one from $F'_1 \to \hat{F}_j$.

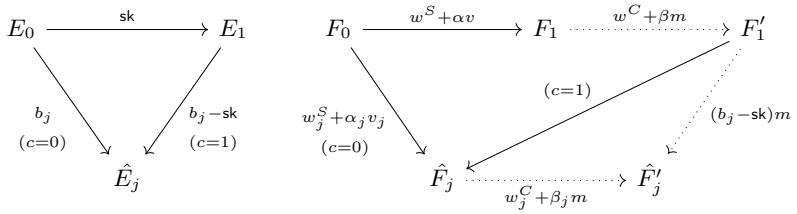

Fig. 2: Idea underlying the OPRF proof system

We integrate our proof system into our OPRF construction in Section 3.6 and prove its security as part of the whole protocol in Section 3.7.

3.5 Subalgorithms

Before we present our full construction in the next section, we introduce some simple algorithms in Figure 3 on which all of our subsequent protocols will be based. The idea behind this presentation is that later we also want to be able to exchange these algorithms with multi-party instantiations. By showing that our multi-party instantiations realize the same ideal functionalities, we can then simply interchange them, while upholding the general structure and thus the desired properties implied by the original protocol. We will discuss this in more detail in the tresholdization Section 4.

The algorithm Samp samples and outputs elements in the acting group $\mathbb{Z}_q$, while Rec computes (recovers) linear combinations of such elements. We introduce the bold notation to designate vectors (of variable length) in $\mathbb{Z}_q$, e.g. $\mathbf{a} = (a_1, \ldots, a_k)$ and we restrict inputs of Rec to vectors of the same length. Finally, GA takes as input a pair of such vectors $\mathbf{a}$ and $\mathbf{s}$ and an elliptic curve $E \in \mathcal{E}$ and computes the group action $[s]E$, where $s = \mathbf{a} \cdot \mathbf{s}$.

Throughout the rest of this work, we will let the first input $\mathbf{a}$ designate publicly available scalars (at least to the parties executing the algorithm) and the second input $\mathbf{s}$ designate secret (or later shared) elements. Note that the

$\mathsf{Samp}(\mathsf{pp})$**:** Sample and return $s \leftarrow \mathbb{Z}_q$.
$\mathsf{Rec}(\mathbf{a}, \mathbf{s})$**:** Compute and return $\mathbf{a} \cdot \mathbf{s} = \sum_{i=1}^{k} a_i s_i$.
$\mathsf{GA}(\mathbf{a}, \mathbf{s}, E)$**:** Compute and return $[\mathbf{a} \cdot \mathbf{s}]E$.

Fig. 3: Subalgorithms for sampling, linear combination and group action computations over linear combinations of elements in the acting group.

secret inputs in $\mathbf{s}$ can either come from the Samp algorithm, or are received from the tuple generation functionality $\mathcal{F}_{\mathsf{TG}}$.

3.6 (Verifiable) Oblivious Pseudorandom Function

We present our OPRF construction in Figure 4. Our protocol utilizes a random oracle $\mathsf{H}_O : \{0,1\}^* \to \{0,1\}^\ell$ to compute the final output of the OPRF. Here ℓ is a function of the security parameter λ. The built-in proof system from Section 3.4 uses another random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ to generate challenges.

We highlight which steps in our construction are introduced in order to make the output *verifiable* (grayed elements). We note that these steps can simply be omitted if a standard (non-verifiable) OPRF is desired. The verifiable version of our construction employs some extra steps to convince the verifier that indeed the output E_m of the OPRF satisfies the relation $((E_0, \mathsf{pk}), (E_0, E_m, m)) \in \mathcal{L}_{\mathsf{Scal}}$.

In Section 3.7, we prove the following theorem.

Theorem 1. *The protocol from Figure 4 realizes the ideal functionality $\mathcal{F}_{\mathsf{VOPRF}}$ from Definition 5 in the $\mathcal{F}_{\mathsf{TG}}$-hybrid model, assuming that Problem 3 is hard.*

3.7 Security proof of our VOPRF

We show that the protocol in Figure 4 realizes the ideal functionality $\mathcal{F}_{\mathsf{VOPRF}}$ from Definition 6. To this end, we introduce a simulator Sim interacting with the protocol adversary $\mathcal{A}$ and simulating the real protocol in the $\mathcal{F}_{\mathsf{TG}}$-hybrid model, such that the view of the environment is computationally indistinguishable from the real world.

The simulator runs the honest parties and further provides the interface between the adversary and the random oracles, as well as the tuple generation functionality $\mathcal{F}_{\mathsf{TG}}$. The simulator runs the random oracle $\mathcal{H}$ truthfully, and can reprogram its input-output pairs. Similarly, Sim correctly forwards tuple queries to $\mathcal{F}_{\mathsf{TG}}$, and sends the outputs back to $\mathcal{A}$, allowing it to learn all the tuples generated and shared with the client and server. Knowledge of these tuples allows Sim to later extract both the client's inputs as well as the server's secret key in a straightline fashion, as they are masked by simple linear relations using exactly these tuples. The extraction of the server's secret key allows to guarantee soundness of the underlying proof system, while the extraction of the client's input is used by the simulator to guarantee that the client cannot learn more outputs than it has submitted queries to the OPRF.

Preprocessing

1. Client and Server decide on a unique identifier id and query $\mathcal{F}_{\mathsf{TG}}(\mathsf{pp}, \mathsf{id})$. The client stores (u, w^C) and the server stores (v, w^S).
2. Client and Server decide on λ further unique identifiers $\underline{\mathsf{id}}$ and query $\mathcal{F}_{\mathsf{TG}}(\mathsf{pp}, \underline{\mathsf{id}})$. The client stores $(\underline{u}, \underline{w^C})$ and the server stores $(\underline{v}, \underline{w^S})$.

Initialization

On message INIT from the environment, S proceeds as follows.

1. Sample and store $\mathsf{sk} \leftarrow \mathsf{Samp}(\mathsf{pp})$.
2. Compute and store $\mathsf{pk} = \mathsf{GA}(1, \mathsf{sk}, E_0)$.
3. Output (INIT, pk).

Evaluation

On message (EVAL, S, m) by the environment, C proceeds as follows.

1. Recover u and compute $\alpha = \mathsf{Rec}((1,-1),(m,u))$. $\triangleright\, \alpha = m - u$
2. For $j = 1, \ldots, \lambda$, recover u_j and compute $\alpha_j = \mathsf{Rec}((1,-1),(m,u_j))$.
3. Store and send $\alpha, \underline{\alpha}$ to the server. $\triangleright\, \alpha_j = m - u_j$

On input $(\alpha, \underline{\alpha})$ from C, the server S recovers $(\mathsf{sk}, \mathsf{pk}, v, w^S, \underline{v}, \underline{w^s})$ and proceeds as follows.

1. Compute $\beta = \mathsf{Rec}((1,-1),(\mathsf{sk},v))$. $\triangleright\, \beta = \mathsf{sk} - v$
2. Compute $E_S = \mathsf{GA}((1,\alpha),(w^S,v),E_0)$. $\triangleright\, E_S = w^S + \alpha v$
3. For $j = 1, \ldots, \lambda$:
   a) Sample $b_j \leftarrow \mathsf{Samp}(\mathsf{pp})$.
   b) Compute $\beta_j = \mathsf{Rec}((1,-1),(b_j,v_j))$. $\triangleright\, \beta_j = b_j - v_j$
   c) Compute $\hat{E}_j = \mathsf{GA}(1, b_j, E_0)$. $\triangleright\, \hat{E}_j = [b_j]E_0$
   d) Compute $\hat{F}_j = \mathsf{GA}((1,\alpha_j),(w_j^S, v_j), E_0)$. $\triangleright\, \hat{F}_j = [w_j^S + \alpha_j v_j]E_0$
4. Query $\underline{c} \leftarrow \mathcal{H}(E_0, \mathsf{pk}, E_S; \underline{\hat{E}}, \underline{\hat{F}})$.
5. For $j = 1, \ldots, \lambda$,
   – if $c_j = 0$, set $r_j = (\mathsf{Rec}(1, v_j), \mathsf{Rec}(1, w_j^S))$, $\triangleright\, r_j = (v_j, w_j^S)$
   – if $c_j = 1$, $r_j = \mathsf{Rec}((1,-1),(b_j,\mathsf{sk}))$. $\triangleright\, r_j = b_j - \mathsf{sk}$
6. Send $(E_S, \beta, \underline{\beta}, \underline{c}, \underline{r})$ to the client.

On input $(E_S, \beta, \underline{\beta}, \underline{c}, \underline{r})$ from S, the client C proceeds as follows.

1. Compute $E_m = \mathsf{GA}((1,\beta),(w^C, m), E_S)$.
2. Recover $\underline{\alpha}$. Then, for $j = 1, \ldots, \lambda$:
   a) If $c_j = 0$, let $r_j = (r_{j,1}, r_{j,2})$. If $u \cdot r_{j,1} \neq w_j^C + r_{j,2}$, return $\perp$.
      Otherwise compute $\hat{E}_j = \mathsf{GA}(\beta_j + r_{j,1}, 1, E_0)$ $\triangleright\, \hat{E}_j = [\beta_j + r_{j,1}]E_0$
      and $\hat{F}_j = \mathsf{GA}((1,\alpha_j),(r_{j,2}, r_{j,1}), E_0)$. $\triangleright\, \hat{F}_j = [r_{j,2} + \alpha_j r_{j,1}]E_0$
   b) If $c_j = 1$, compute $\hat{E}_j = \mathsf{GA}(1, r_j, \mathsf{pk})$ and $\triangleright\, \hat{E}_j = [r_j]\mathsf{pk}$
      $\hat{F}_j = \mathsf{GA}((r_j, -\beta_j, -1),(m, m, w_j^C), E_m)$. $\triangleright\, \hat{F}_j = [r_j m - \beta_j m - w_j^C]E_m$
3. If $\underline{c} \neq \mathcal{H}(E_0, \mathsf{pk}, E_S; \underline{\hat{E}}, \underline{\hat{F}})$, return $\perp$.
4. Compute $y \leftarrow \mathsf{H_O}(m, \mathsf{pk}, E_m)$ and output (EVAL, pk, y).

Fig. 4: Verifiable OPRF protocol.

We give a description of the simulator and design a sequence of hybrids to show that the simulated world is indistinguishable from the real execution of the protocol by any polynomial-time environment.

The simulator Sim proceeds as follows.

1. Whenever a party $P \in \{C, S\}$ queries $\mathcal{F}_{\mathsf{TG}}$, Sim simulates $\mathcal{F}_{\mathsf{TG}}$ honestly and records tuples of the type $(\mathsf{id}, u, v, w^S, w^C)$.
2. On queries x to $\mathcal{H}$, if there is no record for x, sample $\underline{c} \leftarrow \{0,1\}^\lambda$ and record $(\mathcal{H}, x, \underline{c})$. Output $\underline{c}$.
3. On message $(\textsc{Init}, S)$ from $\mathcal{F}_{\mathsf{VOPRF}}$, execute $\mathsf{sk} \leftarrow \mathbb{Z}_q$ and compute $\mathsf{pk} = [\mathsf{sk}]E_0$. Then, return $(\textsc{Params}, S, \Pi = \mathsf{pk})$ to $\mathcal{F}_{\mathsf{VOPRF}}$ and store $(S, \mathsf{sk}, \mathsf{pk})$.
4. On message $(\textsc{Eval}, C, S)$ from $\mathcal{F}_{\mathsf{VOPRF}}$, sample $\alpha, \underline{\alpha} \leftarrow \mathbb{Z}_q$. Send $(\alpha, \underline{\alpha})$ to $\mathcal{A}$ as a message from C to S. Recover the tuples $(\mathsf{id}, u, v, w^S, w^C)$ and $(\underline{\mathsf{id}}, \underline{u}, \underline{v}, \underline{w^S}, \underline{w^C})$ and compute $m = \alpha + u$ and $\underline{m} = \underline{\alpha} + \underline{u}$. and store $(C, m, \underline{m})$.
5. On receiving $(\alpha', \underline{\alpha}')$ from $\mathcal{A}$ as a message from C to the honest server S, the simulator proceeds as follows.
   (a) Recover the tuples $(\mathsf{id}, u, v, w^S, w^C)$, $(\underline{\mathsf{id}}, \underline{u}, \underline{v}, \underline{w^S}, \underline{w^C})$ and $(S, \mathsf{sk}, \mathsf{pk})$.
   (b) Extract the messages $m' = \alpha' + u$ and $\underline{m}' = \underline{\alpha}' + \underline{u}$ and store $(C, m', \underline{m}')$.
   (c) Sample $\beta \leftarrow \mathbb{Z}_q$ and compute $E_m = [m' \cdot \mathsf{sk}]E_0$ and then $E_S = [-w^C - \beta m']E_m$.
   (d) Sample $\underline{c} \leftarrow \{0,1\}^\lambda$ and $\underline{\beta} \leftarrow \mathbb{Z}_q^\lambda$, then for each $j = 1, \ldots, \lambda$:
      i. if $c_j = 0$, sample $r_j \leftarrow \mathbb{Z}_q^2$ with $u \cdot r_{j,1} = w_j^C + r_{j,2}$ and compute $\hat{E}_j = [\beta_j + r_{j,1}]E_0$ and $\hat{F}_j = [r_{j,2} + \alpha_j r_{j,1}]E_0$,
      ii. if $c_j = 1$, sample $r_j \leftarrow \mathbb{Z}_q$ and compute $\hat{E}_j = [r_j]\mathsf{pk}$ and $\hat{F}_j = [r_j m'_j - (w_j^C + \beta_j m'_j)]E_m$,

      then record $(\mathcal{H}, x, \underline{c})$ for $x = (E_0, \mathsf{pk}, E_S, \beta; \underline{\hat{E}}, \underline{\hat{F}})$.
   (e) Send $(E_S, \beta, \underline{\beta}, \underline{c}, \underline{r})$ to $\mathcal{A}$ as a response of S intended for C and send $(\textsc{ServerComplete}, S)$ to $\mathcal{F}_{\mathsf{VOPRF}}$.
6. On receiving $(E'_S, \beta', \underline{\beta}', \underline{c}', \underline{r}')$ from $\mathcal{A}$ as a message from S to C, the simulator proceeds as follows.
   (a) Recover the tuples $(\mathsf{id}, u, v, w^S, w^C)$ and $(\underline{\mathsf{id}}, \underline{u}, \underline{v}, \underline{w^S}, \underline{w^C})$ as well as the client's entry $(C, m, \underline{m})$.
   (b) For each entry $c'_j = 0$ within $\underline{c}'$, verify that $r_{j,1} = v_j$ and $r_{j,2} = w_j^C$. If this is wrong for any c'_j, ignore this message.
   (c) Compute $x = (E_0, \mathsf{pk}', E'_S, \beta'; \underline{\hat{E}}', \underline{\hat{F}}')$ as in the real execution of the protocol. If the entry $(\mathcal{H}, x, \underline{c})$ with $\underline{c} = \underline{c}'$ exists, continue, otherwise ignore this message.
   (d) Pick a j for which $c_j = 1$ and extract $\mathsf{sk}' = \beta'_j + v_j - r'_j$.
   (e) Verify that $\mathsf{pk}' \stackrel{?}{=} [\mathsf{sk}']E_0$ and $[w^C + \beta' m]E_S \stackrel{?}{=} [\mathsf{sk}' \cdot m]E_0 =: E_m$. If this is not the case, return $\textsc{Fail}$ to the environment, otherwise store $(S, \mathsf{sk}', \mathsf{pk}')$.

(f) Return $(\textsc{ClientComplete}, C, \mathsf{pk}')$ to $\mathcal{F}_{\mathsf{VOPRF}}$.

7. On queries x to $\mathsf{H_O}$, the simulator proceeds as follows.
   (a) If a record $(\mathsf{H_O}, x, y)$ exists, output y.
   (b) If x is of the form (m', pk', E') and there exists an entry $(S, \mathsf{sk}, \mathsf{pk})$ with $\mathsf{pk} = \mathsf{pk}'$, and $E' = [m' \cdot \mathsf{sk}]E_0$, then send $(\textsc{Eval}, \Pi, m')$ and $(\textsc{ClientComplete}, \mathcal{S}, \mathsf{pk}')$ to $\mathcal{F}_{\mathsf{VOPRF}}$. If $\mathcal{F}_{\mathsf{VOPRF}}$ aborts, output Fail, otherwise sample and output $y \leftarrow \{0,1\}^\ell$ and create the record $(\mathsf{H_O}, x, y)$.
   (c) In all other cases, sample and output $y \leftarrow \{0,1\}^\ell$ and create the record $(\mathsf{H_O}, x, y)$.

We define the following hybrids, gradually moving from the real protocol to the simulated version.

Hyb_0**:** This is the real execution of the protocol.
Hyb_1**:** Sim simulates the following interfaces:
- Sim simulates $\mathcal{F}_{\mathsf{TG}}$ as in step 1.
- Sim simulates $\mathcal{H}$ as in step 2.
- Sim simulates the honest server as in step 3.

Hyb_2**:** Sim also simulates the honest client as in step 4.
Hyb_3**:** Sim also simulates the honest client as in step 6.
Hyb_4**:** Sim also simulates the honest server as in step 5.
Hyb_5**:** This is the completely simulated execution of the protocol, i.e. Sim also simulates $\mathsf{H_O}$ as in step 7.

The following lemmas argue indistinguishability of each hybrid jump.

Lemma 1. Hyb_0 *and* Hyb_1 *are perfectly indistinguishable.*

Proof. Clearly, Sim simulates the functionality $\mathcal{F}_{\mathsf{TG}}$, the random oracle $\mathcal{H}$, and the honest server exactly as in the real execution of the protocol, which implies that both protocols are perfectly indistinguishable.

Lemma 2. Hyb_1 *and* Hyb_2 *are perfectly indistinguishable.*

Proof. The simulator samples $\alpha, \underline{\alpha}$ at random. In the real execution of the protocol, the honest user computes $\alpha = m - u$ and $\alpha_j = m - u_j$, with m unknown to the adversary and with u and $\underline{u}$ unknown to the adversary and uniformly sampled. Therefore, the simulated $\alpha, \underline{\alpha}$ follow the exact same distribution as in the real protocol.

Lemma 3. Hyb_2 *and* Hyb_3 *are indistinguishable under the assumption that Problem 2 is hard.*

Proof. We start with step (6b), in which the simulator checks consistency of the response in the case $c'_j = 0$ with the generated tuples. In the real protocol, the verifier checks whether the two elements of the response satisfy the correct linear combination. Clearly the latter is satisfied by more elements than only the tuples, yet since the prover does not know the correct linear combination, a

successful malicious prover would need to guess the correct linear combination, which it can only with a probability of $1/q$ per index j.

In step (6c), the simulator, using its knowledge of the client's input and generated tuples, computes the statement $x = (E_0, \mathsf{pk}', E'_S, \beta'; \underline{\hat{E}'}, \underline{\hat{F}'})$ exactly as it would in the real execution of the protocol. Since Sim simulates the hash function, it can immediately check whether an entry $(\mathcal{H}, x, \underline{c})$ already exists.

- If this entry does not exist or if the entry exists and $\underline{c} \neq \underline{c}'$, then the simulation ignores the input by S.
- If this entry exists and $\underline{c} = \underline{c}'$, the simulation assumes that the server must have queried $\mathcal{H}$ on input x and continues.

The first case simulates the verification of the server input failing, while the second case implies acceptance by the client. In both cases it is assumed that the server has previously queried x to the random oracle $\mathcal{H}$. However, in the real execution of the protocol, the server can also try to guess $\underline{c}$ correctly without querying x. The probability for the latter to happen is inversely proportional to the output size of the random oracle $\mathcal{H}$, thus $2^{-\lambda}$, chosen to be negligible in the security parameter. Another possibility is that some other party might have previously queried $\mathcal{H}$ on x. Since this amounts to guessing $\underline{\hat{E}'}, \underline{\hat{F}'}$, i.e. 2λ elliptic curves correctly, this probability is much lower, bounded at $q^{-2\lambda}$ (using the fact that the group action is regular).

Assuming now that the verification succeeded, the simulator extracts sk' in (6d), then verifies whether the server's input actually agrees with the specification of the OPRF, i.e. whether indeed $(E_0, \mathsf{pk}', E_m, m) \in \mathcal{L}_{\mathsf{Scal}}$. If this is not the case, then the simulator outputs Fail. This output is clearly different from the real execution of the protocol, and represents the event in the real world, in which the verifier accepts the proof $(\underline{\beta'}, \underline{c'}, \underline{r'})$ for a wrong statement $(E_0, \mathsf{pk}', E'_S, \beta')$. In the following, we argue why the extraction in (6d) works and that Fail only occurs with a negligible probability.

To this end, assume $(\underline{\beta}, \underline{c}, \underline{r})$ is an accepting tuple. Let $\mathbb{O} \subset \{1, \ldots, \lambda\}$ be the subset of occurrences where $c_j = 0$ and $\mathbb{I} = \{1, \ldots, \lambda\} \backslash \mathbb{O}$, where $c_j = 1$. Due to the verification conditions, we know that

$$\begin{aligned} &\text{for } j \in \mathbb{O}, \quad \hat{E}_j = [\beta_j + r^{(0)}_{j,1}]E_0 \quad \text{and} \quad \hat{F}_j = [r^{(0)}_{j,1} m - w^C_j]E_0\,, \\ &\text{for } j \in \mathbb{I}, \quad \hat{E}_j = [r^{(1)}_j]\mathsf{pk} \quad \text{and} \quad \hat{F}_j = [r^{(1)}_j m - \beta^{(1)}_j m - w^C_j]E_m\,. \end{aligned}$$

Note that the expression for $\hat{F}_j$ for $j \in \mathbb{O}$ follows from the fact that $w^C_j + r^{(0)}_{j,2} = u_j r^{(0)}_{j,1}$ and $\alpha_j = m - u_j$, and therefore

$$r^{(0)}_{j,2} + \alpha_j r^{(0)}_{j,1} = r^{(0)}_{j,2} + m r^{(0)}_{j,1} - u_j r^{(0)}_{j,1} = m r^{(0)}_{j,1} - w^C_j\,.$$

First, let us argue why the fulfillment of both verification conditions implies that $(E_0, \mathsf{pk}', E_m, m) \in \mathcal{L}_{\mathsf{Scal}}$, then we show how the secret can be online extracted. If both conditions would be fulfilled for the same j, then we would have the following.

$$[\beta_j + r^{(0)}_{j,1}]E_0 = [r^{(1)}_j]\mathsf{pk}, \quad \text{and} \quad [r^{(0)}_{j,1} m - w^C_j]E_0 = [r^{(1)}_j m - \beta_j m - w^C_j]E_m\,,$$

therefore

$$\mathsf{pk} = [\beta_j + r_{j,1}^{(0)} - r_j^{(1)}]E_0, \quad \text{and} \quad E_m = [(\beta_j + r_{j,1}^{(0)} - r_j^{(1)})m]E_0\,.$$

Clearly this implies that $(E_0, \mathsf{pk}', E_m, m) \in \mathcal{L}_{\mathsf{Scal}}$ with witness $\mathsf{sk} = \beta_j + r_{j,1}^{(0)} - r_j^{(1)}$. From our argument related to point (6b), we know that $r_{j,1}^{(0)} = v_j$, except with negligible probability. Therefore, the online extraction in step (6d) of the secret key succeeds, except with negligible probability. In the end, this implies that the simulated execution of Hyb_3 is indistinguishable from Hyb_2, assuming that solving instances of $\mathcal{L}_{\mathsf{scal}}$ is hard. Note that [44] has shown this problem is equivalent to the Parallelization Problem 2.

Lemma 4. Hyb_3 *and* Hyb_4 *are perfectly indistinguishable.*

Proof. E_S is constructed exactly as in the real execution of the protocol. Similarly, in both executions, $\underline{c}$ is produced uniformly at random. The values β and $\underline{\beta}$ are sampled at random, which is indistinguishable from the real protocol, where they are constructed via the linear combination of two random values unknown to the client. The same argument applies to r_j in the case $c_j = 1$. We note that in the real execution, the client can compute $r_j - \beta_j = \mathsf{sk} - v_j$. Again, this is indistinguishable to the simulated execution of the protocol, as this value also looks uniformly random to the client in both cases. Finally, for $r_j = (r_{j,1}, r_{j,2})$ in the case $c_j = 0$, the prover discloses two values, that fulfill the linear combination $u_j r_{j,1} = w_j^C + r_{j,2}$, both in the real and the simulated execution. Since in both cases, these values are sampled uniformly at random, these two cases are also indistinguishable.

We finish by arguing why maliciously chosen $\underline{\alpha}'$ do not compromise the security of the protocol. Note that in the real execution of the protocol, there is no way for the prover to verify whether α_j has been computed according to the protocol description or not. Since u_j is fixed through the tuple id_j, this effectively changes m. This is captured in the simulation by distinguishing different m_j for each j, and not changing its behavior in case $m_j \neq m$, making the real execution indistinguishable from the simulated one also in this case. The key point why this is not an issue is to note that the client actually never learns $\hat{F}_j$, but is simply meant to walk there using its own knowledge and the prover's output. In both cases $c_j = 0$ or 1, however, the information sent by the prover is completely independent from α_j, so choosing this maliciously also doesn't allow the client to gain any extra information. In fact, a malicious input α_j only implies how $\hat{F}_j$ is computed by the prover, but the resulting information only ends up as an input to the random oracle, effectively erasing any potential knowledge the malicious client could gain.

Lemma 5. Hyb_4 *and* Hyb_5 *are indistinguishable under the assumption that Problem 3 is hard.*

Proof. In this step, the simulator runs the random oracle $\mathsf{H_O}$ as in the real protocol, i.e. returning random bitstrings of length ℓ for any given input, with

one notable exception. In case the client queries a well-formed OPRF output query of the form (m', pk', E), the simulator sends a $(\textsc{ClientComplete}, \mathcal{S}, \mathsf{pk}')$ message to $\mathcal{F}_{\mathsf{VOPRF}}$ and outputs Fail in case $\mathcal{F}_{\mathsf{VOPRF}}$ aborts. The latter only happens, if $\mathcal{F}_{\mathsf{VOPRF}}$ cannot find an entry (C, S, m), i.e. when the client tries to hash a statement of the form $(m, \mathsf{pk} = [\mathsf{sk}]E_0, [\mathsf{sk} \cdot m]E_0)$ without having interacted with the server on input message m. In the following, we show that a client is unable to form such a *critical* query, except with negligible probability. We show this, by demonstrating that a client able to form a critical query can be used as a subroutine to break Problem 3.

To this end, engage with the client in the OPRF protocol, with the public key set to $\mathsf{pk} \leftarrow \mathcal{O}_{\mathsf{sk}}(1)$ and proceed as follows.

1. Whenever the client sends an OPRF input α for a predetermined id, recover $m = \alpha + u$ using the stored tuple $(\mathsf{id}, u, v, w^S, w^C)$ and query the oracle on $E_m \leftarrow \mathcal{O}_{\mathsf{sk}}(m)$. Then compute $E_S = [-w^C - \beta m]E_m$ and continue exactly as in step (5d) of the simulation. Send $(E_S, \beta, \underline{\beta}, \underline{c}, \underline{r})$ to the client.
2. Whenever the client makes a query to H_{O}, if it is not a critical query, execute H_{O} truthfully. Otherwise, if the client makes the critical query (m, pk, E), output the curve E as a solution to Problem 3.

Since Problem 3 is assumed to be hard, we can conclude that the event Fail in step 7 only occurs in the case where the client can break this assumption, thus with negligible probability in the security parameter.

We can conclude that the real protocol and the simulated execution are computationally indistinguishable for the environment. This concludes the proof of Theorem 1.

4 Thresholdization

In this section, we present a threshold version of our OPRF protocol presented in Figure 4. Note that our protocol is described almost completely using the subroutines introduced in Figure 3, with the exception of routines that are run locally and/or publicly, such as random oracle queries. The goal of this section is to introduce distributed algorithms, which *realize* (using simulation arguments) these single-party subroutines, allowing us to effectively replace single-party instantiations of these algorithms with multi-party ones, while maintaining the overall protocol structure and security guarantees.

In a threshold setting, these subroutines get the following new interpretations.

- $\mathsf{Samp}(\mathsf{pp})$: Sample a shared secret $\langle s \rangle \leftarrow \mathbb{Z}_q$.
- $\mathsf{Rec}(\mathbf{a}, \mathbf{s})$: Reconstruct and return a linear combination of shared secrets $\langle \mathbf{s} \rangle = (\langle s_1 \rangle, \ldots, \langle s_k \rangle)$ with public parameters $\mathbf{a} = (a_1, \ldots, a_k)$, i.e.

$$\mathbf{a} \cdot \mathbf{s} = \sum_{i=1}^{k} a_i s_i \,.$$

– $\mathsf{GA}(\mathbf{a}, \mathbf{s}, E)$: Compute and return the group action $[\mathbf{a}\cdot\mathbf{s}]E$, where $\mathbf{a}$ are public parameters and where the elements of $\mathbf{s}$ are secret shared.

In short, the protocols we are looking for in order to realize these subroutines are verifiable secret sharing and distributed group action computations. Verifiable secret sharing schemes provide a way to cover the Samp and Rec routines as they allow the (verifiable) distributed sampling of secrets, as well as a way to reconstruct them. The GA algorithm can be realized by distributed group action computation schemes, which usually occur in a round-robin structure in the isogeny literature.

Since we want to realize single-party functionalities, our distributed protocols should be guaranteed to finish with a correct output, even in the presence of malicious adversaries. There are multiple ways to achieve this in a distributed setting, but here we go for the strongest such assumption, which is *robustness*. This property guarantees that any distributed protocol, once started, always ends by producing an output, which is guaranteed to be correct.

In this section, we present a threshold instantiation from bivariate secret sharing in the malicious setting, assuming at most $t < n/3$ corruptions (honest supermajority). In Appendix B, we also present an instantiation in the semi-honest setting, for which we assume a higher corruption threshold of $t < n/2$ (honest majority setting). Throughout this work, we assume to work in a (partially) synchronous model, and assume pairwise authenticated channels between the parties, as well as a broadcast channel.

Subroutine functionality. To achieve our goals, we define an ideal functionality, which serves as an interface between the single- and multi-party instantiation of the upper routines. This functionality simply runs the subroutines $(\mathsf{Samp}, \mathsf{Rec}, \mathsf{GA})$ exactly as in the single-party case, but does so only under the conditions realized in a multi-party setting. What this means is that parties can join the sampling, reconstruction and group action routines, but these routines only provide outputs (distributed like in the single-party protocol) whenever a subset of at least $t + 1$ parties agree via sending the relevant message to the functionality. Furthermore, parties cannot sample a shared secret, learn anything about a secret or compute a group action without at least $t + 1$ parties collaborating. We present the subroutine functionality $\mathcal{F}_{sr}$ in Definition 7. At the beginning, the functionality $\mathcal{F}_{sr}$ learns who the corrupted parties are.

The command Tuple prompts the functionality to query $\mathcal{F}_{\mathsf{TG}}$ for tuples. The tuples are output normally to the client, while in the server case, the functionality records the output of $\mathcal{F}_{\mathsf{TG}}$ without sending it to the parties in T. Instead, the functionality merely informs the parties that an element was generated via the output TupleSampled. This represents the fact that the parties don't learn any information about the shared secret tuple, except for its existence.

The functionality behaves similarly, when the shared secret sampling routine is run with the command Sample. It samples and stores a random element with some associated element identification string *eid*, and simply informs the server parties that such an element exists. Note that since the secrets, including tuple

elements, are stored by the functionality and the parties can only refer to them using their *eid*.

When the reconstruction routine is initiated with the RECONSTRUCT command and the requesting set of parties T, the functionality first checks whether this set is qualified, i.e. by checking that enough parties in T actually all possess shares of the relevant secrets. If this is the case, $\mathcal{F}_{\mathsf{sr}}$ computes the requested linear combination using the elements it has previously stored, and outputs it. This captures the fact that any qualified set T, which joined the sampling of any set of secrets, can also reconstruct it, as long at least $t+1$ shares of each of those secrets are held by the parties in T. Similarly, if a group action computation is requested with GROUPACTION by all parties in some determined set T, under the same conditions, the functionality recovers the correct linear combination from its stored element and applies it the the elliptic curve in question, then returns the output.

Definition 7 (Subroutine functionality $\mathcal{F}_{\mathsf{sr}}$).

1. On input $(\text{TUPLE}, \mathsf{id}, C, T)$ by a party $P \in \{C, T\}$, where T a set of at least $t+1$ server parties with identifier S, query $(\text{TUPLE}, \mathsf{id}, C, S)$ to $\mathcal{F}_{\mathsf{TG}}$. If P is the client, send (u, w^C) to C, otherwise, sample two identifiers eid_v and eid_w and create the records (T, eid_v, v) and (T, eid_w, w^S) and send $(\text{TUPLESAMPLED}, T, eid_v, eid_w)$ to the parties in T. Finally, send (OUTPUT, P) to $\mathcal{A}$.
2. On input (SAMPLE, T) by a set T of parties with $|T| > t$, sample an identifier eid and a secret $s_{eid} \leftarrow \mathbb{Z}_q$ and record (T, eid, s_{eid}). Finally, output $(\text{SAMPLED}, T, eid)$ to the parties in T.
3. On input $(\text{RECONSTRUCT}, T, \mathbf{a}, \mathbf{eid})$ by a set of parties T, verify that $\#\mathbf{a} = \#\mathbf{eid}$ and that for each $eid_j \in \mathbf{eid}$, there exist a record (T_j, eid_j, s_{eid_j}) such that $\left|T \cap \bigcap_j T_j\right| > t$ holds. If these checks are successful, compute $r = \sum_j a_j s_{eid_j}$ and output $(\text{RECONSTRUCTED}, T, \mathbf{a}, \mathbf{eid}, r)$ to the parties in T, otherwise ignore this request.
4. On input $(\text{GROUPACTION}, T, \mathbf{a}, \mathbf{eid}, E)$ by a set of parties T, verify that $\#\mathbf{a} = \#\mathbf{eid}$, that $E \in \mathcal{E}$, and that for each $eid_j \in \mathbf{eid}$, there exist a record (T_j, eid_j, s_{eid_j}) such that $\left|T \cap \bigcap_j T_j\right| > t$ holds. If these checks are successful, compute $E' = \left[\sum_j a_j s_{eid_j}\right] E$ and output $(\text{GADONE}, T, \mathbf{a}, \mathbf{eid}, E, E')$ to the parties in T, otherwise ignore this request.

4.1 Malicious threshold VOPRF

In this section, we present a way to thresholdize our OPRF construction in a maliciously secure way. To be able to seamlessly replace single-party computations with multi-party ones, as we did in the previous section, we do not only need to secure secret data from maliciously behaving parties, but we also need to ensure that our computation actually completes with a correct output, a property which is called *robustness*. To achieve this, we opt for an honest-supermajority

(t, n)-threshold setting with a symmetric bivariate sharing and assume t and n to be part of the public parameters throughout this section. In particular, these choices will allow us to reconstruct linear combinations of secrets in a robust way. Fortunately, some research has been performed into realizing (symmetric) bivariate verifiable secret sharing [7, 18, 36], as well as threshold group actions in a bivariate setting [9, 26]. However, neither of these sources treats the case of linear combinations of shares, so we have to modify their constructions to fit our purpose. Furthermore, existing group action based schemes use general bivariate polynomials, which we will adapt to the symmetric case. For a more in-depth discussion of their design choices, we refer to the respective sources. We discuss our VSS and the distributed group action protocols below. Throughout this section, we define $\mathcal{B}$ as the ring of symmetric bivariate polynomials in $\mathbb{Z}_q[X, Y]$ with degree at most t in each of the two variables.

4.1.1 A bivariate VSS with linear reconstruction. We start by presenting a verifiable secret sharing scheme for symmetric bivariate polynomials. Our scheme combines some of the ideas from previous verifiable secret sharing schemes from the literature, but with some modifications to consistently fit our requirements. The main motivation of our construction is to reduce both the communication cost and the number of rounds with respect to the original bivariate design by Ben-Or, Goldwasser and Widgerson [18]. The main ingredients of our VSS (which we explain in more detail below) are the following.

The dealer fixes the bivariate polynomials $F(X, Y)$ encoding the secret by committing to them on a broadcast channel, similar to the approach from [11]. However, we modify these original commitments with the approach introduced by Baghery in [12], i.e. the prover also commits to the evaluations of a second polynomial $R(X, Y)$, which is used to prove consistency of the first one. Unlike [11] this proof system uses the random oracle model, which also results in a 2-round protocol, but with lower communication than the original design. Finally, our reconstruction step also allows the linear reconstructions of secrets. Unlike previous designs, we do not rely on the commitments from the VSS to verify correct reconstruction, since the verification of linear combinations would need either homomorphic commitments or convoluted zero-knowledge proofs. Instead, we simply rely on the superhonest majority to verify the reconstruction, preventing byzantine attacks. This also implies that after the initial verification of correct sharing, the commitments are no longer needed and can be discarded.

We define the commitment function $\mathsf{C} : \{0,1\}^* \to \{0,1\}^{2\lambda}$, which we assume to be collapsing [90] and quantum computationally hiding. For any given pair of symmetric bivariate polynomials $F(X, Y), R(X, Y) \in \mathcal{B}$, we define the following commitment matrix

$$\mathfrak{C}(F, R) = \begin{pmatrix} \mathsf{C}(F(1,1), R(1,1)) & \dots & \mathsf{C}(F(1,n), R(1,n)) \\ \mathsf{C}(F(2,1), R(2,1)) & \dots & \mathsf{C}(F(2,n), R(2,n)) \\ \vdots & \vdots & \vdots \\ \mathsf{C}(F(n,1), R(n,1)) & \dots & \mathsf{C}(F(n,n), R(n,n)) \end{pmatrix},$$

where we refer to the different entry using double indices, e.g. $\mathfrak{C}(F,R)_{ij} = \mathcal{C}(F(i,j),R(i,j))$.

A party sharing a bivariate polynomial $F(X,Y)$ constructs such a commitment with a proof randomizer $R(X,Y)$ and a commitment opening ρ and sends shares and their related openings to each party. To prove that these commitments are correctly structured, a linear combination $R(X,Y) - cF(X,Y)$ for a given challenge c is revealed, which allows parties to verify their own share against the public commitment $\mathfrak{C}(F,R)$.

We present our VSS in Figure 5. In the protocol, $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ denotes a random oracle. To stay as general as possible, we assume that both the sharing and reconstruction steps can be run by any subset $T \subseteq \{1,\ldots,n\}$ of the parties, assuming at least $t+1$ honest parties, and that to every successfully generated secret the subset of parties that generated it is known. This is especially important in the reconstruction phase, in which parties in a set T can only attempt to reconstruct secrets $\{s_i\}_{i=1}^k$ if

$$\left| T \cap \bigcap_{i=1}^{k} T_i \right| > t\,,$$

holds, where s_i was generated by the set of parties T_i. We add this as an extra condition to the reconstruction phase.

Theorem 2. *The verifiable secret sharing scheme from Figure 5 satisfies* secrecy *if the commitment scheme* C *is quantum computationally hiding, and satisfies* correctness *if the commitment scheme* C *is collapsing.*

Proof. **Secrecy:** We argue secrecy by defining a simulator Sim that interacts with the adversary $\mathcal{A}$ and shows that the adversary's view is simulatable for any secret. Let $M \subset T$ be the set of corrupted parties by $\mathcal{A}$ and let $H = T \backslash M$ be the set of honest parties simulated by Sim. Further, let $f_j^{(i)}(X)$ be the share which a malicious party P_j has received from an honest party P_i. On behalf of each honest P_i, Sim proceeds as follows.

1. Sample the random symmetric polynomial $F'^{(i)}(X,Y) \leftarrow \mathcal{B}$ such that it is consistent with the adversary's shares, i.e. $F'^{(i)}(j,X) = f^{(j)}(X)$.
2. Sample $c^{(i)} \leftarrow \{0,1\}^\lambda$, a random symmetric $Z^{(i)}(X,Y) \leftarrow \mathcal{B}$, then let $\mathfrak{S}_{jk}^{(i)} = \mathsf{C}(F'^{(i)}(j,k), Z^{(i)}(j,k) + c^{(i)}F'^{(i)}(j,k))$. Fill the rest of the commitment matrix $\mathfrak{S}^{(i)}$ with dummy commitments. Finally, reprogram the RO to $c^{(i)} \leftarrow \mathsf{H}(\mathfrak{S}^{(i)})$.
3. Publish the polynomial $Z^{(i)}(X,Y)$.

Clearly, the verification will succeed. Although the implicit secret is changed by the choice of the polynomial $F'^{(i)}$, the shares of the malicious parties are unchanged, making them indistinguishable from the real protocol. Note that this captures the fact that t shares of a polynomial of degree t do not uniquely determine the polynomial, and that in fact every secret in $\mathbb{Z}_q$ would be possible. The only "information" about the other shares is hidden in the commitments

$\mathsf{M.Share}(\mathsf{pp}, T)$**:**

1. Each party P_i in T
   (a) samples a secret symmetric bivariate polynomial $F^{(i)}(X,Y) \leftarrow \mathcal{B}$ and sends the share $F^{(i)}(j,X)$ to P_j.
   (b) samples the proof randomizer $R^{(i)}(X,Y) \leftarrow \mathcal{B}$, then publishes the (symmetric) commitment matrix $\mathfrak{S}^{(i)} := \mathfrak{C}(F^{(i)}, R^{(i)})$.
   (c) samples a challenge $c^{(i)} \leftarrow \mathsf{H}(\mathfrak{S}^{(i)})$ and
   (d) reveals the polynomial $Z^{(i)}(X,Y) = R^{(i)}(X,Y) - c^{(i)}F^{(i)}(X,Y)$.
2. Each party P_j, for each share received by another party P_i
   (a) verifies whether $F^{(i)}(j,X) \in \mathbb{Z}_q[X]_t$ and $Z^{(i)}(X,Y) \in \mathcal{B}$.
   (b) computes $c^{(i)} \leftarrow \mathsf{H}(\mathfrak{S}^{(i)})$ and verifies whether for all $k \in T$

$$\mathfrak{S}^{(i)}_{jk} = \mathsf{C}\Big(F^{(i)}(j,k), Z^{(i)}(j,k) + c^{(i)}F^{(i)}(j,k)\Big).$$

   (c) Whenever any check doesn't pass, a party P_j issues a complaint $\mathsf{complaint}_{ji}$ against the dealer P_i. Then
      i. Any party without any complaints against them proceeds to step 3.
      ii. Any party with more that t complaints is disqualified.
      iii. For each other complaint $\mathsf{complaint}_{ji}$, party P_i discloses $F^{(i)}(j,X)$, then each other party runs the verification steps (2a) and (2b) on the broadcast information. If this check fails, P_i is disqualified, otherwise it proceeds to step 3.
3. Let $T_s \subseteq T$ denote the subset of qualifying parties. The polynomial $F(X,Y) = \sum_{i\in T_s} F^{(i)}(X,Y)$ defines the shared secret $s = F(0,0)$. Parties compute the verification shares $F(j,X) = \sum_{i\in T_s} F^{(i)}(j,X)$, which also defines their Shamir share $\langle s\rangle_j = F(j,0) = \sum_{i\in T_s} F^{(i)}(j,0)$. of the secret. The commitments $\mathfrak{S}^{(i)}$ are discarded and the protocol outputs $[\![s]\!]_{T_s}$.

$\mathsf{M.Reconstruct}(\mathsf{pp}, a_1, \ldots, a_k, [\![s_1]\!]_T, \ldots, [\![s_k]\!]_T, T)$**:**

1. Let T_i be the set that generated $[\![s]\!]$. If $|T \cap \bigcap_{i=1}^{k} T_i| \leq t$ return $\perp$.
2. Let $[\![s_l]\!]_i = F_l(i,X)$ for $l = 1, \ldots, k$. Each party P_i computes their share of the output locally as $G(i,X) = \sum_{l=1}^{k} a_l F_l(i,X)$ and publishes it.
3. From the opened polynomials, pick any subset which interpolates to a symmetric bivariate polynomial $G'(X,Y) \in \mathcal{B}$ and output $G'(0,0)$.

Fig. 5: Symmetric bivariate verifiable secret sharing

of the honest parties' shares. Any adversary who could use this information to their advantage would however break the hiding property of the commitment scheme. Finally, since $Z^{(i)}$ and $c^{(i)}$ are uniformly sampled, they follow the same distribution as in the real execution of the protocol. Note that we assume that the committed values have enough entropy, which is the case, since we sample the polynomials randomly from $\mathcal{B}$.

Correctness: For any dealing party that progresses to step 3 of the sharing protocol, we have that at most t parties complained. If there are no complaints,

then clearly all honest parties agree. Each complaint is verified using broadcast information, implying also that all of the honest parties will agree after this step. This means that for every party that progressed to step 3, all honest parties agree that $Z^{(i)}(X,Y) \in \mathcal{B}$ and that $Z^{(i)}(X,Y) = R^{(i)}(X,Y) - c^{(i)}F^{(i)}(X,Y)$ for a randomly generated $c^{(i)}$. Using a special soundness argument, we show that this implies that $F^{(i)}(X,Y) \in \mathcal{B}$, i.e. assume we have two accepting transcripts

$$(\mathfrak{S}^{(i)}, c^{(i)}, Z^{(i)}(X,Y)), \quad \text{and} \quad (\mathfrak{S}^{(i)}, c'^{(i)}, Z'^{(i)}(X,Y))\,,$$

then for all honest j, it must for all $k \in T$, that

$$\mathsf{C}\Big(F^{(i)}(j,k), Z^{(i)}(j,k) + c^{(i)}F^{(i)}(j,k)\Big) = \mathsf{C}\Big(F^{(i)}(j,k), Z'^{(i)}(j,k) + c'^{(i)}F^{(i)}(j,k)\Big)\,.$$

Unless the collapsing property of the commitment can be broken, this implies that $Z^{(i)}(j,k) + c^{(i)}F^{(i)}(j,k) = Z'^{(i)}(j,k) + c'^{(i)}F^{(i)}(j,k)$ and thus that

$$F^{(i)}(j,k) = \frac{Z^{(i)}(j,k) - Z'^{(i)}(j,k)}{c'^{(i)} - c^{(i)}}$$

for all $k \in T$ and at least $n - t$ honest indices j. Since $Z^{(i)}(X,Y) \in \mathcal{B}$, clearly this implies that $F^{(i)}(X,Y) \in \mathcal{B}$, implying correctness.

Robustness: As a consequence of correctness, we know that in the reconstruction phase, the honest parties hold consistent shares of all the polynomials $F_l(X,Y)$ for $l = 1, \ldots, k$. The linearity of the secret sharing scheme also implies that any linear combination of shares with public coefficients leads to the honest parties having consistent shares of $G(X,Y)$. Since there are at least $2t+1$ honest parties, even if there are up t maliciously introduced shares by the adversary there will be (at least) $n - t = t + 1$ consistent polynomials $G(i,X)$ which interpolate to a unique (overdetermined) symmetric bivariate polynomial $G(X,Y)$.

4.1.2 Distributed group action computation.

We show how to realize the distributed group action computation in Figure 6. We note that our construction is an extension of the group action computation from [9] to the setting of linear combinations of secrets. In particular, the group action computation employs *structured piecewise verifiable proofs* (SPVP) as a means to convince other parties that the computation has been executed correctly. We revisit SPVPs in Appendix A.2 and just briefly recall their language here for a single pair of curves (the original design allowed multiple such pairs)

$$\mathcal{L}^T_{SPVP} = \{(E_0, E_1, a) \mid \exists f(X) : E_1 = [af(0)]E_0 \wedge \{x_i = f(i)\}_{i \in T}\}\,.$$

In our protocol, SPVPs are used as a way to prove that a party has acted with a multiple a of a constant term $f(0)$ of a polynomial, of which all the other parties hold shares $f(i)$. In our case, these scalars are (publicly known) Lagrange interpolation polynomials.

Note that the real magic of bivariate sharings is that parties can verify each other's shares in the reconstruction or group action computation simply by using their own shares. This eliminates the need for explicit commitments, as would be the case in a Shamir sharing. The main reason we do this is to remove the need of computing commitments in the correlated tuple generation, which allows us to instantiate this functionality at low cost and using standard methods.

$\mathsf{M.GA}(\mathsf{pp}, a_1, \ldots, a_k, [\![s_1]\!]_T, \ldots, [\![s_k]\!]_T, T, E)$:

1. Let T_i be the set that generated $[\![s_i]\!]$. If $|T \cap \bigcap_{i=1}^{k} T_i| \leq t$ return $\perp$.
2. Let $[\![s_l]\!]_i = F_l(i, X)$ for $l = 1, \ldots, k$. Each party P_i computes their share of the linear combination locally as $G(i, X) = \sum_{l=1}^{k} a_l F_l(i, X)$.
3. In a round-robin way, each party P_k for $k \in T$ computes $E' \leftarrow [L_k^T G(k, 0)]E$ and runs
$$(\tilde{\pi}^{(k)}, \{\pi_i^{(k)}\}_{i \in T}) \leftarrow \mathsf{SPVP.Prove}(L_k^T, G(k, X), (E, E'), \{G(k, i)\}_{i \in T}),$$
then publishes $(E', \tilde{\pi}^{(k)})$ and sends $\pi_i^{(k)}$ to each other player P_i.
4. Each other player P_i runs $\mathsf{SPVP.Verify}(L_k^T, (E, E'), G(k, i), \tilde{\pi}^{(k)}, \pi_i^{(k)})$. If the verification fails, P_i complains. Then P_k broadcasts $(G(k, i), \pi_i^{(k)})$. If this satisfies the SPVP, continue. If P_k doesn't broadcast anything, or it doesn't satisfy the SPVP, each other party P_j publishes their shares $\{G(j, k)\}$, and parties reconstruct $G(k, X)$, then compute $E' \leftarrow [L_k^T G(k, 0)]E$. At the end of this step, set $E \leftarrow E'$.
5. At the end of the round-robin, return E.

Fig. 6: Threshold group action computation.

Theorem 3. *If* M.Share *is correct and* SPVP *is sound, then* M.GA *is correct and robust. If* M.Share *satisfies privacy and* SPVP *is zero-knowledge, then* M.GA *satisfies secrecy.*

Proof. This theorem immediately follows from [9, Theorem 3.5], which proves robust correctness and secrecy of a distributed group action computation in the honest supermajority setting from bivariate sharings. We note that their theorem is more general and also holds for non-symmetric polynomials and supports batched proofs. The main difference between our setting and theirs is that we allow the linear combination of shares. Due to the linearity of the secret sharing scheme and the correctness of M.Share, we know that all honest parties agree on the polynomials $F_l(X, Y)$ and therefore also on the polynomial $G(X, Y)$. This implies that parties have correct sharing of a bivariate polynomial in $\mathcal{B}$. Since the honest parties form a supermajority and the SPVPs are sound, robustness is implied exactly as in [9, Theorem 3.5]. Finally, the same arguments for secrecy hold, since the only difference is the computation of linear combinations, which is done locally, and therefore does not provide the adversary with any more information.

4.1.3 Tuple generation functionality. In Definition 8, we introduce the tuple generation functionality for shares in $\mathcal{B}$. For simplicity we assume that a set T of server parties simultaneously query the functionality for their tuples.

Definition 8 (Bivariate tuple generation functionality $\mathcal{F}_{\mathsf{M.TG}}$).

1. On the first activation, learn the list of corrupted parties from the adversary.
2. On input $(\textsc{Tuple}, \mathsf{id}, C, T)$ by a party P, proceed as follows.
   - If there is no recorded entry for id, sample polynomials $u, v, w^S, w^C \leftarrow \mathbb{Z}_q$ (using pp) with $w^C + w^S = uv$, then generate bivariate sharings $[\![v]\!]_T, [\![w^S]\!]_T$ and record $(C, S, \mathsf{id}, u, w^C, [\![v]\!]_T, [\![w^S]\!]_T)$.
   - Otherwise, retrieve $(C, S, \mathsf{id}, u, w^C, [\![v]\!]_T, [\![w^S]\!]_T)$.
3. Then,
   - If P is the client, send (u, w^C) to P and send $(\textsc{Output}, C)$ to $\mathcal{A}$.
   - If P is the set T of server parties, send $([\![v]\!]_T, [\![w^S]\!]_T)$ to P and send $(\textsc{Output}, T)$ to $\mathcal{A}$.

4.1.4 Plugging in. To show that we can replace the original routines with the maliciously secure ones, we use the subroutine functionality defined in Definition 7, which acts as an interface between the one-party computation and its multi-party counterpart, now in the malicious setting. $\mathcal{F}_{sr}$ perfectly realizes the one-party computations embedded in a multi-party setting. We prove in in Section 4.2, that we can construct a simulator interacting with the adversary in such a way, that the adversary cannot distinguish whether it is in the real execution or the simulated execution of the protocol, thus again showing that the multi-party algorithms realize the single-party algorithms.

4.2 Proof in the malicious threshold

We prove that the algorithms $(\mathsf{M.Share}, \mathsf{M.Reconstruct}, \mathsf{M.GA})$ from Figures 5 and 6, together with the tuple generation functionality from Definition 8 realize the functionality $\mathcal{F}_{sr}$ from Definition 7. We show this by defining the following simulator Sim which has access to the subroutine functionality $\mathcal{F}_{\mathsf{sr}}$.

Our simulator interacts with the adversary $\mathcal{A}$ on behalf of the honest parties in the protocol. Instead of executing the honest parties as in the protocol description, the simulator runs the respective subroutines by querying the subroutine functionality $\mathcal{F}_{\mathsf{sr}}$. We prove security with a set of hybrids going from the real protocol execution to the simulated one.

$\mathcal{F}_{\mathsf{M.TG}}$**:** Whenever Sim receives a query $(\textsc{Tuple}, \mathsf{id}, C, T)$ for $\mathcal{F}_{\mathsf{M.TG}}$

1. Send $(\textsc{Tuple}, \mathsf{id}, C, T)$ to $\mathcal{F}_{\mathsf{sr}}$ and transmit the outputs to the corresponding parties.
2. Sample random polynomials $R_v(X, Y), R_w(X, Y) \leftarrow \mathcal{B}$, and send $(R(i, X), R'(i, X))$ to each $P_i \in M$. Record the shares held by the adversarial parties and output $(\textsc{Output}, S, i)$ to $\mathcal{A}$.

$\mathsf{M.Share}(\mathsf{pp}, T)$**:** Sim interacts with the adversary $\mathcal{A}$ as follows.

1. On behalf of each honest party P_i, Sim sends random shares $F^{(i)}(j, X)$ to the adversarial parties P_j as in the real execution of the protocol, then proceeds exactly as the secrecy simulator of Theorem 2 to generate the adversary's view during the protocol. Record the shares held by the adversarial parties.
2. Verify each received input by an adversarial party and proceed in the complaint resolution as in the real protocol.
3. Let T_Q be the qualified set. Finally, query $(\textsc{Sample}, T_Q)$ to $\mathcal{F}_{\mathsf{sr}}$ and receive $(\textsc{Sampled}, T_Q, eid)$ and output it to the adversary.

$\mathsf{M.Reconstruct}(\mathsf{pp}, \mathbf{a}, \mathbf{eid}, T)$: Sim interacts with the adversary $\mathcal{A}$ as follows.

1. Query $(\textsc{Reconstruct}, T, \mathbf{a}, \mathbf{eid})$ to $\mathcal{F}_{\mathsf{sr}}$. Abort if this request is ignored. Otherwise, receive $(\textsc{Reconstructed}, T, \mathbf{a}, \mathbf{eid}, r)$.
2. Using knowledge of the shares held by the adversarial parties, sample a random $G(X, Y) \leftarrow \mathcal{B}$ with $G(0, 0) = r$ and such that its evaluations at the adversarial positions match with the adversarial shares. Then output $G(i, X)$ on behalf of each honest party P_i.
3. Proceed as in the real execution of the protocol. At the end, output $(\textsc{Reconstructed}, T, \mathbf{a}, \mathbf{eid}, r)$ to $\mathcal{A}$.

$\mathsf{M.GA}(\mathsf{pp}, \mathbf{a}, \mathbf{eid}, T, E)$:

1. Query $(\textsc{GroupAction}, T, \mathbf{a}, \mathbf{eid}, E)$ to $\mathcal{F}_{\mathsf{sr}}$. Abort if this request is ignored. Otherwise, receive $(\textsc{GADone}, T, \mathbf{a}, \mathbf{eid}, E, E_{\text{out}})$.
2. Sample a polynomial $G(X, Y) \leftarrow \mathcal{B}$, matching the corresponding linear combination of adversarial shares. Further, sample and fix any honest index i^*. Then, engage in the round-robin protocol with $\mathcal{A}$ and proceed as follows.
   (a) Whenever a malicious party P_k outputs $(E', \tilde{\pi}^{(k)})$ and honest parties P_i receive $\pi_i^{(k)}$, Sim first verifies the proof as in the real execution of the protocol and follows the complaint resolution procedure.
   (b) For each honest party except P_{i^*}, proceed exactly as in the real execution of the protocol, using $G(X, Y)$.
   (c) In the case of party P_{i^*}, assume S to be the index set of all parties, honest or malicious, following P_{i^*} in the round robin. Instead of computing E' correctly, compute $E' = \big[-\sum_{i \in S} L_i^T G(i, 0) \big] E_{\text{out}}$ using knowledge of these parties' shares. Finally use the SPVP zero-knowledge simulator to construct the proof $\tilde{\pi}^{(i^*)}$ with pieces

$\pi_k^{(i^*)}$ for malicious parties P_k. For any complaint raised by the malicious parties, Sim proceeds as in the real execution of the protocol.

3. At the conclusion of the round-robin, send $(\mathsf{GAdone}, T, \mathbf{a}, \mathbf{eid}, E, E_{\text{out}})$ to the parties in T.

We argue indistinguishability in the real and ideal protocols using the following hybrids.

Hyb_0**:** This is the real execution of the protocol.

Hyb_1**:** Instead of proving correct sharing in $\mathsf{M.Share}$, run the secrecy simulator of Theorem 2. Sample a random honest party P_{i^*}, then instead of proving correct computation of P_{i^*} in their round-robin step, run the SPVP zero-knowledge simulator.

Hyb_2**:** Instead of calling the tuple functionality $\mathcal{F}_{\mathsf{M.TG}}$, call $\mathcal{F}_{\mathsf{sr}}$ and send dummy shares to the malicious parties. Instead of honestly engaging in the sharing step, send dummy shares to the malicious parties, then call $\mathcal{F}_{\mathsf{sr}}$ on $(\textsc{Sample}, T_Q)$, where T_Q is the qualified set after verifying the adversarial shares. Then, in the reconstruction and group action steps, sample any random polynomial in $G(X, Y) \leftarrow \mathcal{B}$ matching the adversarial shares and engage in the protocols as in the real execution.

Hyb_3**:** In the reconstruction phase, query $(\textsc{Reconstruct}, T, \mathbf{a}, \mathbf{eid})$ from $\mathcal{F}_{\mathsf{sr}}$. Then proceed as before but sample $G(X, Y)$ such that also $r = G(0, 0)$ holds, where r is the reconstructed value by $\mathcal{F}_{\mathsf{sr}}$.

Hyb_4**:** In the group action computation phase, on behalf of party P_{i^*}, instead of computing E' correctly, compute E' as in step (2c) of the simulation.

Lemma 6. Hyb_0 *and* Hyb_1 *are indistinguishable under the assumption that the commitment scheme* C *is hiding.*

Proof. Both the secrecy of our VSS and the zero-knowledge property of SPVPs can be reduced to the hiding properties of the underlying commitment schemes, as is proven in Theorem 2 and in Theorem 4.1 of [9]. As a consequence, any adversary capable of distinguishing Hyb_0 from Hyb_1 can be used to break the hiding property of these commitment schemes.

Lemma 7. Hyb_1 *and* Hyb_2 *are perfectly indistinguishable.*

Proof. This lemma immediately follows from the fact that given t monovariate shares of a bivariate polynomial of degree t leave this polynomial undetermined, see e.g. [7, Claim 5.4]. This implies that the dummy shares sent to the malicious parties leave the secret undetermined (or are in fact correct shares for any possible secret). Thus, in the reconstruction and group actions steps, any uniformly sampled polynomial $G(X, Y) \in \mathcal{B}$ matching the (underdetermined) adversarial shares will be perfectly indistinguishable from any other potential choice.

Lemma 8. Hyb_2 *and* Hyb_3 *are perfectly indistinguishable.*

Proof. Following the discussion of Lemma 7, the freely chosen polynomial $G(X, Y)$ is now fixed to have the constant term $r = G(0, 0)$, which is a linear combination of elements output by $\mathcal{F}_{\mathsf{sr}}$. Note that the secret tuples sampled by $\mathcal{F}_{\mathsf{sr}}$ are uniformly random, exactly as in the real protocol, where they are sampled by $\mathcal{F}_{\mathsf{M.TG}}$ from Definition 8. Similarly, the secret sampled by $\mathcal{F}_{\mathsf{sr}}$ in the sharing step is uniformly random, while in the real protocol, the secrets are uniform since honest parties sample their polynomials uniformly. As a consequence, the reconstruction of a linear combination of secrets in $\mathsf{M.Reconstruct}$ is a uniformly random element both in Hyb_2 and Hyb_3. Note that the robustness property proven in Theorem 2 implies that this value is always correctly reconstructed.

Lemma 9. Hyb_3 *and* Hyb_4 *are perfectly indistinguishable.*

Proof. The regularity of the group action implies that for any pair $E, E' \in \mathcal{E}$, there exists a group element a, such that $E' = [a]E$. This means that for any E' chosen in step 2c, there exists such a link a. Due to the nature of the bivariate sharing, the adversarial parties however only hold t shares $\{G(i^*, j)\}_{j \in M}$ of the supposed link $L_{i^*}^T G(i^*, 0)$ between these two curves, which again underdetermines this value. As a consequence, any chosen link a is indistinguishable from any other chosen link, implying indistinguishability of the hybrids.

4.3 A note on verifiability and trust

We note that the outputs of *robust* schemes are by definition correct. We have even shown that our threshold OPRFs in fact realize honest single-party servers. So clearly, it would be a legitimate question to ask why one would care making outputs *verifiable* if they are guaranteed to be correct by our server-side assumptions anyways. There are two points we would like to make in that regard.

The first point is that in the event that the server key indeed gets leaked, for example by a corruption larger than the threshold, a client interacting with a verifiable OPRF can still verify whether the provided output is actually correct or not, while a client solely relying on the server-side security cannot. In such an event, the client, oblivious to the server-side compromise, might be fed biased outputs without realizing it, opening a whole lot of other problems when these outputs are used as part of other protocols.

Our second, and related argument, is that in a real world execution of such protocols, trust is a very dynamic notion. A set of servers realizing a TOPRF might mutually trust each other (up to some bound), but that does not mean that an external client querying these servers has the same trust in completely unbiased outputs. In fact, in our scheme, the client does not even need to know who the individual server parties are. Adding verifiability to the output completely removes the need for any such trust. On the downside, such an assumption comes at a higher cost, since a threshold execution of the proof system is still very expensive.

In a real-life scenario, such considerations will have to be weighed against one another. As we will see in Section 5, removing output verification greatly improves the efficiency of our protocols, in particular in a threshold setting.

5 Instantiation and costs

We discuss different instantiations of effective group actions from isogenies throughout this section and the resulting costs of our protocol.

5.1 Initial considerations

At the very basis of our scheme, we will need an effective group action according to Definition 1, for which Problem 3 is hard. This is not an easy task, but thankfully instantiations from isogenies have been explored throughout many works [22, 31, 38, 41, 79, 80].

In the isogeny effective group action setting, Problem 3 poses two important restrictions to the structure and size of the class group. First, in a group of composite order, an attacker could choose the input m maliciously to see how the secret sk acts on subgroups and extract information about it. Lai [69] has shown that security reduces to the size of the largest prime divisor of the group order. Second, [13] has introduced the following hard problem[3]

Problem 4 (Vectorization Problem with Shifted Inputs). Given an element $E_0 \in \mathcal{E}$ and the pairs $(m_i, [\mathsf{sk} \cdot m_i]E_0)_{i=1}^r$, where $\{m_1, \ldots, m_r\}$ is an exceptional set modulo q, find $\mathsf{sk} \in \mathbb{Z}_q$.

This problem is strictly harder than Problem 3, since r queries to the oracle $\mathcal{O}_s$ yield an instance of Problem 4, and finding s allows us to easily break Problem 3. A recent effort into cryptanalyzing this problem in the quantum setting [52] has led to the conclusion that it is indeed weaker than standard vectorization introduced in Problem 1, which we will have to take into account.

5.2 Honest-client instantiation from CSI-FiSh

The simplest way to instantiate our effective group action is in the context of CSI-FiSh, for which a particularly efficient parameter set of 128-bit classical security is readily available [22]. A major drawback of using the CSI-FiSh parameter set however is that the class group has composite order. There are a few consideration we need to take in that case. The first, which we have already outlined above, is that a malicious client could extract information about the secret of the server(s). As a consequence, in the case of a composite group order, our construction in its current form would be only secure in the case of honest clients.

Further, since our threshold schemes rely on Lagrange interpolation in the acting group $\mathcal{G}$, another restriction is that the Lagrange interpolation polynomials from equation (3) must-be well-defined, i.e. invertible modulo the group order $q = |\mathcal{G}|$. By numbering the parties $1, \ldots, n$, this restriction simply translates to

[3] An exceptional set modulo q is a set of in which the pairwise difference of all the values in the set is invertible modulo q. Since we will ultimately be restricting ourselves to prime subgroups, every set of distinct values is an exceptional set.

n being smaller than the smallest divisor of q. As [42] points out, this can simply be achieved by working in a subgroup of $\mathcal{G}$, for which this condition is fulfilled.

Finally, although the security of the secret sharing and zero-knowledge algorithms introduced in this paper also hold in the setting of a composite group (cf. in particular [20] and [9] for the case of Shamir and bivariate sharing), we point out that the challenges in the VSS proof system from Figure 5 cannot be sampled from the entire group $\mathbb{Z}_q$, but have to be sampled from a set of size smaller than the smallest divisor of q (an exceptional set modulo q). This restriction is treated in detail in [13, Section 3] and arises from the extraction of the secrets via rewinding. In particular this also implies that proof systems like the one used in our VSS might need to be repeated to achieve particular soundness guarantees.

All the issues outlined in this section could be mitigated by simply working in a prime subgroup of the class group. For the CSI-FiSh parameter set, this would imply halving the security at no efficiency gain, which is rather undesirable. We therefore outline in the next section, how a prime order class group can be realized in the isogeny setting.

5.3 Instantiation with prime order acting groups

We first revisit the concept of orientations of supersingular elliptic curves and oriented group actions, and then we describe how to build a group action to fit our constraints outlined above.

5.3.1 Orientations. We briefly recall orientations of supersingular elliptic curves in this section and point the interested reader to [33, 41, 77] for a more thorough presentation. In this section, let K be a quadratic imaginary field and let $\mathcal{O} = \mathbb{Z}[\alpha]$ be a quadratic order in K with $\mathsf{disc}(\mathcal{O}) < 0$.

Definition 9 ((Primitive) Orientations [41, Def. 1]). *For any elliptic curve E, a* K-orientation *is a ring homomorphism $\iota : K \hookrightarrow End(E) \otimes \mathbb{Q}$. A K-orientation induces a* $\mathcal{O}$-orientation *if $\iota(\mathcal{O}) = End(E) \cap \iota(K)$. In that case, the pair (E, ι) is called a* ι-oriented curve *and E is an* $\mathcal{O}$-orientable curve. *We further call an orientation* primitive, *if $\iota(\alpha)$ is a primitive element of $End(E)$.*

Let now (E, ι) be a primitively oriented elliptic curve and let $\mathfrak{a} \in \mathcal{O}$ be an integral invertible ideal of norm coprime to the field characteristic p. Using $\mathfrak{a}$, we can define an $\mathcal{O}$-oriented isogeny $\varphi_\mathfrak{a} : (E, \iota) \to (E_\mathfrak{a} = E/E[\mathfrak{a}], \iota_\mathfrak{a})$, where

$$E[\mathfrak{a}] = \bigcap_{\alpha \in \mathfrak{a}} \ker \iota(\alpha)\,, \quad \text{and} \quad \iota_\mathfrak{a}(x) = \frac{\varphi_\mathfrak{a} \circ \iota(x) \circ \hat{\varphi}_\mathfrak{a}}{\deg \varphi_\mathfrak{a}}\,.$$

An $\mathcal{O}$-isomorphism is an $\mathcal{O}$-oriented isogeny of degree 1. Using this, we can define $\mathcal{E}_\mathcal{O}$ as the set of $\mathcal{O}$-oriented curves up to $\mathcal{O}$-isomorphism. If p does not split in K and does not divide the conductor of $\mathcal{O}$ in $\mathcal{O}_K$, then $\mathcal{E}_\mathcal{O}$ is non-empty [77]. In this case, the ideal class group $\mathsf{cl}(\mathcal{O})$ acts on the set $\mathcal{E}_\mathcal{O}$ via

$$\mathsf{cl}(\mathcal{O}) \times \mathcal{E}_\mathcal{O} \to \mathcal{E}_\mathcal{O}, \quad ([\mathfrak{a}, (E, \iota)) \mapsto \mathfrak{a} \star (E, \iota) = (E_\mathfrak{a}, \iota_a)\,.$$

As shown by Onuki [77], this group action is free and has (at most two orbits). The restriction to one of these orbits also makes this group action transitive, and therefore regular.

5.3.2 Instantiating group actions with prime order. Recent results on scalable class groups, starting with SCALLOP [41], allow flexible control over the size and structure of the acting group $\mathcal{G}$. The SCALLOP framework [6,31,41] proposes a way to work with different orientations by orders of the type $\mathcal{O} = \mathbb{Z}[f\sqrt{-d}]$, where f is a large prime and d a small positive integer. Starting from a quadratic order $\mathcal{O}_0$ of small discriminant, one can construct any such order simply by the choice of the conductor f. By choosing $\mathcal{O}_0 = \mathbb{Z}[\sqrt{-1}]$ to be the Gaussian integers, we find that $\mathcal{O} = \mathbb{Z} + f\mathcal{O}_0$ has class number $f-1$, thus giving us a lot of control of the group size. One restriction is that the conductor should be prime for security reasons, see [91, Theorem 5].

Although the initial SCALLOP approaches in [31, 41] needed the group size to be smooth for efficiency reasons, this restriction has been lifted in Clapoti [79], which allows to embed isogenies defined via ideals $\mathfrak{a} \in \mathcal{O}$ into higher-dimensional representations. To this end, by finding two equivalent ideals $\mathfrak{b}, \mathfrak{c}$ and solving norm equations of the type

$$un(\mathfrak{b}) + vn(\mathfrak{c}) = 2^e\,,$$

(with potential restrictions on u and v), one can embed $\mathfrak{a}$ into a higher-dimensional 2^e-isogeny using Kani's Lemma. More recently, KLaPoTi [80] and PEGASIS [38] have managed to make this approach truly practical, by embedding such isogenies into 2 and 4 dimensions, respectively, and introducing further optimizations. We forego a detailed exposition of these sources and point the interested reader to [38, 79, 80] for more details.

For the sake of simplicity, we opt for an instantiation with KLaPoTi, as their design choices allow us to directly work on $\mathbb{Z}[f\sqrt{-1}]$-oriented elliptic curves with known class number $f-1$. In contrast, PEGASIS takes a more class group-agnostic approach and in particular focuses on orientations by the Frobenius endomorphism to achieve higher efficiency.[4]

Setup. We outline below, how to instantiate a group action with a prime order group using the approach from SCALLOP.

[4] We are however convinced that PEGASIS can also be realized in SCALLOP-type groups, by pushing through orientations as part of the group action computations. This approach probably needs orientations to be represented in higher dimensions as is done in SCALLOP-HD [31] and might need some extra considerations, which are outside of the scope of this work. We note that the main advantage of PEGASIS over KLaPoTi is that the latter needs a field characteristic $p > |\mathrm{disc}(\mathcal{O})|^3$ due to the output sizes of the KLPT algorithm, while PEGASIS only needs $p > |\mathrm{disc}(\mathcal{O})|$, which results in faster arithmetic and smaller representations of elements in $\mathcal{E}$. We leave such an instantiation as an open question for future work.

1. Fix a conductor f, such that $f - 1 = 2q$,[5] where q is a prime depending on the security parameter λ. Define $\mathcal{O} = \mathbb{Z}[f\sqrt{-1}]$.
2. Fix a prime $p = c2^e - 1 \equiv 3 \pmod 4$ with $2^e \approx |\mathrm{disc}(\mathcal{O})|^3$. Let $\mathcal{E}$ be the set of supersingular $\mathcal{O}$-oriented elliptic curves (up to $\mathcal{O}$-isomorphism) with representation over $\mathbb{F}_{p^2}$. For $\mathcal{E}$ to be non-empty by [77, Proposition 3.2], ensure that p is coprime to f and does not split in $\mathcal{O} = \mathbb{Z}[f\sqrt{-1}]$.
3. Find an $\mathcal{O}$-oriented starting curve $(E_0, \iota_0) \in \mathcal{E}$ by using the curve setup approach described in [41, Section 5.1].
4. Sample an element $\mathfrak{g}' \leftarrow \mathsf{cl}(\mathcal{O})$ and set $\mathfrak{g} = \mathfrak{g}'^2$ (if $\mathfrak{g}$ trivial, repeat). Since the SCALLOP approach always results in cyclic groups [31, Section 2.2], this $\mathfrak{g}$ will generate a subgroup of $\mathsf{cl}(\mathcal{O})$ of prime order q.

In the end, our public parameters are

$$\mathsf{pp} = (f, p, \mathfrak{g}, ([\cdot], \mathbb{Z}_q, \mathcal{E}), (E_0, \iota_0)) \,,$$

where $[\cdot]$ denotes the oriented group action described in Section 5.3.1 in its additive notation with generator $\mathfrak{g}$, i.e.

$$[a](E, \iota) = \mathfrak{g}^a \star (E, \iota) \,.$$

The computation of this group action can be realized using Algorithm 2 in the KLaPoTi paper [80].

Notation. For generality throughout this work, we omit the explicit notation of orientations for elements in $\mathcal{E}$ and simply denote oriented elliptic curves as E and the oriented group action as $[a]E$.

5.3.3 Concrete parameters. To fix concrete parameters for quantum security, we have to take into consideration the attack on the vectorization problem with shifted input from [52]. The authors in [52] only provide concrete numbers of their attack for CSIDH-512 and CSIDH-1024. In the case of higher security, in particular when aiming for NIST level-I, which lies somewhere between CSIDH-2048 (optimistically) and CSIDH-4096 (pessimistically) [25,83], we have to take a more in-depth look at the attack. The T-gate complexity of their approach is composed of three main terms [52, Theorem 7.1]

$$\frac{1}{2}(3k+1)\log q(\log q + 1) + kG_T + L_T(k) \,.$$

The first term encodes the gate count for the Fast Fourier Transform, the second the cost of evaluating a group action G_T and the third one the cost solving the CVP instance. Note that each of the three terms scales with the lattice dimension $k = \lceil \frac{\log q}{\log(2M+1)} \rceil$, where again $q = \#\mathsf{cl}(\mathcal{O})$ and where M is the number

[5] q and f are instances of so-called *Sophie Germain* and *safe* primes, respectively.

of queries to the oracle in Problem 3. In the upper equation, the FFT term is relatively small. For the CVP term, we focus on the sieving attack outlined in their [52, Section 6], as this consistently outperforms the enumeration approach in terms of gate count, although at a higher cost in memory. In the experiments run in [52], this term has a larger contribution than kG_T and is generally the limiting factor for the attack. In particular, limiting the number of OPRF queries M gives us full control over k and hence over the efficiency of the attack.

The implementation provided by [52] allows to compute $L_T(k)$ as a function of the lattice dimension k.[6] To estimate the targeted security, we use the lower range estimate of 2^{74} T-gates provided by NIST [76] for breaking an AES-128 instance. By setting $k = 55$, we find that the sieving algorithm would need approximately the same number of T-gates as breaking AES-128, so this is the lattice dimension we will target.

We fine-tune the number of maximally allowed OPRF queries M_{max} to still achieve NIST-I security for a given class group instance. We provide the query limits M_{max} for different class group sizes q ranging from the optimistic ($\log q = 1024$) to the pessimistic ($\log q = 2048$) case for the standard CSIDH security against the Kuperberg attack in Table 4 below. In addition, we consider larger parameter sets in case one would target larger query limits. We also provide concrete parameters for prime class group sizes q such that $f = 2q + 1$ is also prime.

Table 4: Sophie-Germain-primes q and query limits M_{max} for different targeted class group sizes.

$\log q$	q	M_{max}
1024	$2^{1024} + 419317$	$2^{17.6}$
1536	$2^{1536} + 1738345$	$2^{26.9}$
2048	$2^{2048} + 2253895$	$2^{36.2}$
3072	$2^{3072} + 300163$	$2^{54.8}$
4096	$2^{4096} + 241363$	$2^{73.5}$

The parameters in Table 4 provide a trade-off between efficiency and long-term key usage. For the lowest NIST-I parameter set, the (threshold) OPRF can be queried up to 200.000 times, before refreshing the keys. Although this might be sufficient for many applications, in some scenarios, refreshing the keys might not be a viable option. In these cases, one could simply work with a higher parameter set and hence a slower and larger OPRF. Assuming more conservatively that the NIST-I security of CSIDH lies around $\log q = 2048$, the number of maximal queries allowed already reaches 81 billion. Even here, higher options are available by increasing the parameters.

[6] Source code: `https://github.com/vgilchri/CvD-analysis`

5.4 Cost overview

In this section, we establish the communication and computational complexity of our protocols. With regards to the latter, we will focus on the cost in terms of group actions to compute, as group actions are particularly heavy when compared to e.g. commitment computations or polynomial evaluations. Most of the communication involves elements from $\mathbb{Z}_q$ or elliptic curves. Elements from the former can be represented using $\log q$ bits, while elliptic curves can be either $\mathbb{F}_p$-rational or oriented, depending on whether we instantiate our scheme with CSI-FiSh or KLaPoTi (or PEGASIS). We generalize their cost as $c \log p$ bits, where $c = 1$ for CSI-FiSh and $c = 6$ for the oriented version.

5.4.1 OPRF protocol. We summarize the costs of the base (V)OPRF protocol in Table 5 below and compare our results with the state-of-the-art OPRF from [44].[7] For the latter, we only display the more efficient, linear case ($d = 1$) as this is the most comparable case to ours in terms of security assumptions. We can see that our OPRF costs half as many group actions in both the verifiable and non-verifiable case. This comes at a slightly higher communication cost in the verifiable case, while the non-verifiable cases have the same communication costs. We average over the cases where the challenge is 0 and the server responds with the pair (v_j, w_j^S) or where it is 1, and the server responds with $b_j - \mathsf{sk}$, resulting in 2 or 1 element in $\mathbb{Z}_q$, respectively.

Table 5: Communication and computational cost of our (verifiable) OPRF protocol from Figure 4. The parameter $c \in \{1, 6\}$ depends on whether elliptic curves are oriented or not.

	Communication		Group Action Computations	
	Client → Server	Server → Client	Client-side	Server-side
Figure 4 (verifiable)	$\log q$ $+\lambda \log q$	$\log q + c \log p$ $+\lambda(\frac{5}{2} \log q + 1)$	1 $+2\lambda$	1 $+2\lambda$
[44], $d = 1$ (verifiable)	$\log q$	$\log q + c \log p$ $+2\lambda(\log q + 1)$	2 $+4\lambda$	2 $+4\lambda$

5.4.2 Threshold protocols. We continue with our analysis of both the malicious and semi-honest threshold instantiations from Sections 4 and B, respectively. First, we would like to emphasize that, unlike other post-quantum constructions [5,65,89], the communication between client and server, as well as the computational cost on the client side, are independent of the number of server-side parties and thus remain the same in the threshold case as those in Table 5.

[7] We note that [44] also needs a prime order group action, so we describe it in the same two settings.

In this section, we therefore only analyze the internal cost on the server-side, i.e. the communication and computations involved within the multi-party protocols. We distinguish these along two dimensions, i.e. whether we allow semi-honest or malicious corruptions and whether we want to achieve a verifiable output or not.

Due to our modular structure, we can simply establish these costs in terms of the employed subroutines Samp, Rec and GA. For example, from Figure 4, we can easily see that the total server cost in the non-verifiable case is composed of one evaluation of Rec and one of GA, whereas in the verifiable case, we further need to execute λ Samp, $\frac{3}{2}\lambda$ Rec (on average) and 2λ GA. The costs of these individual routines can be extracted from their threshold instantiations in Sections 4 and B. We summarize their costs in Table 6, where the parameter $m = |T| \in \{t+1, \ldots, n\}$ indicates the number of participating parties. Note that the optimal case is always $m = t + 1$. For simplicity, we outline costs for executions of the protocols without complaint resolution.

Table 6: Communication and computational cost of the subroutines Samp, Rec and GA with their respective instantiation in the semi-honest and malicious case. The constant $c \in \{1, 6\}$ indicates the Frobenius or oriented case. The communication cost is the outgoing cost per party. The computational cost is the *total sequential* cost of the entire routine in terms of group actions.

	Communication		Group Action Computations	
	semi-honest	malicious	semi-honest	malicious
Samp	$(m-1)\log q$	$m(m+1)\lambda+$ $(t+1)(m+t)\log q$		
Rec	$\log q$	$(t+1)\log q$		
GA	$c\log p$	$6(m+1)\lambda+$ $(t+1)\lambda\log q + c\log p$	m	$m\lambda + \min\{m, \lambda+2\}$

The established communication costs are visualized in Figure 7, both for an instantiation with CSI-FiSh and with KLaPoTi in the conservative quantum-secure case. We can see that the communication cost stays quite manageable, even when working with multiple parties. Classically, communication remains mostly in the kilobyte and low megabyte range. Even in the quantum case, in the presence of malicious parties and with verifiable output, the communication cost hits the 1 GB boundary only with more than 64 parties participating in the protocol. We emphasize again that verification of the outputs in a robust threshold setting is, at least from a theoretical point of view redundant, since the definition of robustness already guarantees correct outputs. We point the reader again to the discussion in Section 4.3. Without verifiable output, the

communication costs drops considerably to the low megabyte range, even for 64 parties.

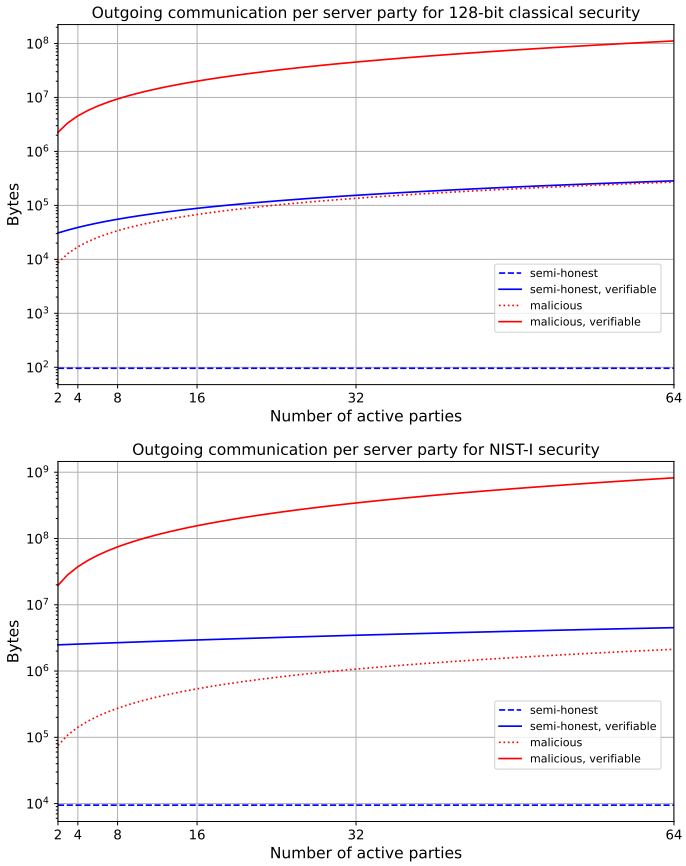

Fig. 7: Outgoing communication per server party for different numbers of parties active in the threshold OPRF execution, both for 128-bit classical security (top) and conservative NIST-I security (bottom). Red indicates the malicious and blue the semi-honest case with (solid) and without (dashed) verifiability of the OPRF output.

To estimate computational costs, we rely on benchmarks for group action computations found throughout the literature. With recent optimizations [19,43] a group action in the 128-bit classical case takes approximately 30 ms. Although λ repetitions within our proof systems makes this sound quite expensive, we note that, as [8] pointed out, proof systems such as SPVPs use completely unrelated commitments at each step, so the group actions are perfectly parallelizable. We

leverage modern architectures, which rarely come without 8 or 16 cores, and plot the total time our threshold OPRF execution on the server side in Figure 8 below. We find that the classical case scales very well, with runtimes in the sub-second and second range in the non-verifiable case. We can see that making the output verifiable in the threshold setting results in much more expensive server-side computations, since the proof computation is itself distributed. Nevertheless, even in the malicious setting, this computation is in the range of minutes and crosses the 1-hour boundary at around 58 parties. However, we emphasize again, that verification in the distributed server case is not strictly necessary, as discussed in Section 4.3.

For the quantum secure case, finding a cost per group action estimate is unfortunately more complicated. In particular, KLaPoTi does not provide an instantiation higher than a class group of a 512-bit discriminant. On the other hand, PEGASIS provides proof-of-concept numbers with a SAGE implementation for a 2000-bit discriminant at about 21 seconds per group action. However, the authors expect to potentially be able to reduce this by two orders of magnitude with an optimized implementation. We note however that PEGASIS runtimes use orientations via the Frobenius endomorphism, while we work with general oriented curves, so we can expect another slowdown due to this modification. As a consequence, we can roughly estimate the quantum secure case to probably be at least a factor 10 slower than the classical case, but we note that future implementations should be able to bring more clarity in this regard.

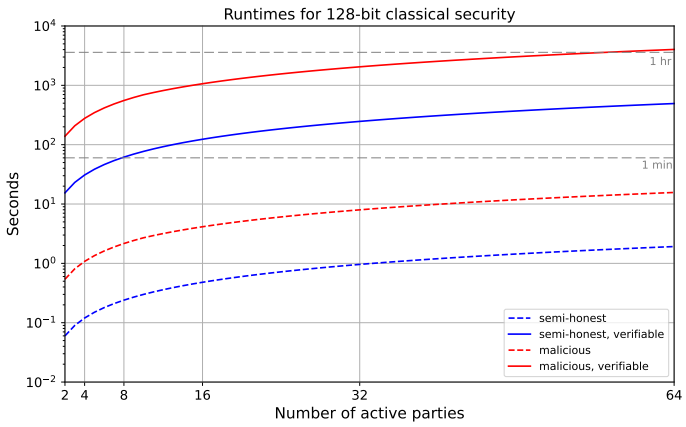

Fig. 8: Estimated total computational cost of the threshold OPRF execution for different numbers of parties active in the protocol for 128-bit classical security case. We assume each party to have access to 16 cores. Red indicates the malicious and blue the semi-honest case with (solid) and without (dashed) verifiability of the OPRF output.

References

1. Agrawal, S., Miao, P., Mohassel, P., Mukherjee, P.: PASTA: PASsword-based threshold authentication. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 2042–2059. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243839
2. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_14
3. Albrecht, M.R., Davidson, A., Deo, A., Gardham, D.: Crypto dark matter on the torus - oblivious PRFs from shallow PRFs and TFHE. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 447–476. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_16
4. Albrecht, M.R., Davidson, A., Deo, A., Smart, N.P.: Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 261–289. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75248-4_10
5. Albrecht, M.R., Gür, K.D.: Verifiable oblivious pseudorandom functions from lattices: Practical-ish and thresholdisable. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part IV. LNCS, vol. 15487, pp. 205–237. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0894-2_7
6. Allombert, B., Biasse, J.F., Eriksen, J.K., Kutas, P., Leonardi, C., Page, A., Scheidler, R., Bagi, M.T.: Faster SCALLOP from non-prime conductor suborders in medium sized quadratic fields. In: Jager, T., Pan, J. (eds.) PKC 2025, Part III. LNCS, vol. 15676, pp. 333–363. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91826-1_11
7. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. Journal of Cryptology **30**(1), 58–151 (Jan 2017). https://doi.org/10.1007/s00145-015-9214-4
8. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: CSI-SharK: CSI-FiSh with sharing-friendly keys. In: Simpson, L., Baee, M.A.R. (eds.) ACISP 23. LNCS, vol. 13915, pp. 471–502. Springer, Cham (Jul 2023). https://doi.org/10.1007/978-3-031-35486-1_21
9. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: Practical robust DKG protocols for CSIDH. In: Tibouchi, M., Wang, X. (eds.) ACNS 2023, Part II. LNCS, vol. 13906, pp. 219–247. Springer, Cham (Jun 2023). https://doi.org/10.1007/978-3-031-33491-7_9
10. Atapoor, S., Baghery, K., Cozzo, D., Pedersen, R.: VSS from distributed ZK proofs and applications. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part I. LNCS, vol. 14438, pp. 405–440. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8721-4_13
11. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 590–609. Springer, Berlin, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_32
12. Baghery, K.: Π: A unified framework for computational verifiable secret sharing. In: Jager, T., Pan, J. (eds.) PKC 2025, Part IV. LNCS, vol. 15677, pp. 110–142. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91829-2_4

13. Baghery, K., Cozzo, D., Pedersen, R.: An isogeny-based ID protocol using structured public keys. In: Paterson, M.B. (ed.) 18th IMA International Conference on Cryptography and Coding. LNCS, vol. 13129, pp. 179–197. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92641-0_9
14. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 433–444. ACM Press (Oct 2011). https://doi.org/10.1145/2046707.2046758
15. Basso, A.: A post-quantum round-optimal oblivious PRF from isogenies. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) SAC 2023. LNCS, vol. 14201, pp. 147–168. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-53368-6_8
16. Basso, A., Kutas, P., Merz, S.P., Petit, C., Sanso, A.: Cryptanalysis of an oblivious PRF from supersingular isogenies. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 160–184. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92062-3_6
17. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer, Berlin, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_34
18. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988). https://doi.org/10.1145/62212.62213
19. Bernstein, D.J., de Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. In: ANTS-XIV-14th Algorithmic Number Theory Symposium. vol. 4, pp. 39–55. Mathematical Sciences Publishers (2020)
20. Beullens, W., Disson, L., Pedersen, R., Vercauteren, F.: CSI-RAShi: Distributed key generation for CSIDH. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 257–276. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_14
21. Beullens, W., Dodgson, L., Faller, S.H., Hesse, J.: The 2Hash OPRF framework and efficient post-quantum instantiations. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part VIII. LNCS, vol. 15608, pp. 332–362. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91101-9_12
22. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
23. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 67–97. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_3
24. Boneh, D., Kogan, D., Woo, K.: Oblivious pseudorandom functions from isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 520–550. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_18
25. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 493–522. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_17
26. Campos, F., Muth, P.: On actively secure fine-grained access structures from isogeny assumptions. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryp-

tography - 13th International Workshop, PQCrypto 2022. pp. 375–398. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-17234-2_18
27. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
28. Casacuberta, S., Hesse, J., Lehmann, A.: SoK: Oblivious pseudorandom functions. In: 2022 IEEE European Symposium on Security and Privacy. pp. 625–646. IEEE Computer Society Press (Jun 2022). https://doi.org/10.1109/EuroSP53844.2022.00045
29. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 423–447. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_15
30. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci. **777**, 155–183 (2019). https://doi.org/10.1016/J.TCS.2019.02.001
31. Chen, M., Leroux, A., Panny, L.: SCALLOP-HD: Group action from 2-dimensional isogenies. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part II. LNCS, vol. 14603, pp. 190–216. Springer, Cham (Apr 2024). https://doi.org/10.1007/978-3-031-57725-3_7
32. Choi, K., Manoj, A., Bonneau, J.: SoK: Distributed randomness beacons. In: 2023 IEEE Symposium on Security and Privacy. pp. 75–92. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179419
33. Colò, L., Kohel, D.: Orienting supersingular isogeny graphs. Cryptology ePrint Archive, Report 2020/985 (2020), https://eprint.iacr.org/2020/985
34. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), https://eprint.iacr.org/2006/291
35. Cozzo, D., Giunta, E.: Round-robin is optimal: Lower bounds for group action based protocols. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023, Part IV. LNCS, vol. 14372, pp. 310–335. Springer, Cham (Nov / Dec 2023). https://doi.org/10.1007/978-3-031-48624-1_12
36. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Berlin, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_22
37. Damgård, I.: On the randomness of Legendre and Jacobi sequences. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 163–172. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34799-2_13
38. Dartois, P., Eriksen, J.K., Fouotsa, T.B., Merdy, A.H.L., Invernizzi, R., Robert, D., Rueger, R., Vercauteren, F., Wesolowski, B.: PEGASIS: Practical effective class group action using 4-dimensional isogenies. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025, Part I. Lecture Notes in Computer Science, vol. 16000, pp. 67–99. Springer, Cham, Switzerland, Santa Barbara, CA, USA (August 17–21, 2025). https://doi.org/10.1007/978-3-032-01855-7_3
39. Das, P., Hesse, J., Lehmann, A.: DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIACCS 22. pp. 682–696. ACM Press (May / Jun 2022). https://doi.org/10.1145/3488932.3517389
40. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 66–98. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_3

41. De Feo, L., Fouotsa, T.B., Kutas, P., Leroux, A., Merz, S.P., Panny, L., Wesolowski, B.: SCALLOP: Scaling the CSI-FiSh. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 345–375. Springer, Cham (May 2023). https://doi.org/10.1007/978-3-031-31368-4_13
42. De Feo, L., Meyer, M.: Threshold schemes from isogeny assumptions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 187–212. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45388-6_7
43. Decru, T.: Radical $\sqrt[N]{\text{élu}}$ isogeny formulae. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part V. LNCS, vol. 14924, pp. 107–128. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68388-6_5
44. Delpech de Saint Guilhem, C., Pedersen, R.: New proof systems and an OPRF from CSIDH. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part II. LNCS, vol. 14603, pp. 217–251. Springer, Cham (Apr 2024). https://doi.org/10.1007/978-3-031-57725-3_8
45. Esgin, M.F., Steinfeld, R., Tairi, E., Xu, J.: LeOPaRd: Towards practical post-quantum oblivious PRFs via interactive lattice problems. Cryptology ePrint Archive, Report 2024/1615 (2024), https://eprint.iacr.org/2024/1615
46. Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 547–562. USENIX Association (Aug 2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/everspaugh
47. Faller, S.H., Ottenhues, A., Ottenhues, J.: Composable oblivious pseudo-random functions via garbled circuits. In: Aly, A., Tibouchi, M. (eds.) LATINCRYPT 2023. LNCS, vol. 14168, pp. 249–270. Springer, Cham (Oct 2023). https://doi.org/10.1007/978-3-031-44469-2_13
48. Faust, S., Hazay, C., Venturi, D.: Outsourced pattern matching. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 545–556. Springer, Berlin, Heidelberg (Jul 2013). https://doi.org/10.1007/978-3-642-39212-2_48
49. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th FOCS. pp. 427–437. IEEE Computer Society Press (Oct 1987). https://doi.org/10.1109/SFCS.1987.4
50. Fouotsa, T.B., Moriya, T., Petit, C.: M-SIDH and MD-SIDH: Countering SIDH attacks by masking information. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 282–309. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_10
51. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Berlin, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30576-7_17
52. Frixons, P., Gilchrist, V., Kutas, P., Merz, S.P., Petit, C.: Another look at the quantum security of the vectorization problem with shifted inputs. Cryptology ePrint Archive, Report 2025/376 (2025), https://eprint.iacr.org/2025/376
53. Galbraith, S., Panny, L., Smith, B., Vercauteren, F.: Quantum equivalence of the DLP and CDHP for group actions. Cryptology ePrint Archive, Report 2018/1199 (2018), https://eprint.iacr.org/2018/1199
54. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: Provably preventing DNSSEC zone enumeration. In: NDSS 2015. The Internet Society (Feb 2015). https://doi.org/10.14722/ndss.2015.23211

55. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986). https://doi.org/10.1145/6490.6503
56. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_10
57. Heimberger, L., Hennerbichler, T., Meisingseth, F., Ramacher, S., Rechberger, C.: OPRFs from isogenies: Designs and analysis. In: Zhou, J., Quek, T.Q.S., Gao, D., Cárdenas, A.A. (eds.) ASIACCS 24. ACM Press (Jul 2024). https://doi.org/10.1145/3634737.3645010
58. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_9
59. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_13
60. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 2017. LNCS, vol. 10355, pp. 39–58. Springer, Cham (Jul 2017). https://doi.org/10.1007/978-3-319-61204-1_3
61. Jarecki, S., Krawczyk, H., Resch, J.: Threshold partially-oblivious PRFs with applications to key management. Cryptology ePrint Archive, Report 2018/733 (2018), https://eprint.iacr.org/2018/733
62. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15
63. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Berlin, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_34
64. Jarecki, S., Shmatikov, V.: Handcuffing big brother: an abuse-resilient transaction escrow scheme. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 590–608. Springer, Berlin, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_35
65. Kaluderovic, N., Cheng, N., Mitrokotsa, A.: A post-quantum distributed OPRF from the legendre PRF. In: Nicomette, V., Benzekri, A., Boulahia-Cuppens, N., Vaidya, J. (eds.) ESORICS 2025: 30th European Symposium on Research in Computer Security, Part II. Lecture Notes in Computer Science, vol. 16054, pp. 205–225. Springer, Cham, Switzerland, Toulouse, France (September 22–24, 2025). https://doi.org/10.1007/978-3-032-07891-9_11
66. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 724–741. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_35
67. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978357

68. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 818–829. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978381
69. Lai, Y.F.: A note on isogeny group action-based pseudorandom functions. Cryptology ePrint Archive, Report 2024/2042 (2024), https://eprint.iacr.org/2024/2042
70. Lai, Y.F., Galbraith, S.D., Delpech de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 213–241. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_8
71. Levin, S., Pedersen, R.: Faster proofs and VRFs from isogenies. In: Hanaoka, G., Yang, B.Y. (eds.) Advances in Cryptology – ASIACRYPT 2025, Part IV. Lecture Notes in Computer Science, vol. 16248, pp. 307–340. Springer, Singapore, Singapore, Melbourne, VIC, Australia (December 8–12, 2025). https://doi.org/10.1007/978-981-95-5113-2_10
72. Liu, F.H., Zhang, E., Qin, L.: Efficient multiparty probabilistic threshold private set intersection. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 2188–2201. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3623158
73. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 448–471. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_16
74. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS. pp. 120–130. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814584
75. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: 38th FOCS. pp. 458–467. IEEE Computer Society Press (Oct 1997). https://doi.org/10.1109/SFCS.1997.646134
76. National Institute of Standards and Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Technical report, NIST Computer Security Resource Center (Dec 2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
77. Onuki, H.: On oriented supersingular elliptic curves. Finite Fields and Their Applications **69**, 101777 (2021)
78. Orsini, E., Zanotto, R.: Simple two-message OT in the explicit isogeny model. CiC **1**(1), 15 (2024). https://doi.org/10.62056/a39qgy4e-
79. Page, A., Robert, D.: Introducing clapoti(s): Evaluating the isogeny class group action in polynomial time. Cryptology ePrint Archive, Report 2023/1766 (2023), https://eprint.iacr.org/2023/1766
80. Panny, L., Petit, C., Stopar, M.: KLaPoTi: An asymptotically efficient isogeny group action from 2-dimensional isogenies. IACR Communications in Cryptology (CiC) **2**(3), 5 (2025). https://doi.org/10.62056/ahp2wakrz
81. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract) (rump session). In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 522–526. Springer, Berlin, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_47

82. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Berlin, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
83. Peikert, C.: He gives C-sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 463–492. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_16
84. Rabin, M.O.: Transaction protection by beacons. J. Comput. Syst. Sci. **27**(2), 256–267 (1983). https://doi.org/10.1016/0022-0000(83)90042-9, https://doi.org/10.1016/0022-0000(83)90042-9
85. Robert, D.: Breaking SIDH in polynomial time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 472–503. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_17
86. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006), https://eprint.iacr.org/2006/145
87. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176
88. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700
89. Singh, A., Patranabis, S., Mukhopadhyay, D.: Threshold OPRF from threshold additive HE. Cryptology ePrint Archive, Report 2024/1032 (2024), https://eprint.iacr.org/2024/1032
90. Unruh, D.: Collapse-binding quantum commitments without random oracles. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 166–195. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53890-6_6
91. Wesolowski, B.: Orientations and the supersingular endomorphism ring problem. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 345–371. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_13
92. Yang, Y., Benhamouda, F., Halevi, S., Krawczyk, H., Rabin, T.: Gold OPRF: Post-quantum oblivious power-residue PRF. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) 2025 IEEE Symposium on Security and Privacy. pp. 259–278. IEEE Computer Society Press, San Francisco, CA, USA (May 12–15, 2025). https://doi.org/10.1109/SP61157.2025.00116

A Preliminaries on distributed protocols from isogenies

A.1 The threshold zero-knowledge proof from Π

In this section, we revisit the proof system from Π [12], in which a dealer convinces parties that they have received consistent Shamir shares, i.e. that they hold evaluations of a polynomial in $\mathbb{Z}_q[X]$ of the correct degree t. Underlying this zero-knowledge proof is the n-distributed relation $(R_1, \ldots, R_n)$, with

$$R_i = \{(s_i, f(X)) \mid f(X) \in \mathbb{Z}_q[X]_t \wedge s_i = f(i)\}\,.$$

Threshold zero-knowledge (TZK) proofs over n-distributed relations are formally introduced in [23] and can be seen as zero-knowledge proofs in which the n verifying parties each hold a piece (or share) of the full statement $(s_1, \ldots, s_n)$. Next to completeness and soundness with respect to honest verifiers, TZK proofs satisfy the *threshold zero-knowledge* property, which implies that even up to t corrupted verifiers do not learn anything about the witness *or the statement*. We point the reader to [23] for an in-depth formalization.

We revisit the proof system underlying Π in its commitment scheme version Π_{LA} in Figure 9. Π satisfies the conditions of a threshold zero-knowledge proof for relations $(R_1, \ldots, R_n)$. Note that Π makes use of a collapsing and quantum computationally hiding commitment scheme $\mathsf{C} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\lambda}$.

Π.Prove$(f(X))$:

1. Sample $r(X) \leftarrow \mathbb{Z}_q[X]_t$ and compute the commitments $C_i = \mathsf{C}(f(i), r(i))$ for $i = 1, \ldots, n$.
2. Run $c \leftarrow \mathcal{H}(C_1, \ldots, C_n)$.
3. Compute the response $z(X) = r(X) + cf(X)$.
4. Output $(C_1, \ldots, C_n, z(X))$

Π.Verify$(\{C_i, s_i\}_{i=1,\ldots,n}, z(X))$:

1. Each party P_i computes $c \leftarrow \mathcal{H}(C_1, \ldots, C_n)$ and then verifies whether
   (a) $z(X) \in \mathbb{Z}_q[X]_t$, and
   (b) using their share s_i, whether $C_i = \mathsf{C}(s_i, z(i) - cs_i)$
2. If any verification fails, parties issue a complaint against the prover. If there are more than t complaints, output `false`. Any other complaints by a party P_i are resolved by the prover publishing s_i, which is verified by all the parties. If this verification fails, output `false`.
3. Output `true`.

Fig. 9: The proof system underlying Π_{LA}.

Remark 1. We note that Π presents another version, where the commitments are generated using extra randomizers, e.g. $C_i = \mathsf{C}(f(i), r(i), \gamma_i)$ for $\gamma_i \leftarrow \{0,1\}^{\lambda}$.

These are necessary, so that the commitments still securely hide low-entropy shares $f(i)$. Throughout this work, all shares are sampled uniformly at random, thus have high enough entropy that they don't need these extra randomizers.

A.2 Structured Piecewise Verifiable Proofs

We revisit a modified version of the piecewise verifiable proofs (PVP) from [20], called *structured* PVPs [9]. PVPs were initially introduced to convince parties that a group action $[s]E$ has been performed correctly using a secret s of which the verifiers hold Shamir shares $\langle s \rangle_i$. As such, it is an example of a zero-knowledge proof on secret shared data as introduced in [23]. The structured variant introduced in [9] allows the prover to show that it has acted with different, public multiples of this secret s. We do not need the full proof system for a variable amount of elliptic curve pairs, but actually only for a single elliptic curve pair. The resulting language is the following,

$$\mathcal{L}^T_{SPVP} = \{(E_0, E_1, a) \mid \exists f(X) : E_1 = [af(0)]E_0 \wedge \{x_i = f(i)\}_{i \in T}\},$$

where the x_i are shares held by the individual parties in T, which are the distributed verifiers of the proof system. The proof system in Figure 10 makes use of a random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and a collapsing and quantum computationally hiding commitment function $\mathcal{C} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{2\lambda}$. We refer to [9, Theorem 4.1] for a security proof in the QROM.

$\mathsf{SPVP.Prove}(a, f(X), x_0 = (E_0, E_1), \{x_i\}_{i \in T})$:

1. For $j = 1, \ldots, \lambda$, sample $b_j \leftarrow \mathbb{Z}_q[X]_t$ and compute $\hat{E}_j \leftarrow [ab_j(0)]E_0$.
2. Sample $y_0, y_0' \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{C}_0 \leftarrow \mathcal{C}(\underline{\hat{E}}, y_0)$, $\mathsf{C}_0' \leftarrow \mathcal{C}(E_0, E_1, y_0')$.
3. For $i \in T$, sample $y_i, y_i' \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{C}_i \leftarrow \mathcal{C}(\underline{b(i)}, y_i)$ and $\mathsf{C}_i' \leftarrow \mathcal{C}(x_i, y_i')$.
4. Compute $\underline{c} \leftarrow \mathcal{H}(\mathsf{C}, \mathsf{C}')$, where $\mathsf{C} = (\mathsf{C}_0, \ldots, \mathsf{C}_n)$ and $\mathsf{C}' = (\mathsf{C}_0', \ldots, \mathsf{C}_n')$.
5. For $j = 1, \ldots, \lambda$, compute $r_j(X) = b_j(X) - c_j f(X)$.
6. Return $\tilde{\pi} = (\mathsf{C}, \mathsf{C}', \underline{r}, (y_0, y_0'))$ and $\{\pi_i = (y_i, y_i')\}_{i \in T}$.

$\mathsf{SPVP.Verify}(a, x_0 = (E_0, E_1), x_i, \tilde{\pi}, \pi_i)$:

1. Verify that $\mathsf{C}_0' = \mathcal{C}(E_0, E_1, y_0')$ and $\mathsf{C}_i' = \mathcal{C}(x_i, y_i')$, otherwise return $\perp$.
2. Compute $\underline{c} \leftarrow \mathcal{H}(\mathsf{C}, \mathsf{C}')$.
3. For $j = 1, \ldots, \lambda$, let $\widetilde{E}_j \leftarrow [r_j(0)]E_{c_j}$.
4. If $\mathsf{C}_0 = \mathcal{C}(\underline{\widetilde{E}}, y_0)$ and $\mathsf{C}_i = \mathcal{C}(\underline{r(i)} + \underline{c}x_i, y_i)$, return `true`, otherwise $\perp$.

Fig. 10: Structured piecewise verifiable proofs

B Semi-honest threshold OPRF

In this section, we present threshold versions of the subroutines from Figure 3 in the setting of a semi-honest adversary. We achieve this using Shamir's secret sharing in a (t, n)-threshold with a number of corruptions of $t < \frac{n}{2}$ parties and assume t and n to be part of the public parameters throughout this section.

In Figure 11, we display the algorithms for a distributed Shamir secret sharing, reconstructions of linear combinations of secrets and distributed group action computations using linear combinations of secrets. We note that our protocols are rather straightforward modifications of protocols already existing throughout the literature.

$\mathsf{S.Share}(\mathsf{pp}, T_S)$**:**

1. Assert that $\#T_S > t$.
2. Each party P_i for $i \in T_S$ proceeds as follows.
   (a) Sample $f^{(i)}(X) \leftarrow \mathbb{Z}_q[X]_t$,
   (b) Send $f^{(i)}(j)$ to each other party P_j.
   (c) Compute the share $f(i) = \sum_{j \in T_S} f^{(j)}(i)$ of $s = f(0)$.
3. The protocol returns $\langle s \rangle_{T_S}$.

$\mathsf{S.Reconstruct}(\mathbf{a}, \langle \mathbf{s} \rangle_{T_R})$**:**

1. Assert that $\left|T_R \cap \bigcap_i T_{S_i}\right| > t$, where s_i has been generated by a subset T_{S_i}.
2. Each party P_i computes their share of the output as $\langle s \rangle_i = \mathbf{a} \cdot \langle \mathbf{s} \rangle_i$, then broadcasts $\langle s \rangle_i$.
3. Compute and output the secret $s = f(0)$ via Lagrange interpolation from $\langle s \rangle_{T_R}$.

$\mathsf{S.GA}(\mathsf{pp}, (\mathbf{a}, \langle \mathbf{s} \rangle_{T_R}), E)$**:**

1. Assert that $\left|T_R \cap \bigcap_i T_{S_i}\right| > t$, where s_i has been generated by a subset T_{S_i}.
2. Each party P_i computes their share of the shared value as $\langle s \rangle_i = \mathbf{a} \cdot \langle \mathbf{s} \rangle_{T_R}$
3. In a round-robin among the members of T_R, each party P_i computes and returns $E \leftarrow [L_i^{T_R} \langle s \rangle_i] E$.
4. Output E.

Fig. 11: Semi-honest Shamir secret sharing protocol S.

The $\mathsf{S.Share}$ algorithm is a distributed Shamir secret sharing, similar to the verifiable secret sharing introduced by Pedersen [82], but without the need of verification, since semi-honest parties correctly execute the protocol. The $\mathsf{S.GA}$ algorithm extends the distributed group action protocol from [42, Algorithm 1] (which we also briefly revisit in Section 2.3) to the setting of linear combinations of secrets. The correctness and secrecy properties from these sources readily extend to our modifications: the adversary learns at most t shares and can therefore not infer any information about the secret, and the output of the protocols are correct, since parties do not deviate from the protocol specifications. Note that

the assert condition verifies whether the parties reconstructing the linear combination of shares have enough information to do so. Finally, the $\mathsf{S.Reconstruct}$ algorithm is an extension of standard reconstruction of a secret, where parties simply reveal their shares, allowing each party to interpolate to the secret. Again, correctness is straightforward in the semi-honest setting.

Tuple generation. Before we can replace the sampling and linear combination steps of our VOPRF protocol, we also have to redefine the workings of the tuple generation functionality. We want to output the same types of tuples as in Definition 6, but instead of simply handing the elements to every party, the functionality sends Shamir shares of these elements to the individual parties. We display this functionality in Definition 10. Note that $\mathcal{F}_{\mathsf{S.TG}}$ takes as input an index set of server parties T_S.

Definition 10 (Shamir tuple generation functionality $\mathcal{F}_{\mathsf{S.TG}}$).

1. On the first activation, learn the list of corrupted parties from the adversary.
2. On input $(\textsc{Tuple}, \mathsf{id}, C, S, T_S)$ by a party P, proceed as follows.
   - If there is no recorded entry for id, sample polynomials $u, v, w^S, w^C \leftarrow \mathbb{Z}_q$ (using pp) with $w^C + w^S = uv$, then generate Shamir sharings $\langle v \rangle_{T_S}, \langle w^S \rangle_{T_S}$ and record $(C, S, \mathsf{id}, u, w^C, \langle v \rangle_{T_S}, \langle w^S \rangle_{T_S})$.
   - Otherwise, retrieve $(C, S, \mathsf{id}, u, w^C, \langle v \rangle_{T_S}, \langle w^S \rangle_{T_S})$.
3. Then,
   - If P is the client, send (u, w^C) to P and send $(\textsc{Output}, C)$ to $\mathcal{A}$.
   - If P is the server party with index $i \in T_S$, send $(\langle v \rangle_i, \langle w^S \rangle_i)$ to P and send $(\textsc{Output}, S, i)$ to $\mathcal{A}$.

Security. We omit a full security proof, as this directly follows from the security proof of the malicious design. Note that semi-honest threshold schemes are by definition correct and robust. Indistinguishability follows using the same arguments, i.e. that an adversary holding up to t shares information-theoretically hides the secret value, which allows the simulator to adapt the reconstructed secret a posteriori, as in the proof in the malicious setting in Section 4.2.