High-throughput Verifiable Distributed OPRF from Gold PRF*

Full Version

Nan Cheng
University of St. Gallen
St. Gallen, Switzerland
nan.cheng@unisg.ch

Yohei Watanabe
The University of Electro-Communications
Tokyo, Japan
AIST
Tokyo, Japan
watanabe@uec.ac.jp

Yugo Kasashima
The University of Electro-Communications
Tokyo, Japan
kasashima@uec.ac.jp

Ioannis Katis
University of St. Gallen
St. Gallen, Switzerland
ioannis.katis@unisg.ch

Aikaterini Mitrokotsa
University of St. Gallen
St. Gallen, Switzerland
katerina.mitrokotsa@unisg.ch

Abstract

An oblivious pseudorandom function (OPRF) is a two-party protocol that enables a client to obtain $F_k(x)$ on an input x without learning the server-held key k, while the server learns nothing about x. OPRF is a fundamental building block in a wide range of privacy-preserving applications, including password-authenticated key exchange (PAKE), private set intersection (PSI), and distributed function secret sharing. In this work, we present the first concrete, high-throughput, post-quantum secure, verifiable distributed OPRF (dOPRF) tolerating $t < n/2$ malicious servers *and a malicious client* over replicated secret sharing, extending the recently introduced Gold OPRF of Yang et al. (IEEE S&P 2025) to a threshold setting.

We adopt an offline–online paradigm and introduce new protocol designs for both phases. *(i)* In the offline phase, we develop efficient protocols for batched generation of replicated secret sharing of α^e whose cost is independent of e. We present two complementary approaches, each designed for different parameter regimes: the first leverages a degenerate additive encoding under which exponentiation commutes with secret sharing; the second employs $t+1$ designated dealers that prove dual-share consistency via non-interactive zero-knowledge proofs, instantiated using both VOLE-in-the-Head and Ligero, yielding post-quantum security based solely on collision-resistant hashing. *(ii)* In the online phase, we propose a constant-round protocol that tightly integrates secure multiplication-and-opening with distributed zero-knowledge proof (DZKP) verification, reducing communication in every setting we evaluate and computation in all but the largest batched setting, compared to naively using the standard DZKP framework. *(iii)* We give a verifiable input protocol that enforces a possibly-malicious client providing well-formed inputs; its well-formedness check is absorbed into a hash exchange the online protocol already performs, and it lowers the client's upload from a full replicated share to one field element per server per input.

Our end-to-end benchmarks show that the construction substantially outperforms the state-of-the-art Legendre-PRF dOPRF of Kaluđerović et al. (ESORICS 2025) in terms of communication across all evaluated settings, and remains practical up to $(n, t) = (9, 4)$, a regime where prior approaches become bandwidth- or memory-prohibitive.

*This is the full version of the paper published in the Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), https://doi.org/10.1145/3830454.3846645.

1 Introduction

Oblivious pseudorandom functions (OPRFs) [21] let a client compute $F_k(x)$ without revealing x to the key-holding server, and without learning k beyond what the output implies. They are a core building block of many privacy-preserving protocols.

OPRFs enable clients to transform low-entropy secrets, such as passwords, identifiers, or personal attributes, into high-entropy cryptographic material without revealing the underlying secret. This property has led to numerous applications, including password-authenticated key exchange (PAKE), password-protected secret sharing, private set intersection (PSI), single sign-on systems, and function secret sharing. OPRFs also serve as a mechanism for delegating computation to servers, allowing clients to remain lightweight while avoiding the storage of sensitive cryptographic state.

Distributed & Verifiable OPRFs. Relying on a single OPRF server represents a single point of trust and failure. This has motivated the study of *distributed OPRFs* (dOPRFs), in which the PRF key is secret-shared among multiple servers that jointly assist the client in evaluating $F_k(x)$. As long as a threshold of servers remains honest, the privacy of both the client input and the server-held key is preserved. Beyond secrecy, modern applications also require *verifiability*, ensuring that even malicious servers cannot deviate from the protocol and cause the client to accept an incorrect output. Thus, verifiable dOPRFs are a key building block for adversarial environments and large-scale deployments. Constructing verifiable dOPRFs that tolerate malicious adversaries while maintaining low communication and round complexity remains a significant challenge.

Post-Quantum Security & the Gold OPRF. Nearly all deployed OPRF constructions rely on Diffie–Hellman-type assumptions [26]. Although these schemes are efficient, they are vulnerable to quantum adversaries due to Shor's algorithm. Thus, there is a growing need for OPRF constructions that offer post-quantum (PQ) security guarantees [1–3, 6, 8, 10, 17, 25, 32]. Although recent years have

seen a surge of interest in PQ OPRFs, existing constructions either incur prohibitive computational and communication overheads, rely on semi-honest security assumptions, or do not extend naturally to distributed and verifiable settings. Closing the efficiency gap between classical and PQ-secure OPRFs remains an open problem.

A notable development is the Gold OPRF [34], based on the power-residue (Gold) pseudorandom function. Unlike classical PRFs, Gold naturally supports multi-bit outputs in a single invocation and admits efficient constant-round two-party evaluation based on vector oblivious linear evaluation (VOLE) correlations.

Prior Works & Their Limitations. Our construction follows the paradigm of [18, 24] that evaluates $(x+k)^g$ obliviously: it lets the client C learn $\alpha^e \cdot (x+k)$ for a uniformly random $\alpha \in \mathbb{F}_p$ that is *unknown* to C; then C computes $\left(\alpha^e \cdot (x+k)\right)^g$. This works because (1) $\alpha^{e \cdot g} \equiv 1 \pmod{p}$ for every $\alpha \in \mathbb{F}_p^*$, ensuring correctness unless $\alpha = 0$—an event of probability $1/p$, accounted for in the statistical error of Theorem 6.4—and (2) $\alpha^e \cdot (x+k)$ is uniformly distributed over the roots of $X^g = (x+k)^g$, which a simulator can reproduce by sampling such a root.

In the MPC context, the paradigm splits power-residue PRF evaluation into an *offline/online* decomposition (Fig. 1): servers precompute $[\![\alpha^e]\!]$ with no client input in the offline phase. In the online phase, given $[\![x]\!]$ from the client, they perform one secure multiplication $[\![\alpha^e]\!] \cdot [\![x+k]\!]$ and open $[\![\alpha^e(x+k)]\!]$ to the client, who applies $(\cdot)^g$ locally. Kaluđerović et al. [28] followed this paradigm, instantiating the technique for the Legendre PRF ($e = 2$) in the distributed setting and leveraging duplicated replicated secret sharing to run the online phase in only two rounds. However, their protocol tolerates only a *dishonest-minority* threshold $t < n/3$, outputs a single bit per invocation, scales unfavorably in total communication with (t, n), and does not support Gold PRF ($e > 2$).

Yang et al. [34] target Gold PRF directly and achieve strong concrete efficiency in the client-server setting. However, their construction is a *two-party* OPRF that presupposes a VOLE correlation produced by an auxiliary setup, which is hard to obtain in practical deployments and does not generalize to multiple servers. Furthermore, their offline phase prepares α^e by *direct sequential squaring*: starting from a VOLE-committed α, they chain $\log_2 e = \lambda$ squarings and attach a batched line-point zero-knowledge proof to certify each step, incurring an $\tilde{O}(\lambda)$ per-evaluation overhead on top of the VOLE correlations, a cost that scales with the exponent e; moreover, each squaring would additionally require a 2PC secure multiplication. When λ is a relatively large integer, the *direct sequential squaring* could be a real efficiency bottleneck.

Challenges & Our Contribution. Existing constructions of distributed verifiable OPRFs fall into two camps, both unsatisfactory for deployment. Classical Diffie–Hellman-based dVOPRFs are concretely efficient but break under quantum adversaries; recent post-quantum candidates close the security gap but pay a heavy concrete cost because their PRF circuits demand many secret-shared multiplications, each of which costs a round of communication and a distributed zero-knowledge verification step. The compounded effect is that no published PQ dVOPRF runs at competitive throughput in WAN deployments under malicious-server adversaries.

We close this gap by exploiting an algebraic property of the recent Gold PRF [34]: its evaluation circuit C_{PRF} consists of a single addition and a single secret-shared multiplication, an unusually low multiplicative complexity that makes Gold uniquely well-matched to honest-majority MPC. Building on this property, we design an *outsourced MPC* scheme (Fig. 1) in which a committee of servers jointly evaluates Gold on behalf of the client, with the verifiability proof amortized into the same round as the multiplication itself.

Using this insight, we present the first *concrete, high-throughput, PQ-secure, verifiable distributed OPRF* tolerating $t < n/2$ malicious servers *and a malicious client.* Our construction builds on the Gold OPRF and follows an offline-online paradigm with new protocol designs in both phases.

In the offline phase, we develop efficient protocols for batched generation of secret-shared exponentiations $[\![\alpha^e]\!]$, with costs independent of the security parameter. We present two approaches: (i) a degenerate additive encoding under which exponentiation commutes with secret sharing; (ii) a construction based on t+1 designated dealers that prove dual-share consistency via non-interactive zero-knowledge proofs, using VOLE-in-the-Head [7] and Ligero [5]. *In the online phase*, we introduce a constant-round protocol that integrates secure multiplication with distributed zero-knowledge proof verification, substantially reducing communication compared to standard DZKP frameworks. We also propose an optimization that significantly decreases the communication sent to the client, further improving practicality in latency-sensitive settings.

On the client's input, we observe that naively handing a secret-shared input to a committee—the client generates the shares and distributes them—is not verifiable: nothing stops a malicious client from sending mismatched copies of a replicated share, leaving different servers evaluating on different points. We close this with Π_{Input}, a mask-and-broadcast input protocol whose consistency check the online protocol absorbs into a hash exchange it already performs, so it costs no extra round there. Because the client uploads one field element per server per input instead of a full replicated share, Π_{Input} is also *cheaper* than the direct sharing it replaces at every threshold we evaluate beyond $(3, 1)$, where it breaks even.

We implement all protocols in `Rust` and benchmark them on a simulated 100 ms / 100 Mbps WAN. End-to-end, our protocol suite consistently outperforms the naive PQ baseline $\Pi^{\mathsf{Aly}}_{\mathsf{DZKP}}$: at $(n, t) = (3, 1)$ we run ≈1.7× faster on WAN time and reduce total communication by ≈1.8×, and at $(n, t) = (5, 2)$ we run ≈1.6× faster. Compared to the closest existing post-quantum dOPRF $\Pi_{\text{L-dOPRF}}$ [28], at matched threshold t=2 we outperform it by ≈44× on WAN time and ≈1,600× on communication, widening to ≈460× and ≈2,200× at m=100. Against the naive baseline the advantage likewise grows with threshold and batch size: at $(n, t) = (7, 3)$, m=100, our protocol $\Pi^{\mathsf{DegGen}}_{\mathsf{dVOPRF}}$ runs ≈3.7× faster than $\Pi^{\mathsf{Aly}}_{\mathsf{DZKP}}$ while reducing total communication by ≈8.0×.

2 Technical Overview

We present the first dVOPRF for Gold PRF that is simultaneously *multi-server, honest-majority ($t < n/2$), maliciously-secure*, and practical. Like prior constructions, our protocol follows the offline/online pipeline of Fig. 1: the offline phase realizes $\mathcal{F}_{\mathsf{RandExp}}$

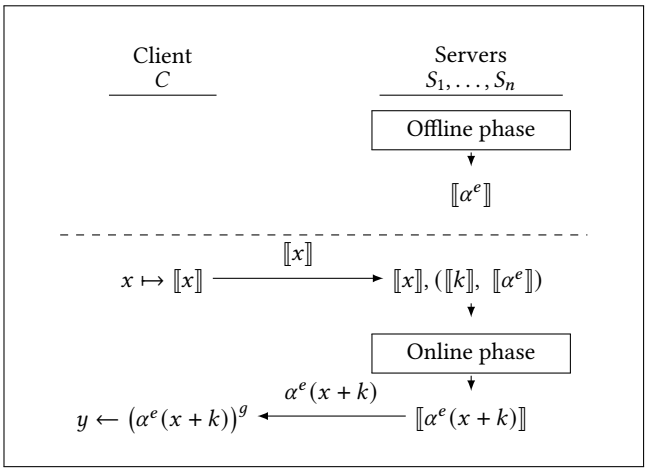


Figure 1: Offline/online Gold PRF $\mathsf{F}_k(x) = (x+k)^g \bmod p$.

(Fig. 4, §4.1); the online phase performs a verifiable multiply-and-open. We sketch both phases and the client-input protocol below; the formal descriptions follow in §5 and §6.

2.1 Why Replicated Secret Sharing

We work over *replicated secret sharing* (RSS) rather than Shamir secret sharing because, in the honest-majority setting $t < n/2$ for small n, RSS offers two advantages in terms of concrete efficiency. *First*, RSS admits a more efficient semi-honest multiplication protocol than Shamir secret sharing, which our offline generators and online multiply-and-open exploit directly. *Second*, the fully-linear-PCP-based (FLPCP-based) DZKP of [9]—which we use to verify semi-honest multiplications in batch—applies to any linear secret sharing in principle, but matches RSS particularly well: a semi-honest RSS multiplication already produces an *additive* sharing of the product, which is precisely the witness shape the FLPCP requires. Over Shamir the same framework applies, but the degree-$2t$ output of a Shamir multiplication must first be reinterpolated back to degree t—adding an extra interactive round and $\Theta(n)$ communication per multiplication layer—or else verified through a degree-$2t$ FLPCP variant whose witness lives over $2t{+}1$ evaluation points rather than $t{+}1$ additive shares, roughly doubling proof size and verifier work. Either way, Shamir pays a concrete overhead per multiplication that RSS avoids.

The cost RSS pays for this efficiency is a per-party complexity of $N' = \binom{n-1}{t}$ shares—versus $O(1)$ for Shamir—which grows combinatorially in n and would indeed favor Shamir in the large-n regime. In practice, however, threshold deployments consistently operate in the small-(t, n) regime, e.g., $n \le 9$, and in this regime the per-party share count stays small (at most $\binom{9}{4} = 126$), and the complexity of dealing with RSS's local shares remains modest.

2.2 Offline Phase

We propose two novel methods for batched generation of $[\![\alpha^e]\!]$; both avoid the $O(\log e)$ sequential modular exponentiations and concretely outperform the state-of-the-art protocol of Aly et al. [4, 30] for computing $[\alpha^e] \leftarrow \exp([\alpha], e)$. Our first method works over what we call the *degenerate additive encoding*, which exists in RSS and enables efficient secure multiplication by converting locally computed multiplicative RSS to additive RSS at low cost. Our second approach reduces the multiplication count to t using an additional tailored post-quantum ZKP. We give two instantiations, Π_{VitH} (VOLE-in-the-Head) and Π_{Lig} (Ligero). Both are adapted to a non-standard multi-prover/multi-verifier setting in which $t{+}1$ servers prove dual-share consistency to verifiers that each hold only a partial witness—an interesting use case of ZKP in MPC. Together these give three instantiations; we provide all three precisely because no single one dominates across all deployment settings—the trade-offs among (t, n), batch parameter m, bandwidth, and latency are intricate, and we lay them out in detail in §7.

2.3 Online Phase

Given $([\![k]\!], [\![\alpha^e]\!])$ and $[\![x]\!]$, servers compute $[\![\alpha^e(x+k)]\!]$ via a *verifiable inner product* protocol, introduced in this paper, that fuses the secret-shared multiplication with a batched DZKP optimized for the designated-verifier setting. The client reconstructs $\alpha^e(x+k)$ and locally raises it to g to recover $\mathsf{F}_k(x)$. Our online protocol is concretely efficient in terms of communication and rounds. In particular, our protocol batches across both servers and parallel queries, and incurs $O(3L \log L\,(n{-}1))$ server-to-server and $O(n)$ server-to-client communication, where $L = N^2/n$.

2.4 Verifiable Client Input

A secret-shared input is only as trustworthy as its consistency. If the client generated the replicated sharing $[\![x]\!]$ itself, a malicious client could send different copies of the same replicated component to different servers, which would then evaluate on a point the client never committed to. Π_{Input} (§6.5) instead lets the servers open a pre-computed random mask r to the client with abort; the client broadcasts $u \coloneqq r + x$, and every server sets $[\![x]\!] \coloneqq u - [\![r]\!]$. The servers echo a hash of u to force a single, well-defined input, and this echo rides a hash exchange that the online protocol already performs. As a side effect, the client uploads one field element per server per input instead of a full replicated share.

3 Preliminaries

Notations. Let $[a] \coloneqq \{1, 2, \ldots, a\}$ for any integer a. We denote by $x \xleftarrow{\$} S$ that x is sampled uniformly at random from a finite set S. Throughout, p denotes a prime, $\mathbb{F}_p$ the corresponding finite field, and $\mathbb{F}_p^*$ its multiplicative group. We let $\mathbb{G}$ denote the g-th power-residue subgroup of $\mathbb{F}_p^*$. We write κ for the statistical and λ for the computational security parameter. Throughout we assume an honest majority, $n \ge 2t + 1$; the few formulas that require the tight case state $n = 2t + 1$ explicitly.

3.1 Secret Sharing Schemes

In a t-out-of-n secret sharing scheme, a secret value s is distributed across n *shares*, denoted by $[\![s]\!] = \{[\![s]\!]_1, \ldots, [\![s]\!]_n\}$, and can be reconstructed from any combination of at least $t + 1$ shares, while any t shares leak no information about s. For any shares $[\![s]\!]$, we implicitly assume that party P_i is holding $[\![s]\!]_i$, and computing $f([\![s]\!])$ for any function f means that each party P_i computes $f([\![s]\!]_i)$. We use the notation $[\![s]\!]_I \coloneqq \{[\![s]\!]_i\}_{i \in I}$ for a subset of shares corresponding to

$\mathcal{I} \subset [n]$. Formally, there are two interactive protocols: Share(s), which takes the secret s as input and outputs n shares $[\![s]\!]$ for all parties, and Reconstruct$([\![s]\!]_Q, i)$, which takes $[\![s]\!]_Q$ and reveals the secret to party P_i by sending the shares in $[\![s]\!]_Q$ to P_i. The latter one outputs the secret s only if $|Q| \geq t+1$. We say that shares are *consistent* if Reconstruct$([\![s]\!]_{Q_0}, i)$ = Reconstruct$([\![s]\!]_{Q_1}, i)$ for any two sets of honest parties $Q_0, Q_1 \subset [n]$ such that $|Q_b| \geq t+1$ for all $b \in \{0,1\}$. Plus, an interactive algorithm Open$([\![s]\!])$ takes consistent shares $[\![s]\!]$ and outputs the secret s to all parties.

3.1.1 Local Linear Operations and Share Conversion. In this work, we consider secret sharing schemes that support local linear operations, such as $[\![a+b]\!] \coloneqq [\![a]\!]+[\![b]\!]$, $[\![c+a]\!] \coloneqq c+[\![a]\!]$, and $[\![c\cdot a]\!] \coloneqq c\cdot[\![a]\!]$. The *local share conversion* [14] allows parties to locally generate shares $[\![[\![s]\!]_i]\!]$ of the i-th share for any $i \in [n]$.

3.1.2 Additive Secret Sharing. Additive secret sharing (ASS) is an $(n-1)$-out-of-n secret sharing scheme supporting the local linear operations: a secret $s \in \mathbb{F}_p$ is divided into n random field elements, called *additive shares*, denoted by $[s]_1, \ldots, [s]_n$. The secret s can be recovered by computing $s = \sum_{i=1}^{n} [s]_i$ over $\mathbb{F}_p$. A multiplicative variant $[s]^*$ can also be defined as $s = \prod_{i=1}^{n} [s]_i^*$ over $\mathbb{F}_p^*$.

3.1.3 Replicated Secret Sharing. Replicated secret sharing (RSS) generalizes ASS to handle an arbitrary threshold t by providing the same additive shares of s to multiple parties. Consider an ASS scheme that produces $N \coloneqq \binom{n}{t}$ shares $[s]_1, \ldots, [s]_N$ indexed by a set $\mathcal{T} \coloneqq \{T \subset [n] \mid |T| = t\} = \{T_1, \ldots, T_N\}$. A secret s is shared as *RSS shares* $[\![s]\!] = \{[\![s]\!]_1, \ldots, [\![s]\!]_n\}$, defined by $[\![s]\!]_i \coloneqq \{[s]_j \mid i \notin T_j \in \mathcal{T}\}$ for all $i \in [n]$. Therefore, each RSS share consists of $N' \coloneqq \binom{n-1}{t}$ field elements. Any $t+1$ RSS shares $\{[\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_{t+1}}\}$ cover all additive shares, thereby recovering s, whereas any set of at most t RSS shares does not cover at least one additive share. RSS supports the local linear operations and the local share conversion.

A note on notation. We write $[s]_\ell$ with $\ell \in [N]$ for the additive *component* of an RSS sharing indexed by $T_\ell \in \mathcal{T}$, and $[s]_i$ with $i \in [n]$ for the i-th share of a standalone $(n-1)$-out-of-n additive sharing (as produced by $\mathcal{F}_{\mathsf{Zero}}$, or as the first component of a double sharing $([r], [\![r]\!])$). The two are distinct objects—there are $N = \binom{n}{t}$ of the former and n of the latter—and we reserve ℓ for component indices and i for party indices throughout.

The local share conversion is realized in RSS as follows. Party P_i is arbitrarily fixed. For every $T \in \mathcal{T}$ such that $i \notin T$, let $[\![[s]_T]\!]_j \coloneqq [s]_T$ for all $j \notin T$ and $[\![[s]_T]\!]_j \coloneqq 0$ for all $j \in T$. Therefore, every party P_j can locally compute $[\![[\![s]\!]_i]\!]_j \coloneqq \{[\![[s]_T]\!]_j \mid j \notin T \in \mathcal{T}\}$.

A multiplicative RSS $[\![s]\!]^*$ can also be defined by replacing the underlying additive shares $[s]$ with their multiplicative variant $[s]^*$.

3.1.4 Verifiable Secret Sharing. Verifiable secret sharing (VSS) guarantees that at the end of the Share protocol, either the parties hold consistent shares of a secret or the honest parties abort the protocol. This can be achieved by an additional consistency check after each party receives its share from the dealer.

3.2 Basic Ideal Functionalities

Our constructions invoke the following standard ideal functionalities; their realizations over RSS are given in Appendix A.

Players: Servers $S_1, \ldots, S_n$ and adversary $\mathcal{A}$ corrupting up to $t < n/2$ servers.

Input: Each server S_i inputs its local shares $[\![c]\!]_i$, $\{[\![a_k]\!]_i\}_{k\in[L]}$, and $\{[\![b_k]\!]_i\}_{k\in[L]}$.

Computation: Reconstruct c, $\{a_k\}_{k\in[L]}$, and $\{b_k\}_{k\in[L]}$ from the honest servers' shares. Check whether $c = \sum_{k=1}^{L} a_k \cdot b_k$ holds. If the check holds, deliver accept to $\mathcal{A}$, which replies with continue or abort. If the check does not hold, set the adversary's reply to abort.

Output: On continue, send accept to all servers; on abort, send reject to all servers.

Figure 2: $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ – Proving a degree-2 relation over RSS

- $\mathcal{F}_{\mathsf{Rand}}$: hands parties shares of a random secret value, while allowing the adversary to choose the corrupted parties' shares.
- $\mathcal{F}_{\mathsf{Coin}}$: hands parties fresh random coins.
- $\mathcal{F}_{\mathsf{RSS.Share}}$: on input a secret $s \in \mathbb{F}_p$ from a dealer, outputs $[\![s]\!]$ to parties.
- $\mathcal{F}_{\mathsf{RSS.Open}}$: on input $[\![s]\!]$ from parties, outputs s to all parties, or $\perp$ if the shares are inconsistent (security with abort).
- $\mathcal{F}_{\mathsf{A2R}}$: on input an additive sharing $[s]$ from parties, outputs a replicated sharing $[\![s]\!]$ of the same value.
- $\mathcal{F}_{\mathsf{RSS.Mul}}$: on input $[\![a]\!]$, $[\![b]\!]$ from parties, outputs $[\![ab]\!]$.
- $\mathcal{F}_{\mathsf{DoubleRand}}$: samples a uniformly random $r \in \mathbb{F}_p$ and outputs a double sharing $([r], [\![r]\!])$ to parties.
- $\mathcal{F}_{\mathsf{Zero}}$: hands parties additive shares of zero.
- $\mathcal{F}_{\mathsf{Setup}}$: a one-time setup that hands every server S_i the replicated PRF keys k_T for all $T \in \mathcal{T}$ with $i \notin T$ and the pairwise PRG seeds $s_{i,j}$ it shares with every other server S_j; for each designated dealer S_i of Π_{ZKPGen} and each $T \in \mathcal{T}$, it also hands a seed $\mathsf{sd}_{i,T}$ to S_i and to every server in $[n] \setminus T$.

3.3 Distributed Zero-Knowledge Proofs

Distributed zero-knowledge proof (DZKP) systems (over secret sharing) provide proofs of a statement on an input that is distributed among several verifiers. Adapting the framework of Boyle et al. [11], our protocols use DZKPs to verify degree-2 relations over RSS-shared values through the functionality $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ (Fig. 2).

At a high level, the functionality receives RSS sharings $[\![c]\!]$, $\{[\![a_k]\!]\}_{k\in[L]}$, and $\{[\![b_k]\!]\}_{k\in[L]}$, reconstructs the corresponding values, and checks whether $c = \sum_{k=1}^{L} a_k \cdot b_k$ holds. If the relation holds, the adversary is allowed to choose whether the servers continue or abort, capturing security with abort. If the relation does not hold, the functionality outputs reject to all servers. Thus, the adversary may prevent progress, but cannot make the servers accept an invalid degree-2 relation.

This functionality is used in our offline protocols to verify multiplication chains. For example, Π_{DegGen} uses it to check the relations $q_j = q_{j-1} \cdot M_{T_j}$ arising when converting multiplicative RSS into additive RSS, and Π_{ZKPGen} uses it to verify the multiplication tree that accumulates the designated dealers' contributions. We also use it in the analysis of the naive dVOPRF baseline, which relies on the standard DZKP-based multiplication-verification framework.

Players: Servers $S_1, \ldots, S_n$, client C, adversary $\mathcal{A}$ corrupting up to t servers and, possibly, C, and $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$.

Input: Each server S_i inputs its share $[\![k]\!]_i$ of the PRF key k; C inputs m evaluation points $(x^{(1)}, \ldots, x^{(m)}) \in \mathbb{F}_p^m$. A corrupted C may submit arbitrary points of $\mathcal{A}$'s choice, or the abort signal $\perp$.

Computation: If C's input is $\perp$, $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$ halts. Otherwise, it reconstructs k from the honest servers' shares $[\![k]\!]_{\mathcal{H}}$ and sets $c^{(j)} \leftarrow \mathsf{F}_k(x^{(j)})$ for every $j \in [m]$; delivers the corrupted parties' shares to $\mathcal{A}$, which replies with continue or abort.

Output: On continue, send $(c^{(1)}, \ldots, c^{(m)})$ to C; on abort, or if $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$ halted, send $\perp$ to C. Servers receive no output.

Figure 3: $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$ – dVOPRF for PRF F

4 Verifiable Distributed Oblivious PRF

This section formalizes the dVOPRF primitive informally introduced in §1. We write $\mathsf{F}_k : \mathcal{D} \to \mathcal{R}$ for a pseudorandom function indexed by a key $k \xleftarrow{\$} \mathcal{K}$, and capture the dVOPRF setting [2, 21, 27, 28] as the ideal functionality $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$.

$\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$ (Fig. 3) reconstructs k from the honest servers' shares and returns $(\mathsf{F}_k(x^{(j)}))_{j\in[m]}$ to the client, modulo a continue/abort decision by the adversary $\mathcal{A}$ that captures *security with abort* in the malicious honest-majority setting. **In this work, we focus on how to efficiently instantiate $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$.**

Note that $\mathcal{F}_{\text{dVOPRF}}^{\mathsf{F}_k}$ accepts *any* $(x^{(1)}, \ldots, x^{(m)}) \in \mathbb{F}_p^m$, and a corrupted C may submit points of $\mathcal{A}$'s choosing. This is deliberate: input *well-formedness* is an obligation of the protocol, not a restriction of the functionality. Guaranteeing that the servers end up holding a consistent sharing of a single, well-defined input—even when C is malicious—is exactly what Π_{Input} (§6.5) provides, and what lets us drop any honesty assumption on the client in Theorem 6.4.

4.1 Random Power Generation

The offline phase realizes $\mathcal{F}_{\mathsf{RandExp}}$ (Fig. 4): it hands the servers a fresh RSS sharing of α^e for a uniformly random α, and every online evaluation consumes one such sharing. Like $\mathcal{F}_{\mathsf{Rand}}$ (§3.2), it lets the adversary choose the corrupted servers' shares, while the shared value remains uniformly random.

4.2 PRF Candidate: Gold PRF

We use *Gold PRF* [34], one of the most efficient post-quantum PRF in terms of the balanced trade-off between computation and communication, as the PRF in our protocol construction. Gold PRF is an instantiation of the *power-residue PRFs*, which work over a *power-residue subgroup* of $\mathbb{F}_p^*$, where p is a sufficiently large prime of $O(\lambda)$ bits such that $p \coloneqq eg + 1$ for public integers e and g. Define the *g-th power-residue subgroup* $\mathbb{G} \coloneqq \{a^g \bmod p \mid a \in \mathbb{F}_p^*\}$, which is the unique subgroup of $\mathbb{F}_p^*$ of order e. A power-residue PRF $\mathsf{F}_k : \mathbb{F}_p \to \mathbb{G}$ is defined as $\mathsf{F}_k(x) \coloneqq (x + k)^g \bmod p$.[1] Gold PRF sets $e = 2^\lambda$ (so $|\mathbb{G}| = 2^\lambda$), and thus a single evaluation yields an output carrying λ bits of entropy. Yang et al. [34] analyzed the post-quantum security of Gold PRF, which is based on the underlying computational problem called the *Decisional Shifted Power-Residue Symbol* (DSPRS) problem.

Note that the Power-Residue PRF can yield a $\lfloor \log e \rfloor$-bit output. In particular, we consider two variants of the Power-Residue PRF that outputs a $\lfloor \log e \rfloor$-bit string, i.e., $F_k : \mathbb{F}_p \to \{0,1\}^{\lfloor \log e \rfloor}$: the *Legendre PRF* [15] and the *Gold PRF* [34].

- Legendre PRFs: When $e = 2$, the Legendre symbol $\left(\frac{a}{p}\right)$ is exactly the map that connects $\mathbb{F}_p^*$ to $\mathbb{G}$. Since $g = (p-1)/2$, the Power-Residue PRF with $e = 2$ can be expressed as $F_k(x) \coloneqq (k+x)^g = \left(\frac{k+x}{p}\right) \in \{1, -1\}$, which is commonly referred to as the Legendre PRF. Therefore, a single evaluation of Legendre PRF can produce 1-bit randomness.
- Gold PRFs: Power-Residue PRF with $e = 2^\lambda$ is called the Gold PRF. A single evaluation of Gold PRF outputs a λ-bit element of $\mathbb{G}$. In particular, if h is a generator of $\mathbb{G}$, then any output $y \in \mathbb{G}$ can be written uniquely as $y = h^a$ for some exponent $a \in [2^\lambda]$, and this exponent corresponds to a λ-bit string. The exponent a can be recovered by solving the discrete logarithm of y to the base h; since $e = 2^\lambda$ is smooth, the Pohlig–Hellman algorithm [31] recovers a in $O(\lambda^2)$ group operations.

The security of the Legendre PRF relies on the *Decisional Shifted Legendre Symbol* (DSLS) problem. Kaluđerović et al. [28] analyzed the post-quantum security of the Legendre PRF under the DSLS assumption.

[1] Practical instantiations fix an injective encoding from bitstrings to $\mathbb{F}_p$ and explicitly handle the negligible-probability event $x + k \equiv 0 \pmod p$, in which $\mathsf{F}_k(x) = 0 \notin \mathbb{G}$.

Players: Servers $S_1, \ldots, S_n$, adversary $\mathcal{A}$ corrupting up to t servers, and $\mathcal{F}_{\mathsf{RandExp}}$.

Input: A public exponent $e \in \mathbb{Z}^+$.

Computation: $\mathcal{F}_{\mathsf{RandExp}}$ samples $\alpha \xleftarrow{\$} \mathbb{F}_p$ and computes α^e. It receives from $\mathcal{A}$ the components of $[\![\alpha^e]\!]$ held by corrupted servers, samples the remaining components of $[\![\alpha^e]\!]$ uniformly subject to α^e being the shared value, and asks $\mathcal{A}$ to reply with continue or abort.

Output: On continue, send $[\![\alpha^e]\!]_i$ to each honest S_i; on abort, send $\perp$ to all honest servers.

Figure 4: $\mathcal{F}_{\mathsf{RandExp}}$ – Random Power RSS Generation

5 Offline Phase

The offline phase prepares $[\![\alpha^e]\!]$ for $\alpha \xleftarrow{\$} \mathbb{F}_p$, secure against $t < n/2$ malicious servers. Since every online dVOPRF evaluation consumes one fresh instance of $[\![\alpha^e]\!]$, achieving high throughput hinges on generating this correlated randomness in bulk efficiently.

State-of-the-art over general additive secret sharing. Given a secret base $[\alpha]$ and a public exponent e, Aly et al.'s protocol [4, Protocol 7], stated over $\mathcal{F}_{\mathsf{ABB}}$, is the state-of-the-art for generating $[\alpha^e]$: it replaces $\log e$ sequential multiplications by two public-base exponentiations, one secure multiplication, and one public opening, via $[\bar{r}] \leftarrow \exp(g, [r'])$, $[c] \leftarrow [\bar{r}] \cdot [\alpha]$, $c \leftarrow \mathsf{Open}([c])$,

$[\alpha^e] \leftarrow c^e \cdot \exp(g, -e[r'])$. Round and communication complexity are thus independent of the public exponent, dominated by two calls to $\Pi_{\exp(b,[\cdot])}$ [4, Protocol 6]. Mennink [30] re-analyzed these protocols and reported the malicious cost as 10 rounds, $4n$+11 multiplications, and $8n$+19 openings per invocation – the baseline for additively-shared instantiations under $\mathcal{F}_{\mathsf{ABB}}$.

Our RSS adaptation Π_{AlyGen}. We port Aly et al.'s protocol to the honest-majority RSS setting and refer to the resulting protocol as Π_{AlyGen}; the detailed instantiation is deferred to Protocol 14 in Appendix B. The high-level structure is unchanged compared to [4, Protocol 7], but two RSS-specific factors increase communication complexity. First, exponentiation by a public scalar is *local* in RSS, but each public-base exponentiation $[\![\bar{r}]\!] \leftarrow \exp(g, [r'])$ and $[\![\rho]\!] \leftarrow \exp(g, [\bar{e}])$ produces $N = \binom{n}{t}$ locally-computed field elements (one per replicated component of the secret exponent) that must be folded into a standard RSS sharing via $N-1$ sequential $\Pi_{\mathsf{RSS.Mul}}$ steps. Second, the malicious upgrade is realized by a single batched $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$ DZKP over all multiplications, deferred to the end of the protocol. A single invocation of Π_{AlyGen} therefore costs $2(N-1)+1 = 2N-1$ $\Pi_{\mathsf{RSS.Mul}}$ invocations, 1 public opening, 1 batched DZKP, and a per-generation $\mathbb{F}_p \leftrightarrow \mathbb{Z}_{p-1}$ share conversion. In the RSS setting, Π_{AlyGen} thus presents two structural disadvantages: (i) the multiplication count is $O(N)$ rather than $O(n)$; and (ii) it requires a per-instance $\mathbb{F}_p \leftrightarrow \mathbb{Z}_{p-1}$ share conversion, which further complicates protocol instantiation.

To overcome these disadvantages, we present two novel protocols for generating $[\![\alpha^e]\!]$, denoted Π_{DegGen} and Π_{ZKPGen}, both tailored to the RSS structure: they eliminate the share conversion and avoid high communication volume. Π_{ZKPGen} contains two concrete ZKP-dependent instantiations: $\Pi^{\mathsf{VitH}}_{\mathsf{ZKPGen}}$ (VOLEitH-based) and $\Pi^{\mathsf{Lig}}_{\mathsf{ZKPGen}}$ (Ligero-based). Tab. 1 summarizes the resulting three constructions in terms of primitives, security assumption, and asymptotic cost.

Table 1: Offline Phase Protocol Overview

Protocol	Primitives	Sec.	#Mul.*	Extra Comm.†
Π_{AlyGen} [4, 30]	$\Pi_{\exp(b,[\cdot])}$ + DZKP	IT	$2N-1$	$O(1)$
Our Π_{DegGen}	DegMul + DZKP	IT	$(N-1)^{\ddagger}$	0
Our $\Pi^{\mathsf{VitH}}_{\mathsf{ZKPGen}}$	RSS.Mul + VitH + DZKP	CRH	t	$O(t\lambda N)$
Our $\Pi^{\mathsf{Lig}}_{\mathsf{ZKPGen}}$	RSS.Mul + Lig + DZKP	CRH	t	$O(tn(N+\lambda))$

$n \geq 2t+1$, $N=\binom{n}{t}$, λ: computational security parameter. IT: information-theoretic; CRH: collision-resistant hashing. *Number of $\Pi_{\mathsf{RSS.Mul}}$ or Π_{DegMul} invocations. ‡Counts Π_{DegMul} invocations. †Extra communication beyond multiplication overhead; Π_{AlyGen} additionally requires $\mathbb{F}_p \leftrightarrow \mathbb{Z}_{p-1}$ share conversion.

In brief, Π_{DegGen} roughly halves the multiplication count of Π_{AlyGen} and removes its opening and $\mathbb{F}_p \leftrightarrow \mathbb{Z}_{p-1}$ conversion while remaining IT secure; Π_{ZKPGen} further drops the count to t and gains post-quantum security from collision-resistant hashing, at the cost of extra ZKP communication. We present Π_{DegGen} and Π_{ZKPGen} in turn below, with concrete benchmarks deferred to §7.1.

5.1 Protocol Π_{DegGen}

Rather than computing $[\![\alpha^e]\!]$ from an additive sharing $[\![\alpha]\!]$, we start from a multiplicative secret sharing $[\![\alpha]\!]^*$. We observe that two structural properties of $[\![\alpha]\!]^*$ make this advantageous:

(1) **Commutativity with exponentiation.** If $\alpha = \prod_{T\in\mathcal{T}} m_T$ is the secret reconstructed from the multiplicative sharing $[\![\alpha]\!]^*$, then

$$\alpha^e = \Big(\prod_{T\in\mathcal{T}} m_T\Big)^e = \prod_{T\in\mathcal{T}} m_T^e.$$

Each server can therefore *locally* raise its multiplicative shares to the power e, obtaining a multiplicative sharing $[\![\alpha^e]\!]^*$ without any interaction.

(2) **Degenerate structure.** Each component m_T of $[\![\alpha]\!]^*$ is held by exactly the $n-t$ servers in $[n] \setminus T$, forming a special type of additive replicated secret sharing that we call a *degenerate additive encoding* (Definition 5.1).

It remains to convert $[\![\alpha^e]\!]^*$ into its standard additive form $[\![\alpha^e]\!]$. We accomplish this by sequentially multiplying all N degenerate encodings via a dedicated protocol Π_{DegMul}, and then verifying all multiplication results in a single batched invocation of $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$. Below, we first define the degenerate additive encoding and the multiplication protocol, and lastly present the complete protocol Π_{DegGen}.

Definition 5.1 (Degenerate Additive Encoding). Let $v \in \mathbb{F}_p$ and $T \in \mathcal{T}$. We say that a standard RSS $[\![v]\!]$ is a *degenerate additive encoding* with respect to T, denoted $\langle v \rangle_T$, if its components $\{s_{T'}\}_{T'\in\mathcal{T}}$ satisfy

$$s_{T'} = \begin{cases} v & \text{if } T' = T, \\ 0 & \text{otherwise.} \end{cases}$$

With parties holding $\langle v \rangle_T$, under the standard RSS distribution rule, the $n-t$ parties in $[n] \setminus T$ all hold the identical value v, while the remaining t parties in T hold only zeros. Hence, we observe that any standard RSS $[\![s]\!]$ with components $\{s_T\}_{T\in\mathcal{T}}$ is composed of $N = |\mathcal{T}| = \binom{n}{t}$ degenerate additive encodings, i.e., $\{\langle s_T \rangle_T\}_{T\in\mathcal{T}}$.

5.1.1 Target-Matched Double Randomness. Multiplying a standard RSS by a degenerate encoding requires correlated randomness whose additive part involves only the parties that hold the non-zero component. We adapt $\Pi_{\mathsf{DoubleRand}}$ (Protocol 13 in Appendix A) to produce a double sharing $([r], [\![r]\!])$ of the same random r, where $[\![r]\!]$ is a standard RSS (held by all parties) and $[r]$ is an additive sharing restricted to a target subset $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \geq t+1$ (parties outside $\mathcal{A}$ hold zero).

The construction follows the same approach as $\Pi_{\mathsf{DoubleRand}}$: a public covering policy $P_{\mathcal{A}} : \mathcal{T} \to \mathcal{A}$ with $P_{\mathcal{A}}(T) \notin T$ assigns each RSS component r_T to exactly one party in $\mathcal{A}$ that holds key k_T. Since the resulting base shares are deterministic (any party holding k_T can recompute them), we add a locally generated *random secret sharing of zero* among the parties in $\mathcal{A}$, derived from pairwise PRG seeds, to re-randomize the additive shares.

Correctness. The pairwise PRG terms form a secret sharing of zero among $\mathcal{A}$: each term $\mathsf{PRG}(s_{i,j}, \mathsf{ctr})$ appears with +1 for one party and −1 for the other, canceling in the sum. Hence $\sum_{i\in\mathcal{A}} [r]_i = \sum_{T\in\mathcal{T}} r_T = r$.

Protocol 1 $\Pi^{\text{deg}}_{\text{DoubleRand}}$ – Target-Matched Double Randomness for Degenerate Encodings

Players: Servers $S_1, \ldots, S_n$ with $n \geq 2t+1$.

Pre-shared material: Replicated PRF keys $\{k_T\}_{T \in \mathcal{T}}$ (key k_T held by all S_i with $i \notin T$); $\forall i, j \in [n]$, pairwise PRG seeds $\{s_{i,j}\}$ shared between every pair of servers S_i, S_j.

Input: Target subset of parties $\mathcal{A} \subseteq [n]$ where $|\mathcal{A}| \geq t+1$; a public counter ctr.

`// Step 1 – RSS component (identical to` $\Pi_{\text{DoubleRand}}$`)`

1: For each $T \in \mathcal{T}$ with $i \notin T$, server S_i locally computes $r_T \leftarrow \text{PRF}_{k_T}(\text{ctr})$.

2: Each server S_i sets its replicated share $[\![r]\!]_i \leftarrow \{r_T \mid i \notin T\}$, which defines the standard RSS $[\![r]\!]$ with $r = \sum_{T \in \mathcal{T}} r_T$.

`// Step 2 – Re-randomized subset additive share`

3: Let $P_{\mathcal{A}} \colon \mathcal{T} \to \mathcal{A}$ be a public *covering policy* satisfying $P_{\mathcal{A}}(T) \notin T$ for every $T \in \mathcal{T}$ (such a map exists since $|\mathcal{A}| \geq t+1$ and $|T| = t$). Each server $S_i \in \mathcal{A}$ locally computes:

$$[r]_i \leftarrow \sum_{T \in \mathcal{T}:\ P_{\mathcal{A}}(T) = i} r_T + \sum_{\substack{j \in \mathcal{A} \\ j \neq i}} (-1)^{1[j<i]} \,\text{PRG}(s_{i,j}, \text{ctr}).$$

Output: $([r], [\![r]\!])$, where $[r]_i = 0$ for $i \notin \mathcal{A}$.

Non-interactivity. All computations are local: Step 1 uses replicated PRF keys; Step 2 uses the deterministic inclusion rule (from locally held PRF values) and pairwise PRG seeds. Although multiple parties hold the same key k_T and can compute the same r_T, the covering policy $P_{\mathcal{A}}$ ensures each r_T is counted exactly once.

Privacy. Let $C \subset [n]$ with $|C| \leq t$ be the corrupted set, and let $\mathcal{H}_{\mathcal{A}} \coloneqq \mathcal{A} \setminus C$ be the honest parties in $\mathcal{A}$; since $|\mathcal{A}| \geq t+1$, we have $|\mathcal{H}_{\mathcal{A}}| \geq 1$. Because S_i holds k_T exactly when $i \notin T$, the adversary can recompute r_T precisely for those T with $C \nsubseteq T$; for every $T \supseteq C$ no corrupted party holds k_T, so r_T is pseudorandom in its view, and at least one such T exists because $|C| \leq t = |T|$. Likewise $\text{PRG}(s_{h,h'}, \text{ctr})$ is pseudorandom in its view whenever both $h, h' \in \mathcal{H}_{\mathcal{A}}$, and by security of the PRF and the PRG these values are jointly pseudorandom and mutually independent.

The covering policy places every component the adversary cannot recompute on an *honest* party of $\mathcal{A}$: for $T \supseteq C$ we have $P_{\mathcal{A}}(T) \in \mathcal{A}$ by definition and $P_{\mathcal{A}}(T) \notin T \supseteq C$ by the covering condition, hence $P_{\mathcal{A}}(T) \in \mathcal{H}_{\mathcal{A}}$. Since $r = \sum_{T \in \mathcal{T}} r_T$ contains such a term, r itself is pseudorandom in the adversary's view.

Fix one $T^* \supseteq C$ and put $h^* \coloneqq P_{\mathcal{A}}(T^*) \in \mathcal{H}_{\mathcal{A}}$. Writing e_h for the standard basis vector of coordinate h, the unknown r_{T^*} shifts the honest additive shares $([r]_h)_{h \in \mathcal{H}_{\mathcal{A}}}$ along e_{h^*}, while the unknown $\text{PRG}(s_{h,h'}, \text{ctr})$ of each honest pair $\{h, h'\} \subseteq \mathcal{H}_{\mathcal{A}}$ shifts them along $e_h - e_{h'}$. The pair directions span the $(|\mathcal{H}_{\mathcal{A}}| - 1)$-dimensional subspace of zero-sum vectors, and e_{h^*} has coordinate sum 1 and so lies outside it, so together they span $\mathbb{F}_p^{|\mathcal{H}_{\mathcal{A}}|}$. Hence the honest parties' additive shares are *jointly* computationally indistinguishable from uniform, even though the adversary can recompute every other r_T from its replicated keys. In the extreme case $\mathcal{A} = C \cup \{h\}$ there is no honest pair at all, and privacy rests entirely on r_C, which the covering condition necessarily assigns to h.

5.1.2 Multiplication of Standard RSS by Degenerate Encoding. Π_{DegMul} (Protocol 2) realizes secure multiplication of a standard RSS $[\![a]\!]$ by a degenerate additive encoding $\langle v \rangle_{T'}$. It follows the same structure as the DN protocol—masking the cross-product with correlated randomness and publicly reconstructing the masked difference—but exploits the sparse structure of degenerate encodings to reduce both computation and communication. As in the DN protocol, the required correlated randomness (a dual sharing $([r], [\![r]\!])$) is generated by $\Pi^{\text{deg}}_{\text{DoubleRand}}$ (Protocol 1, §5.1.1), which restricts the additive share $[r]$ to the target subset $\mathcal{A} = [n] \setminus T'$ (with $[r]_i = 0$ for $i \in T'$) so that only the $t+1$ parties in $\mathcal{A}$ contribute to reconstructing r, in contrast to the standard $\Pi_{\text{DoubleRand}}$ which involves all n parties.

This sparsity carries through to the multiplication itself. On the computation side, only parties in $\mathcal{A}$ hold the non-zero component of $\langle v \rangle_{T'}$, so the cross-multiplication involves $|\mathcal{T}| + 1$ non-zero terms instead of $|\mathcal{T}|^2$ in general RSS multiplication, while the t parties in T' hold zero shares and skip cross-multiplication entirely. On the communication side, the target-matched double sharing ensures that parties in T' have $d_i = 0$ and remain silent: only the $t+1$ parties in $\mathcal{A}$ communicate, each sending one field element to S_{agg}, which broadcasts δ back to $\mathcal{A}$, with the output remaining a standard RSS.

Protocol 2 Π_{DegMul} – Deg. Encoding Multiplication

Players: Servers $S_1, \ldots, S_n$.

Input: $[\![a]\!]$; degenerate encoding $\langle v \rangle_{T'}$ for some $T' \in \mathcal{T}$; target-matched double sharing $([r], [\![r]\!])$ from $\Pi^{\text{deg}}_{\text{DoubleRand}}$ (Protocol 1) with target subset $\mathcal{A}$, where $\mathcal{A} = [n] \setminus T'$ has a designated aggregator $S_{\text{agg}} \in \mathcal{A}$.

1: **foreach** $S_i \in \mathcal{A}$ **do**
2: Cross-multiply: $[av]_i \leftarrow [\![a]\!]_i \cdot \langle v \rangle_{T'}$.
3: Send $d_i \leftarrow [av]_i - [r]_i$ to S_{agg}.
4: **end**
5: S_{agg} reconstructs $\delta \leftarrow \sum_{i \in \mathcal{A}} d_i = av - r$ and broadcasts δ to $\mathcal{A}$, so that each server in $\mathcal{A}$ computes $[\![av]\!] = \delta + [\![r]\!]$.

Output: $[\![av]\!]$.

5.1.3 Full Protocol. We combine all sub-protocols into the full generation protocol Π_{DegGen} (Protocol 3), which proceeds in three steps. First, in a *local exponentiation* step, each server S_i raises its multiplicative shares m_T to the power e, obtaining $M_T = m_T^e$ for every T with $i \notin T$; since $\prod_{T \in \mathcal{T}} M_T = \alpha^e$, this yields an RSS multiplicative sharing of α^e, requiring each server to derive $\binom{n-1}{t}$ shares from PRG seeds and compute $\binom{n-1}{t}$ modular exponentiations. Next, in the *sequential multiplication* step, servers convert the multiplicative sharing into an additive RSS by sequentially accumulating the degenerate encodings $\langle M_{T_j} \rangle_{T_j}$ via Π_{DegMul}: fixing an ordering $T_1, \ldots, T_N$ of $\mathcal{T}$, let

$$[\![q_1]\!] = \langle M_{T_1} \rangle_{T_1}, \quad [\![q_j]\!] = [\![q_{j-1}]\!] \cdot \langle M_{T_j} \rangle_{T_j} \text{ for } 2 \leq j \leq N,$$

so that after $N-1$ multiplications, $[\![q_N]\!] = [\![\alpha^e]\!]$. In each invocation, only the $t+1$ parties in $\mathcal{A}_j$ participate, each sending 1 field element to S_{agg}, which broadcasts δ back, for $2(t+1)$ field elements per multiplication. Since the cross-multiplication involves $|\mathcal{T}| + 1$ non-zero terms instead of $|\mathcal{T}|^2$ as in general RSS multiplication, the total communication over the $N-1$ multiplications is $2(t+1)(N-1)$ field elements. Finally, in the *DZKP verification* step, all $N-1$ multiplication relations are verified in a single batched invocation of $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$, with sublinear communication complexity $O(n \log N)$.

Protocol 3 Π_{DegGen} – Offline Generation via Π_{DegMul}

Players: Servers $S_1, \ldots, S_n$.
Input: Public exponent e; multiplicative RSS $[\![\alpha]\!]^* = \{m_T\}_{T\in\mathcal{T}}$ with $\alpha = \prod_{T\in\mathcal{T}} m_T$; ordering $T_1, \ldots, T_N$ of $\mathcal{T}$.

1: **foreach** $S_i \in [n]$, $T \in \mathcal{T}$ *with* $i \notin T$ **do**
2: $\quad M_T \leftarrow m_T^e$; form $\langle M_T \rangle_T$.
3: **end**
4: Set $[\![q_1]\!] \leftarrow \langle M_{T_1} \rangle_{T_1}$.
5: **for** $j = 2, \ldots, N$ **do**
6: $\quad$ Let $\mathcal{A}_j \leftarrow [n] \setminus T_j$ and $([r], [\![r]\!]) \leftarrow \Pi^{\mathsf{deg}}_{\mathsf{DoubleRand}}(\mathcal{A}_j)$.
7: $\quad [\![q_j]\!] \leftarrow \Pi_{\mathsf{DegMul}}([\![q_{j-1}]\!], \langle M_{T_j} \rangle_{T_j}, ([r], [\![r]\!]))$.
8: **end**
9: Verify $\{q_j = q_{j-1} \cdot M_{T_j}\}_{j=2}^{N}$ via $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$; abort on failure.

Output: $[\![q_N]\!]$ as $[\![\alpha^e]\!]$.

5.1.4 Security. We assume a broadcast channel among the servers, so that all servers in $\mathcal{A}_j$ receive the same δ in each invocation of Π_{DegMul}, and that the $N-1$ relations checked in the last step of Protocol 3 are combined with random coefficients from $\mathcal{F}_{\mathsf{Coin}}$ into a single inner-product relation for $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$.

Theorem 5.2. *Let $t < n/2$. In the hybrid model with a broadcast channel and functionalities $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$, Π_{DegGen} (Protocol 3) securely realizes $\mathcal{F}_{\mathsf{RandExp}}$ with abort against a static malicious adversary corrupting up to t servers. If the values derived from the replicated keys and pairwise seeds are uniformly random, the statistical error is at most $N/|\mathbb{F}_p|$; deriving them with a PRF and a PRG adds the corresponding distinguishing advantages.*

Proof sketch. Let C be the corrupted set and fix $T^* \supseteq C$; such a T^* exists because $|C| \le t$, and no corrupted server holds m_{T^*}. The simulator emulates the hybrid functionalities and the honest servers, and runs Π_{DegGen} with the adversary. Besides values it can compute itself, the adversary only receives masked differences d_i and corrections δ from Π_{DegMul}; by the privacy of $\Pi^{\mathsf{deg}}_{\mathsf{DoubleRand}}$ (§5.1.1), the honest servers' additive shares of each fresh r are jointly uniform in its view, so these messages are uniform and independent of m_{T^*}. If the adversary makes some $q_j \neq q_{j-1} \cdot M_{T_j}$, the combined check rejects except with probability $1/|\mathbb{F}_p|$, and the servers abort. Otherwise $q_N = \prod_{T\in\mathcal{T}} M_T = \alpha^e$ for $\alpha = \prod_{T\in\mathcal{T}} m_T$, and the simulator hands the components of $[\![q_N]\!]$ held by corrupted servers, which it computes from the emulated keys and the broadcast corrections, to $\mathcal{F}_{\mathsf{RandExp}}$. Since m_{T^*} is uniform and hidden from the adversary, $\alpha = m_{T^*} \cdot \prod_{T\neq T^*} m_T$ is uniform in $\mathbb{F}_p$ unless one of the other $N-1$ components is zero; the honest servers' remaining output components are uniform subject to the shared value, exactly as sampled by $\mathcal{F}_{\mathsf{RandExp}}$. The full proof is in Appendix D.1. □

5.2 Protocol Π_{ZKPGen}

Π_{DegGen} requires $N-1$ invocations of Π_{DegMul}. Here we present Π_{ZKPGen} (Protocol 4) that reduces the number of multiplication invocations to t, at some additional ZKP cost.

The idea is to select $t+1$ designated dealers $\mathcal{A} \subset [n]$. Each dealer $S_i \in \mathcal{A}$ distributes a replicated secret sharing $[\![M^i]\!]$, where M^i admits an e-th root in $\mathbb{F}_p$. The servers then accumulate the product $[\![M]\!] = [\![\prod_{i\in\mathcal{A}} M^i]\!]$ using t invocations of $\Pi_{\mathsf{RSS.Mul}}$, thus reducing the multiplication invocations from $N-1$ to t. The challenge in this alternative is to prove that each M^i indeed admits an e-th root in $\mathbb{F}_p$; we address this via a post-quantum dual-share consistency ZKP, for which we provide two instantiations: Π_{VitH} (Protocols 15–16 in Appendix C.1), based on VOLE-in-the-Head with communication complexity $O(t\lambda N)$, and Π_{Lig} (Protocols 17–18 in Appendix C.2), based on the Ligero ZKP system with communication complexity $O(tn(N+\lambda))$. Both are non-interactive, require only collision-resistant hashing, and need no trusted setup. Concretely, Π_{ZKPGen} works as follows:

(1) Each of $t+1$ designated servers $S_i \in \mathcal{A}$ uses its seeds $\mathsf{sd}_{i,T}$ for $T \in \mathcal{T}$, each shared with the servers in $[n] \setminus T$ (see $\mathcal{F}_{\mathsf{Setup}}$), to locally derive multiplicative shares $\{m_T\}_{T\in\mathcal{T}}$ and additive shares $\{a_T\}_{T\in\mathcal{T}}$ with $A^i = \sum_{T\in\mathcal{T}} a_T$. Each server then locally raises its multiplicative shares to the e-th power, yielding $M^i = \prod_{T\in\mathcal{T}} (m_T)^e$, which is an e-th power by construction.
(2) The designated server S_i reconstructs both M^i and A^i, computes a correction $\delta \leftarrow M^i - A^i$, and broadcasts δ to all servers. Accordingly, each server updates its local share of M^i.
(3) S_i invokes $\Pi_{\mathsf{DualShareZKP}}$ to prove that δ was correctly computed, i.e., that the additive sharing is consistent with the multiplicative shares and that M^i is indeed an e-th power.
(4) After all $t+1$ designated servers have contributed and proven their factors, servers multiply the $t+1$ replicated sharings $[\![M^1]\!], \ldots, [\![M^{t+1}]\!]$ using t invocations of $\Pi_{\mathsf{RSS.Mul}}$, yielding $[\![\alpha^e]\!]$ where $\alpha^e = \prod_{i\in\mathcal{A}} M^i$.

Only $t+1$ parties need to contribute a factor, since at least one honest party's contribution suffices to randomize the product; with $t+1$ factors, even if t parties are corrupt, the product remains uniformly random from the adversary's perspective.

The remaining challenge is how to efficiently realize $\Pi_{\mathsf{DualShareZKP}}$, which operates under a non-standard ZKP setting: multiple distributed verifiers each hold a *subset* of the prover's witness (the PRG-derived shares). We present two concrete instantiations of $\Pi_{\mathsf{DualShareZKP}}$—one based on VOLE-in-the-Head and one based on Ligero, both requiring only collision-resistant hashing (CRH) and no trusted setup. Below, we first define the extended witness and the relation circuit that both instantiations prove, then summarize the key ideas and properties of each protocol (full protocol descriptions are deferred to Appendices C.1 and C.2).

We capture the guarantee required of each dealer's invocation of $\Pi_{\mathsf{DualShareZKP}}$ by the functionality $\mathcal{F}_{\mathsf{DualShareZKP}}$ (Fig. 5), which checks the dual-share relation against the shares that honest servers hold and reveals nothing beyond acceptance and the pairs that corrupted servers already hold. An honest dealer inputs $M_T = m_T^e$.

Theorem 5.3. *Let $t < n/2$ and $|\mathcal{A}| = t+1$. In the hybrid model with a broadcast channel and functionalities $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{DoubleRand}}$, $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{DualShareZKP}}$ and $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$, Π_{ZKPGen} (Protocol 4) securely realizes $\mathcal{F}_{\mathsf{RandExp}}$ with abort against a static malicious adversary corrupting up to t servers. If the seed-derived values are uniformly random, the statistical error is at most $(t+1)N/|\mathbb{F}_p|$; deriving them with a PRG adds its distinguishing advantage.*

Protocol 4 Π_{ZKPGen} – Offline Generation via $\Pi_{\mathsf{DualShareZKP}}$

Players: Servers $S_1, \ldots, S_n$.

Input: A designated dealer set $\mathcal{A} \subset [n]$ with $|\mathcal{A}| = t+1$; for each $i \in \mathcal{A}$ and $T \in \mathcal{T}$, a seed $\mathsf{sd}_{i,T}$ held by S_i and by every server in $[n] \setminus T$ (from $\mathcal{F}_{\mathsf{Setup}}$).

1: **for** $i \in \mathcal{A}$ **do**
2: Let $\mathcal{P} = S_i$ be the current dealer.
`// Dual-share derivation`
3: For every $T \in \mathcal{T}$, $\mathcal{P}$ and every server in $[n] \setminus T$ locally derive pseudorandom $m_T, a_T \in \mathbb{F}_p$ from $\mathsf{sd}_{i,T}$.
These implicitly define $M^i \leftarrow \prod_{T \in \mathcal{T}} (m_T)^e$ and $A^i \leftarrow \sum_{T \in \mathcal{T}} a_T$, with M^i uniform among e-th powers and A^i uniform in $\mathbb{F}_p$.
`// Correction broadcast`
4: $\mathcal{P}$ locally reconstructs M^i and A^i, computes $\delta \leftarrow M^i - A^i$, and broadcasts δ to all servers.
`// Share conversion`
5: Servers locally compute $[\![M^i]\!] \leftarrow \delta + [\![A^i]\!]$, where $[\![A^i]\!]$ has components $\{a_T\}_{T \in \mathcal{T}}$.
`// Consistency proof`
6: $\mathcal{P}$ invokes $\Pi_{\mathsf{DualShareZKP}}(\delta, \{a_T, m_T\}_{T \in \mathcal{T}})$ to prove that $\delta = M^i - A^i = \prod_T (m_T)^e - \sum_T a_T$; all other servers verify the non-interactive proof and broadcast their verdicts, and every server aborts unless all verdicts accept.
7: **end**
`// Accumulation via multiplication tree`
8: Order $\mathcal{A}$ as $i_1, \ldots, i_{t+1}$, set $[\![q_1]\!] \leftarrow [\![M^{i_1}]\!]$, and for $j = 2, \ldots, t+1$ compute $[\![q_j]\!] \leftarrow \Pi_{\mathsf{RSS.Mul}}([\![q_{j-1}]\!], [\![M^{i_j}]\!])$, yielding $[\![\alpha^e]\!] = [\![q_{t+1}]\!]$ after t invocations.
`// Multiplication verification`
9: Verify $\{q_j = q_{j-1} \cdot M^{i_j}\}_{j=2}^{t+1}$ via $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$; abort on failure.

Output: Each server S_j holds $[\![\alpha^e]\!]_j$, where $\alpha^e = \prod_{i \in \mathcal{A}} M^i$.

Players: Dealer S_i, servers S_j for $j \in [n] \setminus \{i\}$, and adversary $\mathcal{A}$ corrupting up to t servers.

Input: S_i inputs δ and $\{(M_T, a_T)\}_{T \in \mathcal{T}}$; each S_j inputs δ and $\{(m_T, a_T)\}_{T \in \mathcal{T}: j \notin T}$.

Computation: Set $b \leftarrow 1$ if all honest servers input the same δ, every pair (m_T, a_T) input by an honest S_j satisfies $m_T^e = M_T$ and matches the dealer's a_T, and $\delta = \prod_{T \in \mathcal{T}} M_T - \sum_{T \in \mathcal{T}} a_T$; otherwise set $b \leftarrow 0$. Send b, and the dealer's (M_T, a_T) for every T that some corrupted server is outside of, to $\mathcal{A}$, which replies with continue or abort.

Output: On continue and $b = 1$, send accept to all servers; otherwise send reject.

Figure 5: $\mathcal{F}_{\mathsf{DualShareZKP}}$ – Dual-Share Consistency Proof

Proof sketch. Since $|\mathcal{A}| = t+1$, some dealer $S_{i^*} \in \mathcal{A}$ is honest. Fix $T^* \supseteq C$ for the corrupted set C: the seed sd_{i^*,T^*} is held only by S_{i^*} and honest servers, so its pair (m_{T^*}, a_{T^*}) is uniform in the adversary's view. Consequently $\delta_{i^*} = M^{i^*} - A^{i^*}$ is uniform and independent of M^{i^*}, and $\mathcal{F}_{\mathsf{DualShareZKP}}$ reveals only acceptance. For a corrupted dealer, every T has an honest holder because $n - t > t$, so $\mathcal{F}_{\mathsf{DualShareZKP}}$ binds the dealer's pairs to the setup-determined values and $[\![M^i]\!]$ shares the e-th power $\prod_T m_T^e$, or the servers abort. As in Theorem 5.2, the combined degree-2 check forces $q_{t+1} = \prod_{i \in \mathcal{A}} M^i$ except with probability $1/|\mathbb{F}_p|$, and the simulator hands the components of $[\![q_{t+1}]\!]$ held by corrupted servers to $\mathcal{F}_{\mathsf{RandExp}}$. Unless one of the other derived components m_T is zero, the product equals $m_{T^*}^e \cdot c$ for a nonzero e-th power c independent of m_{T^*}, and is therefore distributed exactly as α^e for $\alpha \xleftarrow{\$} \mathbb{F}_p$. The full proof is in Appendix D.2. □

5.2.1 The Dual-Share Circuit C_{dual}. Both ZKP instantiations prove correctness of the same arithmetic circuit C_{dual} over $\mathbb{F}_p$. Recall that for each dealer $S_i \in \mathcal{A}$, the PRG-based derivation in Π_{ZKPGen} produces a pair $(m_T, a_T) \in \mathbb{F}_p^2$ for every subset $T \in \mathcal{T}$. The share m_T (resp. a_T) is held by every server in $\bar{T} = [n] \setminus T$, since it is derived from the seed $\mathsf{sd}_{i,T}$ shared among S_i and all servers in $\bar{T}$. In particular, each server S_j knows (m_T, a_T) for every T with $j \in \bar{T}$, while only the dealer S_i knows all N pairs. The correction δ is a public broadcast issued by the dealer. Once an arbitrary ordering $T_1, T_2, \ldots, T_N$ of $\mathcal{T}$ is fixed, we write $m_k \coloneqq m_{T_k}$ and $a_k \coloneqq a_{T_k}$ for $k = 1, \ldots, N$. We define C_{dual} as an arithmetic circuit over $\mathbb{F}_p$ in detail as follows.

Input wires. The circuit has $2N$ input wires, partitioned into multiplicative and additive shares:

$$\mathbf{x} = (x_1, x_2, \ldots, x_{2N-1}, x_{2N}) = (M_1, a_1, \ldots, M_N, a_N).$$

Each verifier S_j can independently check the input wires corresponding to subsets T with $j \in \bar{T}$, since it holds those shares locally.

Multiplication gates and wires. The product $\prod_{T \in \mathcal{T}} (m_T)^e$ is computed by a chain of $N-1$ multiplication gates $\{G_\ell^\times\}_{\ell \in [N-1]}$, with running-product wires $w_1, \ldots, w_N$ defined by

$$w_1 = M_1, \qquad w_k = w_{k-1} \cdot M_k \quad \text{for } k = 2, \ldots, N.$$

where $M_k = m_k^e$ for any $k \in [N]$. Gate $G_\ell^\times$ takes left input w_ℓ, right input $M_{\ell+1}$, and produces $w_{\ell+1}$. The intermediate wires $w_2, \ldots, w_{N-1}$ are unknown to the verifier and form part of the extended witness. The final wire $w_N = \prod_{T \in \mathcal{T}} (m_T)^e$ feeds into the output computation.

Addition gates (linear). The sum $\sum_{T \in \mathcal{T}} a_T$ and the final subtraction of δ are computed by addition (and constant-subtraction) gates, which are purely linear and thus "free" in the arithmetic circuit model—they require no multiplicative witness and contribute no soundness cost.

Output wire. The circuit has a single output wire computing:

$$C_{\mathsf{dual}}(\mathbf{w}) = w_N - \sum_{T \in \mathcal{T}} a_T - \delta. \tag{1}$$

The prover must demonstrate that (1) evaluates to zero. Because (1) is linear, it is not proven as a separate constraint but folded into the last multiplication gate: instead of producing a fresh wire w_N, gate G_{N-1} is checked against the linear combination $\sum_{T \in \mathcal{T}} a_T + \delta$ of input wires and the public correction, i.e., it enforces $w_{N-1} \cdot M_N = \sum_{T \in \mathcal{T}} a_T + \delta$. Given the first $N-2$ gates, this holds if and only if (1) evaluates to zero, so δ is bound to the witness rather than only entering the Fiat–Shamir hash.

Extended witness. The full witness for the ZKP consists of the $2N$ input wire values $\mathbf{x}$ together with the $N-2$ intermediate multiplication output wires $(w_2, \ldots, w_{N-1})$, for a total of $L = 3N-2$ witness elements. We denote the extended witness vector by

$$\mathbf{w} \;=\; (M_1,\, a_1,\, \ldots,\, M_N,\, a_N,\, w_2,\, \ldots,\, w_{N-1}) \;\in\; \mathbb{F}_p^L,$$

where the first $2N$ entries are the input wires $\mathbf{x}$ and the remaining $N-2$ entries are the intermediate multiplication output wires. Π_{VitH} commits to $\mathbf{w}$ and proves the $M = N-1$ gate relations $w_{\ell+1} = w_\ell \cdot M_{\ell+1}$ for $\ell = 1, \ldots, N-2$ and $w_{N-1} \cdot M_N = \sum_{T \in \mathcal{T}} a_T + \delta$; Π_{Lig} encodes the same relations with running-product and running-sum rows and enforces (1) as a boundary constraint (Appendix C.2).

5.2.2 Π_{VitH}*: VOLEitH-based Dual-Share ZKP.* We first present an instantiation based on VOLE-in-the-Head (VOLEitH) [7]. VOLEitH achieves post-quantum security from symmetric-key primitives alone (collision-resistant hashing and PRGs), requires no trusted setup, and proves the relation $\delta = \prod_T M_T - \sum_T a_T$. Its proof sizes scale as $O(M \log|\mathbb{F}| + \lambda^2/\log\tau)$ (where $M = N-1$ is the number of multiplication gates and τ is a tree-arity parameter), making it particularly efficient for small-to-moderate N. At a high level, VOLEitH combines three ingredients:

- **QuickSilver commit–challenge–verify pipeline [33].** The prover holds the extended witness $\mathbf{w}$ and a VOLE correlation: for every wire j, values (u_j, v_j) satisfy $q_j = u_j \cdot \Delta + v_j$ for a global secret verifier key Δ. The prover publishes masked wire values $\tilde{w}_j = w_j + u_j$. The verifier checks each multiplication gate by computing a degree-2 expression in Δ from the VOLE-committed wires; when the gate is satisfied, the Δ^2 terms cancel and the check reduces to a linear identity (see Appendix C.1 for the full derivation). All M gate checks are batched via a random Fiat–Shamir challenge χ into a single verification equation. Since VOLE commitments are linearly homomorphic, the last gate is checked against a commitment to $\sum_T a_T + \delta$ computed from the committed input wires, which binds δ (§5.2.1).
- **Non-interactive VOLE from a GGM tree.** Rather than running an interactive OT-based VOLE between the prover and the verifier, VOLEitH [7] generates the correlation locally. A GGM tree of arity τ expands a single root seed into τ leaf seeds via a length-doubling PRG; each leaf i determines a virtual party P_i's random shares $\{s_j^{(i)}\}$ for all wires j, with $u_j = \sum_i s_j^{(i)}$ and $v_j = -\sum_i i \cdot s_j^{(i)}$. The prover commits to the GGM tree (Merkle hash of the leaf nodes) and computes its masked witness and QuickSilver check values; only then does it derive $\Delta \in [\tau]$ by hashing all of them (Fiat–Shamir), and it opens every leaf *except* leaf Δ by revealing the $\lceil \log_2 \tau \rceil$ GGM sibling seeds along the path to Δ. The order matters: a prover that knew Δ before fixing its check values could pass the check for any witness. The batching challenge χ is hashed earlier, from the commitments and the masked witness. The verifier reconstructs all sd_i for $i \neq \Delta$, expands them to shares, and computes the VOLE tags $q_j = \sum_{i \neq \Delta} s_j^{(i)} \cdot (\Delta - i)$. Since the verifier's check is a degree-2 polynomial identity in Δ and Δ ranges over only τ values, a cheating prover can make it hold for two of them, so a single instance has soundness error $2/\tau$.
- **Parallel repetitions for soundness amplification.** The protocol runs R independent repetitions, each with its own GGM tree, fresh VOLE masking, and separate QuickSilver check. The batching challenges χ_ρ are hashed from the commitments and masked witnesses of all repetitions, and all hidden leaves Δ_ρ are hashed together from these and the check values of all repetitions, so no repetition can be ground on its own. Since the hidden leaves come from a set of only τ values, a cheating prover can still re-randomize its commitments and re-hash offline, succeeding with probability about $q_H(2/\tau)^R$ after q_H hash queries. We therefore set $R = \lceil \lambda/(\log_2 \tau - 1) \rceil$ for the computational security parameter λ rather than the statistical κ. The proof is sent in a single round. Over a large prime field $\mathbb{F}_p$ (with $p \gg \tau$), the shares $s_j^{(i)}$ live directly in $\mathbb{F}_p$, so each repetition produces a full VOLE over $\mathbb{F}_p$—no extension-field or subspace-VOLE machinery is needed. The resulting R independent GGM trees can be merged into a single *Batch All-But-One Vector Commitment* (BAVC) to reduce commitment overhead [7].

Input binding via two-level GGM commitment (our contribution). The standard two-party VOLEitH proof by itself establishes only that *some* witness satisfies C_{dual}; it does not bind the witness to the partial witness held by a verifier as in our setting. To close this gap, we use a *two-level Merkle commitment* for the GGM tree. In the standard VOLEitH construction, each leaf commitment is simply $\mathsf{h}_i = \mathsf{H}(\mathsf{sd}_i)$, so rt commits only to the leaf *seeds*, not to the share vectors expanded from them. After all-but-one opening, the verifier uses the co-path sibling hashes to check each opened h_i against rt. But h_Δ commits only to the seed sd_Δ, not to its expanded shares, so the verifier cannot verify any individual share $s_k^{(\Delta)}$ against h_Δ—precisely the check that our distributed-verifier setting requires.

Our variant instead builds, for each GGM leaf $i \in [\tau]$, a *sub-Merkle-tree* over the expanded share vector $\mathbf{s}^{(i)} = (s_0^{(i)}, \ldots, s_L^{(i)})$ derived from sd_i, where $j = 0$ indexes the check wire and $j \in [L]$ the witness wires, giving $\mathsf{h}_i = \mathsf{MT}(\mathbf{s}^{(i)})$; each share is hashed with its position and a λ-bit salt also derived from sd_i, so the hashes of the hidden leaf reveal nothing even about wires with few possible values. The outer Merkle tree over $\{\mathsf{h}_i\}_{i \in [\tau]}$ yields the global commitment rt. Once Δ is derived, the prover opens all leaves except leaf Δ via the co-path. For the hidden leaf Δ, each verifier S_j recovers individual shares at its known input-wire positions k (those with $j \in \bar{T}_k$) as follows:

(1) Compute the VOLE mask $u_k = \tilde{w}_k - w_k^{\mathrm{local}}$, using the masked value $\tilde{w}_k$ from the proof and the locally held share w_k^{local}.
(2) Recover the hidden leaf's share: $s_k^{(\Delta)} = u_k - \sum_{i \neq \Delta} s_k^{(i)}$, where the sum is over all opened leaves.
(3) Verify $s_k^{(\Delta)}$ against the sub-Merkle-tree commitment h_Δ using the salt and authentication path provided by the prover.

Since $|\bar{T}_k| = n - t \geq t+1$, each input-wire share is held by more than t verifiers, so at least one of them is honest. Because rt enters the hash that yields Δ, the prover cannot adapt the hidden leaf's shares to Δ. This two-level structure increases the prover's hashing cost from $O(\tau)$ to $O(\tau L)$, but the per-verifier Merkle cost is only $O(L + \log\tau)$ rather than $O(\tau L)$, keeping verification efficient. A single invocation of Π_{VitH} has broadcast cost $O(\lambda N)$ field elements and per-verifier

P2P cost $O(\lambda \log N)$ hashes, so the t+1 invocations inside Π_{ZKPGen} total $O(t\lambda N)$ broadcast and $O(t\lambda \log N)$ per-verifier P2P, entirely from symmetric-key primitives. The full prove and verify protocols $\Pi_{\mathsf{VitH.Prove}}$ and $\Pi_{\mathsf{VitH.Verify}}$, together with the detailed proof-size and complexity breakdown, are given in Appendix C.1.

Lemma 5.4. *Let $t < n/2$, and model H as a random oracle with 2λ-bit outputs. With a broadcast channel, Π_{VitH} followed by the verdict broadcast of Π_{ZKPGen} securely realizes $\mathcal{F}_{\mathsf{DualShareZKP}}$ with abort against a static malicious adversary that corrupts up to t servers and makes at most q_H oracle queries. The distinguishing advantage is at most*

$$(q_H+1)\left(\left(\frac{2}{\tau}\right)^R + \frac{R(N-1)}{|\mathbb{F}_p|}\right) + O\left(\frac{q_H^2}{2^\lambda}\right) + R\left(\lceil \log_2 \tau \rceil + 1\right)\epsilon_{\mathsf{PRG}},$$

where ϵ_{PRG} is the distinguishing advantage against the PRG that expands the GGM tree and its leaves.

Proof sketch. *Honest dealer.* Given the co-path, the shares of each hidden leaf are pseudorandom, so the masked witness and $\tilde{A}_1$ are uniform and $\tilde{A}_0$ is fixed by the verification equation; the salts hide the hidden leaf's hashes wherever a corrupted server does not hold the share. The simulator samples the hidden leaves, the challenges and these values itself, and programs the oracle. *Corrupted dealer.* The simulator extracts the committed leaves from the adversary's oracle queries, which fixes an effective witness $\tilde{\mathbf{w}} - \mathbf{u}$ before any challenge is derived. Every input wire has an honest holder that checks it against the committed hidden leaf, so the effective witness must carry the honest servers' shares. If the statement is still false, the effective witness violates a gate, and each repetition then passes for at most two of the τ hidden leaves unless its batching challenge is bad. All hidden leaves come from one oracle output, so grinding gains at most the factor $q_H + 1$. The full proof is in Appendix D.3. □

5.2.3 Π_{Lig}*: Ligero-based Dual-Share ZKP.* As an alternative to the VOLEitH-based proof above, we present a second instantiation of $\Pi_{\mathsf{DualShareZKP}}$ based on Ligero [5]. Like Π_{VitH}, it achieves post-quantum security from collision-resistant hashing alone, with no trusted setup, and proves the relation $\delta = \prod_T M_T - \sum_T a_T$ over $\mathcal{C}_{\mathsf{dual}}$. A key advantage of the Ligero approach is its natural support for *multi-instance batching*: a single proof can simultaneously establish $B \geq 1$ independent dual-share consistency relations $\delta^{(b)} = \prod_k M_k^{(b)} - \sum_k a_k^{(b)}$ for $b = 1, \ldots, B$, by laying the instances out side by side in shared rows. The commitment, the consistency polynomial and the sampled points are shared across all instances, so the amortized per-instance cost decreases as B grows. At a high level, Π_{Lig} proceeds in four stages:

(1) Lay out, encode and commit. The prover places the shares $(M_k^{(b)}, a_k^{(b)})$ of c instances, with their running products and sums, side by side in four rows of length $L = cN$; the B instances thus fill $\lceil B/c \rceil$ groups of rows, the last one padded with dummy instances. Each row takes its L values on a subgroup H of size $k' \geq L + 2q$, is filled with uniformly random padding on the rest of H, and is RS-encoded on a larger subgroup $D \supset H$ of size n_c. The prover adds a random mask row per verifier and a random blinding row, and Merkle-hashes the salted columns into $\mathrm{rt}_{\hat{W}}$.

(2) Compose. All constraints are batched with a hashed challenge α into a composition polynomial $C(x) = \mathcal{N}(x)/Z_W(x)$, which is RS-encoded and Merkle-committed together with a random blinding polynomial Z_C.

(3) Consistency and binding polynomials (our contribution). The prover broadcasts a random linear combination u of all committed rows, and sends each verifier S_j a *binding polynomial* q_j: a random combination of the share rows, weighted so that on the witness positions it depends only on S_j's own shares, and masked by S_j's mask row, which vanishes on those positions. It also broadcasts a digest of each q_j.

(4) Sample and open. The prover derives q points of $D \setminus H$ by hashing everything above, including the digests of the binding polynomials, so every verifier can re-derive them. The prover opens the committed columns at each sampled point x and at ηx (the shift used by the transition constraints), together with $C(x)$. Each verifier checks u, the constraint identity $Z_W(x)\,C(x) = \mathcal{N}(x)$ and q_j at the sampled points, and checks q_j against its own shares at the positions it holds. The whole proof is sent in a single round.

Because instances share rows, an opened column has $O(B/c + n)$ entries rather than $O(B + n)$. Choosing c to balance the column length against the row length makes the proof grow with $\sqrt{B}$ rather than with B, Ligero's usual square-root regime. The padding makes every opened column uniformly random, so zero knowledge does not limit q. Soundness follows from Ligero's proximity lemma together with root counting on the error polynomials, and the binding polynomials tie the proven witness to the shares held by the distributed verifiers. Because the sampled points are hashed from a small set, q and n_c are sized for the computational security parameter λ. The layout, the soundness bound and parameter choice, the prove and verify protocols, and the complexity analysis are given in Appendix C.2.

Lemma 5.5. *Let $t < n/2$, model H as a random oracle with 2λ-bit outputs, and let the layout satisfy $k' \geq L + 2q$ and $4e < T - K^* + 1$ for $T = n_c - k'$. With a broadcast channel, Π_{Lig} over B instances, followed by the verdict broadcast of Π_{ZKPGen}, securely realizes with abort the functionality that runs $\mathcal{F}_{\mathsf{DualShareZKP}}$ on each instance and accepts only if every instance accepts, against a static malicious adversary that corrupts up to t servers and makes at most q_H oracle queries. The distinguishing advantage is at most*

$$(q_H+1)\left(\left(1-\frac{e}{T}\right)^q + \left(\frac{2K^*+c-2+2e}{T}\right)^q + O\left(\frac{n_c+\ell+n}{|\mathbb{F}_p|}\right)\right) + O\left(\frac{q_H^2}{2^\lambda}\right).$$

Proof sketch. *Honest dealer.* Each row has at least $2q$ uniform padding values on H, so its values at the at most $2q$ opened points outside H are uniform. The blinding rows Z_0 and Z_C make u uniform subject to the values that the openings determine, and the mask rows do the same for the corrupted verifiers' binding polynomials. The salts, and the entropy left in an honest verifier's q_j, make every unopened hash simulatable by programming the oracle. *Corrupted dealer.* The simulator extracts the committed matrix from the oracle queries. If the matrix is far from the code, then by Ligero's proximity lemma u disagrees with the committed combination at more than e of the T candidate points, and the sampled points avoid all of them with probability at most $(1 - e/T)^q$. Otherwise its rows decode

to polynomials that encode an effective witness, fixed before any challenge. If that witness violates a constraint or differs from an honest verifier's shares, the constraint identity or that verifier's binding identity is a nonzero polynomial of degree at most $2K^*+c-2$, which a sampled point satisfies only if it is a root or lies among the at most e corrupted columns. Every message the checks depend on enters the hash of the sampled points, so grinding gains at most the factor $q_H + 1$. The full proof is in Appendix D.4. □

Corollary 5.6. *Instantiate $\mathcal{F}_{\mathsf{DualShareZKP}}$ in Π_{ZKPGen} with Π_{VitH} or with Π_{Lig}, prefixing every hash input with the dealer's index and a session identifier. In the random-oracle model, the resulting protocol securely realizes $\mathcal{F}_{\mathsf{RandExp}}$ under the conditions of Theorem 5.3, with error at most that of Theorem 5.3 plus $t+1$ times the bound of Lemma 5.4 or Lemma 5.5, respectively.*

The prefixes give each dealer's invocation its own oracle domain, so the simulators of the $t+1$ invocations program disjoint points and compose.

6 Online Phase: Verifiable dOPRF Evaluation

In the online phase, we assume servers hold $[\![k]\!]$ and $[\![\alpha^e]\!]$, obtained from $\mathcal{F}_{\mathsf{RandExp}}$, and take in $[\![x]\!]$ from the client C. The goal is for C to receive $\alpha^e(x+k)$ in the presence of $t < n/2$ corrupted servers.

Naive Idea. A straightforward way to compute $[\![ab]\!]$ is to employ a semi-honest secure multiplication protocol, such as the DN protocol [16], together with Boyle et al.'s DZKP protocol that proves the correctness of multiplication [11]; we denote this baseline by $\Pi^m_{\text{naive-dVOPRF}}$. However, it requires 13 rounds and blows up the communication cost as the number of client inputs increases. We propose a more efficient solution.

Our Approach and Challenges. Consider two additive sharings of $a = \sum_{j=1}^{N}[a]_j$ and $b = \sum_{j=1}^{N}[b]_j$, where $N = \binom{n}{t}$. Expanding the product gives $ab = ([a]_1 + \cdots + [a]_N)([b]_1 + \cdots + [b]_N) = \sum_{u=1}^{N}\sum_{v=1}^{N}[a]_u \cdot [b]_v$. Another straightforward way to compute $[\![ab]\!]$ would be to convert each additive component, $[a]_u, [b]_v$, into RSS form and then invoke an interactive secure multiplication protocol [16, 23] to compute RSS shares of each term $[a]_u \cdot [b]_v$ of ab. However, this approach requires N^2 multiplications and is prohibitively inefficient. Instead, we partition the N^2 product terms into n disjoint collections, one per server, so that each server can evaluate its assigned terms *locally* from its RSS view. Each server S_i computes a local partial sum c_i of the form $c_i = \sum_{(u,v)\in\Lambda_i}[a]_u \cdot [b]_v$ for an assignment $\{\Lambda_i\}_{i\in[n]}$ that we construct in §6.1, and then reshapes c_i into an RSS sharing. The desired RSS shares $[\![ab]\!]$ can be locally computed by $[\![c_1]\!] + \cdots + [\![c_n]\!]$. This eliminates interactive multiplication altogether, while still producing the correct multiplication result in shared form. Therefore, the challenges for this approach are: (1) *how we assign RSS shares to servers accordingly;* and (2) *how we efficiently make a protocol secure against up to $t < n/2$ malicious servers.*

6.1 Share Assignments

Our goal is to obtain an additive sharing of the product ab using local multiplications only, i.e., without interactive multiplication protocols. In this section, we propose a (balanced) share-assignment protocol for RSS multiplications that partitions the N^2 product terms among n servers.

By definition of RSS, server S_i holds $[\![a]\!]_i = \{[a]_u \mid i \notin T_u \in \mathcal{T}\}$ and $[\![b]\!]_i = \{[b]_v \mid i \notin T_v \in \mathcal{T}\}$. Hence, S_i can locally compute the term $[a]_u \cdot [b]_v$ if and only if $i \notin T_u \cup T_v$. Define the set of *eligible servers* for term (u,v) as $\mathcal{E}_{u,v} \coloneqq [n] \setminus (T_u \cup T_v)$. Assuming $n = 2t+1$, we have $|\mathcal{E}_{u,v}| = n - |T_u \cup T_v| = n - (2t - |T_u \cap T_v|) = |T_u \cap T_v| + 1$, and in particular $1 \le |\mathcal{E}_{u,v}| \le t+1$ for all $u, v \in [N]$, so every product term is computable by at least one server.

We now compute a balanced assignment $\{\Lambda_i\}_{i\in[n]}$ such that (i) $(u,v) \in \Lambda_i$ implies $i \in \mathcal{E}_{u,v}$ for local computability, and (ii) each server computes the same number of terms. Since there are N^2 terms in total and N^2 is divisible by n in our setting, each server is assigned exactly $L \coloneqq N^2/n$ terms. We compute such an allocation by reducing it to a (weighted) bipartite b-matching and max-flow instance. First, consider a bipartite graph in which left vertices represent pair indices $(u,v) \in [N]^2$, right vertices represent servers $i \in [n]$, and each left vertex (u,v) has edges to right vertex i with weight 1 if and only if $i \in \mathcal{E}_{u,v}$. We impose the degree bounds $b((u,v)) = 1$ on each left vertex and $b(i) = L$ on each right vertex. Then, construct a directed flow network based on the bipartite graph as follows: a source node n_s has edges to each left node (u,v) with capacity 1, a sink node n_t has incoming edges with capacity L from each right node. Any integral max flow of value $N^2 = nL$ yields a balanced assignment: term (u,v) is assigned to the unique server i with one unit of flow on edge $(u,v) \to i$. This guarantees that each server S_i can compute its assigned terms locally since $i \in \mathcal{E}_{u,v}$, and that each server receives exactly L terms. The seminal Ford–Fulkerson algorithm [20] (or more efficient algorithms such as [22]) computes the max flow in polynomial time in N.

6.2 A Semi-Honest Construction

Given the above assignment, our basic construction, which works in a semi-honest security setting, is as follows. Each server S_i locally computes its partial sum $c_i \coloneqq \sum_{(u,v)\in\Lambda_i}[a]_u \cdot [b]_v$ using only its local RSS components $[\![a]\!]_i$ and $[\![b]\!]_i$. The servers then distribute RSS sharings of these values, so that each server obtains $[\![c_1]\!]_i, \ldots, [\![c_n]\!]_i$ and computes $[\![c]\!]_i \coloneqq \sum_{j=1}^{n}[\![c_j]\!]_i$, where $c = \sum_{j=1}^{n} c_j = ab = \alpha^e(x+k)$. Finally, the client reconstructs c from the received RSS shares and obtains a PRF value $c^g \bmod p = (x+k)^g \bmod p$.

6.3 Verifiable Inner Product

To upgrade the above protocol to malicious security, we build on the DZKP techniques for degree-2 relations of Boyle et al. [11]. In particular, we introduce a core sub-protocol, which we call the *verifiable inner product* (VIP) protocol.[2] For each server S_i, the VIP protocol outputs not only an RSS sharing $[\![c_i]\!]$, but also RSS sharings of four auxiliary elements d_i, a_i, b_i, σ_i used to certify the correctness of $[\![c_i]\!]$. At a high level, the L multiplication terms contributing to c_i are *compressed* into three values (d_i, a_i, b_i) that are required to satisfy the multiplicative relation $d_i = a_i b_i$. Intuitively, this compression binds S_i to a single well-defined computation: if S_i attempts to inconsistently encode the underlying L terms while

[2]This protocol can be viewed as computing a shared inner product $c_i = \langle \vec{a}_i, \vec{b}_i \rangle$ for $\vec{a}_i = (a_i^{(1)}, \ldots, a_i^{(L)})$ and $\vec{b}_i = (b_i^{(1)}, \ldots, b_i^{(L)})$, together with additional values used to (later) check the validity of the computation; therefore we name it the VIP protocol.

still claiming a valid relation, then the equality $d_i = a_i b_i$ fails with high probability by the Schwartz–Zippel lemma. The additional check value σ_i enforces that the claimed inner product value c_i exactly equals $\sum_{(u,v)\in\Lambda_i} [a]_u \cdot [b]_v$; in particular, $\sigma_i = 0$ if and only if c_i is consistent with the L terms. Consequently, if the protocol is executed honestly, then both checks hold, i.e., $d_i = a_i b_i$ and $\sigma_i = 0$, whereas if up to t servers (possibly including S_i) deviate arbitrarily, then at least one of these conditions fails.

We begin with the formal description of the VIP protocol Π_{VIP} in Protocol 5, which computes RSS sharing of $c_i := \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$, where $\{(a_i^{(\ell)}, b_i^{(\ell)})\}_{\ell\in[L]} := \{([a]_u, [b]_v) \in [\![a]\!]_i \times [\![b]\!]_i\}_{(u,v)\in\Lambda_i}$ (namely, c_i consists only of additive shares that S_i holds), together with RSS sharings of the corresponding auxiliary proof elements $(d_i^{(\gamma)}, a_i^{(1)}, b_i^{(1)}, \sigma_i)$.

This protocol requires all servers to hold RSS shares of all $a_i^{(\ell)}$ and all $b_i^{(\ell)}$, which can be locally computed by the local share conversion [14], since, by the RSS property of $[\![a]\!]$ and $[\![b]\!]$, each of $a_i^{(\ell)}$ $(= [a]_u)$ and $b_i^{(\ell)}$ $(= [b]_v)$ is shared among $t+1$ servers.

Suppose L is even.[3] For $j \in [L]$ and $\ell \in [2]$, define linear polynomials $f_1, \ldots, f_L$ and a quadratic polynomial $q^{(1)}$ as:

$$f_j(\ell) = \begin{cases} a_i^{(\frac{L}{2}\cdot(\ell \bmod 2)+\lceil \frac{j}{2} \rceil)} & \text{if } j \text{ is odd, i.e., } j \bmod 2 = 1, \\ b_i^{(\frac{L}{2}\cdot(\ell \bmod 2)+\frac{j}{2})} & \text{if } j \text{ is even, i.e., } j \bmod 2 = 0, \end{cases} \tag{2}$$

$$q^{(1)}(x) = \sum_{\ell=1}^{L/2} f_{2\ell-1}(x) \cdot f_{2\ell}(x). \tag{3}$$

That is, each $f_j(x)$ contains every 2 elements of $2L$ inputs $\{(a_i^{(\ell)}, b_i^{(\ell)})\}_{\ell\in[L]}$, and therefore the polynomial $q^{(1)}(x)$ can be seen as a folded one of $q^{(0)}(x)$, which is a polynomial such that $q^{(0)}(1) = c_i$ (and $q^{(0)}(2) = 0$). S_i then evaluates random points $f_1(r_1), \ldots, f_L(r_1)$, $q^{(1)}(r_1)$ with randomness $r_1 \in \mathbb{F}_p \setminus \{1,2,3\}$; since these polynomials are fixed before r_1 is drawn, the Schwartz–Zippel lemma binds S_i to the 2 elements embedded in each of $f_1, \ldots, f_L$ and to the folded relation between $q^{(1)}$ and $q^{(0)}$. In such a way, S_i recursively folds polynomials γ $(:= \lceil \log L \rceil)$ times in total. In the k-th iteration, S_i sets $a_i^{(\ell)} \leftarrow f_{2\ell-1}(r_{k-1})$ and $b_i^{(\ell)} \leftarrow f_{2\ell}(r_{k-1})$ for $\ell \in [L/2^{k-1}]$, redefines $f_1, \ldots, f_{L/2^{k-1}}$, and folds the previous polynomial $q^{(k-1)}(x)$ to get a folded polynomial $q^{(k)}(x)$. After the γ iterations, S_i only holds $a_i^{(1)}$ $(:= f_1(r_\gamma))$, $b_i^{(1)}$ $(:= f_2(r_\gamma))$, and $d_i^{(\gamma)}$ $(:= q^{(\gamma)}(r_\gamma))$; below we write d_i, a_i and b_i for $d_i^{(\gamma)}$, $a_i^{(1)}$ and $b_i^{(1)}$, which still ensures that $d_i^{(\gamma)} = a_i^{(1)} \cdot b_i^{(1)}$ holds if S_i proceeds correctly; $d_i^{(\gamma)} \neq a_i^{(1)} \cdot b_i^{(1)}$ holds otherwise.

In addition, we must verify the relation between adjacent iterations, i.e., $q^{(k-1)}(r_{k-1}) \stackrel{?}{=} q^{(k)}(1) + q^{(k)}(2)$ for every $k \in [\gamma]$. To this end, S_i computes $\sigma_i := \sum_{k=2}^{\gamma} \epsilon_k (q^{(k-1)}(r_{k-1}) - q^{(k)}(1) - q^{(k)}(2))$, where $\epsilon_2, \ldots, \epsilon_\gamma$ are random field elements; $\sigma_i = 0$ holds if and only if S_i honestly connects all adjacent iterations.[4] All servers except for S_i can follow the same procedures by using their RSS shares $\{([\![a_i^{(\ell)}]\!], [\![b_i^{(\ell)}]\!])\}_{\ell\in[L]}$, which is derivable by the local conversion,

[3] If L is odd, one may just set $L \leftarrow L+1$ and $a_i^{(L)} = b_i^{(L)} = 0$.

[4] c_i itself ensures that $c_i - q^{(1)}(1) - q^{(1)}(2) = 0$ so we can ignore the case of $k = 1$.

Protocol 5 Π_{VIP} – Verifiable Computation for Inner Products over RSS in the $\mathcal{F}_{\mathsf{Coin}}$-Hybrid Model

Players: Servers $S_1, \ldots, S_n$, where S_i is a dealer.

Input: Servers hold $\{[\![a_i^{(\ell)}]\!], [\![b_i^{(\ell)}]\!]\}_{\ell\in[L]}$ where S_i knows $\{a_i^{(\ell)}, b_i^{(\ell)}\}_{\ell\in[L]}$.

1: Let $\gamma \leftarrow \lceil \log L \rceil$.
2: **for** $k = 1, \ldots, \gamma$ **do**
3: If L is odd, let $L \leftarrow L+1$.
4: Servers define (or update) polynomials $f_1, f_2, \ldots, f_L$, and $q^{(k)}$ as defined in Eqs. (2) and (3).
5: S_i shares $q^{(k)}(1), q^{(k)}(2), q^{(k)}(3)$ to all servers using VSS.
6: **if** $k = 1$ **then**
7: Servers locally compute $[\![c_i]\!] \leftarrow [\![q^{(k)}(1)]\!] + [\![q^{(k)}(2)]\!]$.
8: **end**
9: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $r_k \stackrel{\$}{\leftarrow} \mathbb{F}_p \setminus \{1,2,3\}$.
10: S_i sets $\{a_i^{(\ell)}, b_i^{(\ell)}\} \leftarrow \{f_{2\ell-1}(r_k), f_{2\ell}(r_k)\}$ for all $\ell \in [L/2]$, and $d_i^{(k)} \leftarrow q^{(k)}(r_k)$.
11: Servers use Lagrange interpolation to locally compute $\{[\![a_i^{(\ell)}]\!], [\![b_i^{(\ell)}]\!]\} \leftarrow \{[\![f_{2\ell-1}(r_k)]\!], [\![f_{2\ell}(r_k)]\!]\}$ for all $\ell \in [L/2]$ and $[\![d_i^{(k)}]\!] \leftarrow [\![q^{(k)}(r_k)]\!]$, and set $L \leftarrow L/2$.
12: **end**
13: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\epsilon_2, \epsilon_3, \ldots, \epsilon_\gamma \stackrel{\$}{\leftarrow} \mathbb{F}_p$ and locally update $[\![\sigma_i]\!] \leftarrow \sum_{k=2}^{\gamma} \epsilon_k ([\![d_i^{(k-1)}]\!] - [\![q^{(k)}(1)]\!] - [\![q^{(k)}(2)]\!])$.

Output: Servers record $([\![d_i^{(\gamma)}]\!], [\![a_i^{(1)}]\!], [\![b_i^{(1)}]\!], [\![\sigma_i]\!], [\![c_i]\!])$.

and $\{[\![q^{(k)}]\!]\}_{k\in[\gamma]}$ sent by S_i, and obtain $[\![d_i^{(\gamma)}]\!]$, $[\![a_i^{(1)}]\!]$, $[\![b_i^{(1)}]\!]$, $[\![\sigma_i]\!]$, and $[\![c_i]\!]$ $(= [\![q^{(1)}(1)]\!] + [\![q^{(1)}(2)]\!])$.

Unlike in Boyle et al.'s setting, neither the client nor the servers open the output, particularly $a_i^{(1)}$ and $b_i^{(1)}$, in the current form during the dVOPRF protocol; consequently, we do not require the additional output-randomization step used in Boyle et al.'s protocol [11], where openings would otherwise reveal linear relations among inputs. The step is reintroduced in §6.4, where $U(\rho)$ and $V(\rho)$ are opened to the client and must therefore be masked by the random a_0 and b_0.

Lemma 6.1. *In the $\mathcal{F}_{\mathsf{Coin}}$-hybrid model, Π_{VIP} (Protocol 5) satisfies the following properties.*

(1) ***Correctness.*** *If the sharing server S_i is honest, then the protocol outputs consistent secret sharings $[\![c_i]\!], [\![d_i^{(\gamma)}]\!], [\![a_i^{(1)}]\!], [\![b_i^{(1)}]\!], [\![\sigma_i]\!]$ of values satisfying $c_i = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$, $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$, and $\sigma_i = 0$.*
(2) ***Soundness.*** *If the sharing server S_i is corrupted, then except with probability at most $\varepsilon_{\mathsf{VIP}} = (2\lceil \log L \rceil + 1)/(|\mathbb{F}_p| - 3)$, either the protocol aborts, or the protocol outputs consistent secret sharings $[\![c_i]\!], [\![d_i^{(\gamma)}]\!], [\![a_i^{(1)}]\!], [\![b_i^{(1)}]\!], [\![\sigma_i]\!]$ such that, if the reconstructed values satisfy $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$ and $\sigma_i = 0$, then necessarily $c_i = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$.*

We prove Lemma 6.1 in Appendix D.5.

Protocol 6 $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ – Batched Verifiable Computation for Inner Products in the $(\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{RSS.Mul}})$-Hybrid Model

Players: Servers $S_1, \ldots, S_n$.

Input: Each server S_i holds $\{a_i^{(\ell)}, b_i^{(\ell)}\}_{\ell\in[L]}$ and $\{[\![a_j^{(\ell)}]\!], [\![b_j^{(\ell)}]\!]\}_{\ell\in[L]}$ for all $j \in [n]$.

1: Run n instances of Π_{VIP} with the servers' shares in parallel, where S_i acts as the dealer in the i-th instance. All instances share the same outputs $(r_k, \epsilon_k)_{k\in[\lceil \log L\rceil]}$ of $\mathcal{F}_{\mathsf{Coin}}$.
2: Each instance i produces $([\![d_i]\!], [\![a_i]\!], [\![b_i]\!], [\![\sigma_i]\!], [\![c_i]\!])$.
3: Servers locally compute $[\![c]\!] = \sum_{i=1}^n [\![c_i]\!]$.
4: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\{\epsilon'_i\}_{i\in[n]}$, and then locally compute $[\![\Sigma]\!] = \sum_{i=1}^n \epsilon'_i \cdot [\![\sigma_i]\!]$.
5: Servers call $\mathcal{F}_{\mathsf{Rand}}$ to obtain $[\![a_0]\!], [\![b_0]\!]$.
6: Servers locally compute $[\![U(x)]\!]$ and $[\![V(x)]\!]$ of degree n by interpolating through $\{(0, [\![a_0]\!])\} \cup \{(i, [\![a_i]\!])\}_{i\in[n]}$ and $\{(0, [\![b_0]\!])\} \cup \{(i, [\![b_i]\!])\}_{i\in[n]}$, respectively.
7: **for** $i \in \{0, n+1, \ldots, 2n\}$ **do**
8: | Servers call $\mathcal{F}_{\mathsf{RSS.Mul}}$ to get $[\![W(i)]\!] = [\![U(i) \cdot V(i)]\!]$.
9: **end**
10: Servers locally compute $[\![W(x)]\!]$ of degree $2n$ by interpolating via $\{(i, [\![W(i)]\!])\}_{i\in\{0,\ldots,2n\}}$, where $[\![W(i)]\!] \leftarrow [\![d_i]\!]$ for $i \in [n]$.
11: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\rho \xleftarrow{\$} \mathbb{F}_p \setminus \{0, 1, \ldots, 2n\}$, then locally compute $[\![U(\rho)]\!], [\![V(\rho)]\!], [\![W(\rho)]\!]$.

Output: Servers hold $([\![W(\rho)]\!], [\![U(\rho)]\!], [\![V(\rho)]\!], [\![\Sigma]\!], [\![c]\!])$.

6.4 Batched Verifiable Inner Product

At a high level, our dVOPRF protocol proceeds as follows. First, the servers execute n instances of the VIP protocol Π_{VIP} in parallel (one instance per server) in order to obtain RSS sharings of the partial products $\{[\![c_i]\!]\}_{i\in[n]}$, together with the auxiliary proofs $([\![d_i]\!], [\![a_i]\!], [\![b_i]\!], [\![\sigma_i]\!])_{i\in[n]}$ required for verification. Second, the servers batch the auxiliary values into a single tuple of aggregated proofs of correctness while applying the output-randomization step described above. Third, the client (robustly) reconstructs the product c from $[\![c]\!]$ $(= \sum_{i=1}^n [\![c_i]\!])$ if verification with aggregated proofs succeeds; the client then computes $c^g \bmod p = (x+k)^g \bmod p$.

We encapsulate the first two steps into a single *batched VIP* protocol, denoted $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ (Protocol 6). Here, we elaborate on the second step: the batching process. With the proof values $([\![d_i]\!], [\![a_i]\!], [\![b_i]\!])_{i\in[n]}$ for proving the folding relation, servers also compress them into single polynomials as follows: $U(x)$ and $V(x)$ are polynomials such that $U(i) = a_i$ and $V(i) = b_i$ for all $i \in \{0, 1, \ldots, n\}$, where a_0 and b_0 are random values to randomize the polynomials, which we call the output-randomization step mentioned above. This randomization is necessary to prevent $U(x)$ and $V(x)$ from leaking information on the servers' inputs. Therefore, servers can interpolate $[\![U(x)]\!]$ and $[\![V(x)]\!]$ from $\{[\![a_i]\!], [\![b_i]\!]\}_{i\in[n]}$ and the random shares $[\![a_0]\!], [\![b_0]\!]$. Let $W(x)$ be a degree-$2n$ polynomial such that $W(x) = U(x) \cdot V(x)$, so that $W(i) = d_i$ for all $i \in [n]$. Since servers already have (shares of) n evaluations $\{[\![d_i]\!]\}_{i\in[n]}$, they additionally execute secure multiplication protocols to obtain $[\![W(i)]\!]$ from $[\![U(i)]\!]$ and $[\![V(i)]\!]$ for $i = 0, n+1, \ldots, 2n$, enabling them to interpolate $[\![W(x)]\!]$.

Together with the last step, i.e., the client-side robust opening procedure, $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ yields a dVOPRF protocol for a *single* input that is maliciously secure against up to $t < n/2$ corrupted servers.

Lemma 6.2. *In the $(\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{RSS.Mul}})$-hybrid model, $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ (Protocol 6) satisfies the following properties. Let $\varepsilon_{\mathsf{VIP}} \coloneqq (2\lceil \log L\rceil + 1)/(|\mathbb{F}_p| - 3)$, $C_i^* \coloneqq \sum_{\ell=1}^L a_i^{(\ell)} b_i^{(\ell)}$ and $C^* \coloneqq \sum_{i=1}^n C_i^*$. We say that an output tuple $(c, \Sigma, W(\rho), U(\rho), V(\rho))$ is* accepting *if it satisfies $\Sigma = 0$ and $W(\rho) = U(\rho)V(\rho)$.*

*(1) **Correctness.** If all underlying Π_{VIP} instances are executed honestly, then the protocol outputs consistent secret sharings of values satisfying $c = C^*$, and the corresponding output tuple is accepting.*

*(2) **Designated-verifier privacy.** Let $\mathsf{View}^{\mathsf{VIP}}_{\mathsf{corr}}$ denote the joint view of the corrupted sharing servers in the underlying Π_{VIP} instances. The opened tuple $(U(\rho), V(\rho), W(\rho), \Sigma, c)$ is statistically simulatable from c, the public coins, and $\mathsf{View}^{\mathsf{VIP}}_{\mathsf{corr}}$. In particular, if the output tuple is accepting, then the opened tuple is simulatable from c and the fact that the tuple is accepting.*

*(3) **Soundness.** For any adversary corrupting at most $t < n/2$ servers, except with probability at most $\varepsilon^{\mathsf{Prl}}_{\mathsf{VIP}} = n\varepsilon_{\mathsf{VIP}} + 2n/(|\mathbb{F}_p| - 2n - 1) + 1/|\mathbb{F}_p|$, either the protocol aborts, or the output sharings are consistent and satisfy the following implication: if the output tuple is accepting, then $c = C^*$.*

We prove Lemma 6.2 in Appendix D.6.

6.5 Verifiable Client Input

Our online protocol requires the client to hand the servers an RSS sharing $[\![x]\!]$ of its evaluation point. A *direct* sharing—the client generates $[\![x]\!]$ itself and sends each S_i its list $[\![x]\!]_i$—is not verifiable: a malicious client can send mismatched copies of a replicated component $[x]_\ell$ to two servers that both hold it, and nothing detects the inconsistency. We therefore present Π_{Input} (Protocol 7), which realizes $\mathcal{F}_{\mathsf{RSS.Share}}$ with abort for a malicious *client*, at strictly lower communication cost than the direct sharing except for $(n, t) = (3, 1)$.

Π_{Input} follows the mask-and-broadcast pattern: the servers hold a pre-computed RSS sharing $[\![r]\!]$ of a uniform r, open r to the client, and the client broadcasts $u \coloneqq r + x$, from which every server derives $[\![x]\!] \coloneqq u - [\![r]\!]$ locally. Two details make it verifiable. *First*, r is opened *with abort* rather than by plain reconstruction: each server also sends a hash of its full local list, and the client—having assembled every additive component—recomputes each server's list and checks it, exactly as in $\Pi_{\mathsf{RSS.Open}}$. Without this check a single corrupted server could send a wrong component, the client would reconstruct some $r' \neq r$, and the servers would end up holding $[\![x + (r - r')]\!]$: an honest client's input silently shifted by the adversary. *Second*, u must reach *all* n servers identically; the servers therefore echo a hash of u and abort on disagreement, which is what forces a malicious client to commit to a single input. When Π_{Input} is composed into Π^m_{dVOPRF}, this echo rides the hash exchange that protocol already performs, and so costs no additional round.

Each $[\![r^{(j)}]\!]$ must be consumed by *exactly one* input: reusing a mask across $x^{(1)}$ and $x^{(2)}$ reveals $u^{(1)} - u^{(2)} = x^{(1)} - x^{(2)}$ to every server.

Standalone, Protocol 7 requires 3 rounds; composed into Π^m_{dVOPRF} the echo of Step 4 is merged into an existing hash exchange, leaving 2. Because every additive component is sent by exactly one designated server, C receives mN field elements and n hash values in total, and sends m field elements to each server. A direct sharing

Protocol 7 Π_{Input} – Verifiable (t, n)-RSS Sharing of an Input

Players: Client C and servers $S_1, \ldots, S_n$.

Input: C holds $(x^{(1)}, \ldots, x^{(m)}) \in \mathbb{F}_p^m$; servers hold pre-computed RSS sharings $[\![r^{(1)}]\!], \ldots, [\![r^{(m)}]\!]$ of independent uniform $r^{(j)} \in \mathbb{F}_p$, each used exactly once. A public assignment $P\colon [N] \to [n]$ with $P(\ell) \notin T_\ell$ for every $\ell \in [N]$.

1: Each server S_i sends $\{[r^{(j)}]_\ell \mid j \in [m],\ \ell \in [N],\ P(\ell) = i\}$ and $\psi_i \leftarrow H([\![r^{(1)}]\!]_i \| \cdots \| [\![r^{(m)}]\!]_i)$ to C.

2: C assembles $[r^{(j)}]_\ell$ for all $j \in [m]$, $\ell \in [N]$, recomputes each server's local list $[\![r^{(j)}]\!]_i = \{[r^{(j)}]_\ell \mid i \notin T_\ell\}$, and verifies $\psi_i \stackrel{?}{=} H([\![r^{(1)}]\!]_i \| \cdots \| [\![r^{(m)}]\!]_i)$ for all $i \in [n]$. `// abort and output ⊥ if any check fails`

3: C computes $r^{(j)} \leftarrow \sum_{\ell=1}^{N} [r^{(j)}]_\ell$ and $u^{(j)} \leftarrow r^{(j)} + x^{(j)}$ for every $j \in [m]$, and sends $(u^{(1)}, \ldots, u^{(m)})$ to *all* n servers.

4: Each server S_i computes $\chi_i \leftarrow H(u^{(1)} \| \cdots \| u^{(m)})$ from the values it received and sends χ_i to each $S_{i'}$ for $i' \in [n] \setminus \{i\}$. `// abort if` $\chi_{i'} \neq \chi_i$ `for some` i'

5: Each server S_i locally sets $[\![x^{(j)}]\!]_i \leftarrow u^{(j)} - [\![r^{(j)}]\!]_i$ for every $j \in [m]$.

Output: Each server S_i holds $[\![x^{(j)}]\!]_i$ for every $j \in [m]$; or the honest parties output $\perp$.

instead costs mnN' field elements from C; since $N' = \binom{n-1}{t}$ grows combinatorially in n while the client now uploads a single element per server per input, compared with the client naively handling Replicated Secret Shares to servers, Π_{Input} is cheaper for every (n, t) beyond $(3, 1)$ and increasingly so as n grows.

Lemma 6.3. *Let $t < n/2$ and let H be collision resistant. In the presence of a static adversary corrupting up to t servers and, possibly, the client C, Protocol 7 satisfies the following.*

(1) (Consistency) *If the honest servers do not abort, then for every $j \in [m]$ there is a unique $x^{(j)} \in \mathbb{F}_p$ such that the honest servers' outputs are consistent components of a single RSS sharing $[\![x^{(j)}]\!]$ of $x^{(j)}$.*

(2) (Input integrity) *If moreover C is honest and does not abort, then $x^{(j)}$ is exactly the input C supplied.*

(3) (Extractability) *There is an efficient extractor that, given the honest servers' views, outputs the $(x^{(1)}, \ldots, x^{(m)})$ of item 1.*

(4) (Privacy) *If C is honest, the adversary's view is independent of $(x^{(1)}, \ldots, x^{(m)})$.*

Proof. *Consistency.* Suppose no honest server aborts at Step 4. Every honest server compares the full vector $(\chi_1, \ldots, \chi_n)$ against its own value, so non-abort means all these hashes agree; by collision resistance of H, all honest servers therefore hold the same $(u^{(1)}, \ldots, u^{(m)})$. By assumption the servers entered the protocol holding consistent components of a single RSS sharing $[\![r^{(j)}]\!]$. Step 5 applies the same public affine map $z \mapsto u^{(j)} - z$ at every honest server, so the resulting shares are consistent components of the RSS sharing of $x^{(j)} \coloneqq u^{(j)} - r^{(j)}$, which is uniquely determined.

Input integrity. Suppose C is honest and passes Step 2, and suppose for contradiction that some corrupted designated sender $S_{P(\ell)}$ sent $\widetilde{[r^{(j)}]_\ell} \neq [r^{(j)}]_\ell$. The component indexed by ℓ is held by the $n - t \geq t + 1$ servers in $[n] \setminus T_\ell$, of which at most t are corrupted; let S_{i^*} be an honest one. Then S_{i^*} sent ψ_{i^*} computed over its true local lists, whereas the list C recomputes for S_{i^*} contains $\widetilde{[r^{(j)}]_\ell}$ and hence differs. By collision resistance the check for i^* fails and C aborts, a contradiction. Hence C reconstructs the true $r^{(j)}$, and $u^{(j)} - r^{(j)}$ is the input C supplied.

Extractability. The honest servers hold $[\![r^{(j)}]\!]_i$ for $n - t \geq t + 1$ distinct i, which together cover all N additive components; the extractor reconstructs $r^{(j)}$ from them, reads $u^{(j)}$ off any honest server's Step-3 transcript, and outputs $u^{(j)} - r^{(j)}$.

Privacy. The adversary holds at most t RSS shares of $r^{(j)}$, which by the RSS access structure omit at least one additive component $[r^{(j)}]_\ell$. That component is uniform and independent of the adversary's view, so $r^{(j)}$ is uniform in that view and $u^{(j)} = r^{(j)} + x^{(j)}$ is uniform and independent of $x^{(j)}$. As each mask is consumed by exactly one input, this holds jointly over $j \in [m]$. The remaining messages the adversary observes, $\{\psi_i\}$ and $\{\chi_i\}$, are functions of $[\![r^{(j)}]\!]$ and $(u^{(j)})_{j \in [m]}$ alone. □

6.6 The Full Protocol

Putting the pieces together, our maliciously secure dVOPRF protocol proceeds as follows. Each server S_i runs the VIP protocols to share S_i's terms c_i with other servers, and obtains $[\![c]\!]_i = [\![c_1]\!]_i + \cdots + [\![c_n]\!]_i$ and their proof elements. Servers batch those proofs and send them along with $[\![c]\!]$ to the client C, thereby allowing C to securely open $[\![c]\!]$ by checking the validity of the computation with the proofs.

We are ready to describe our final protocol Π^m_{dVOPRF} (Protocol 8), a more efficient variant that supports m client inputs $(x^{(1)}, \ldots, x^{(m)})$ and amortizes the verification and communication costs across multiple evaluations. Briefly speaking, servers run the batched VIP protocol $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ once for *batched* product $c = \sum_{j=1}^{m} \epsilon_j c^{(j)}$, where $c^{(j)} = (\alpha^{(j)})^e \cdot (x^{(j)} + k)$ and ϵ_j is randomness synchronously and securely generated by using random oracles. Here c is no longer the dVOPRF output; together with the other outputs of $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$, it forms part of the proof that the batched computation was performed correctly. Thus, each server S_i additionally prepares its local partial sums $c_i^{(1)}, \ldots, c_i^{(m)}$ to have the client reconstruct the products $c^{(j)} = \sum_{i=1}^{n} c_i^{(j)}$. Although the partial sums $\{c_i^{(j)}\}_{i \in [n], j \in [m]}$ can be seen as additive shares and hence have no robustness, the batched proofs $([\![W]\!], [\![U]\!], [\![V]\!], [\![\Sigma]\!], [\![c]\!])$ generated by $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ guarantee the correctness of computations for all $c^{(j)}$.

A remaining concern is that each partial sum $c_i^{(j)}$ would leak information about S_i's inputs. To resolve this issue, we employ random additive sharings $[z^{(j)}]$ of zero; S_i sends $(v_i^{(j)} \coloneqq c_i^{(j)} + [z^{(j)}]_i)_{j=1}^m$, instead of $(c_i^{(j)})_{j=1}^m$, to the client. Since $\sum_{i=1}^n v_i^{(j)} = \sum_{i=1}^n c_i^{(j)} = c^{(j)}$ whenever $\sum_{i=1}^n [z^{(j)}]_i = 0$, this masking does not affect the above procedures while hiding the crucial information of $c_i^{(j)}$.

A final concern is the client's own input. The batched proof certifies the computation the servers actually performed, so if a malicious client could hand different servers inconsistent shares of $x^{(j)}$, the servers would evaluate on a point the client never committed to and the proof would certify it regardless. Step 2 therefore invokes Π_{Input} (Protocol 7, §6.5): the servers open a pre-computed mask $[\![r^{(j)}]\!]$ to C with abort, and C broadcasts $u^{(j)} \coloneqq r^{(j)} + x^{(j)}$, from which every server derives $[\![x^{(j)}]\!] \coloneqq u^{(j)} - [\![r^{(j)}]\!]$ locally. The

Protocol 8 Π^m_{dVOPRF} – Our dVOPRF Protocol in the ($\mathcal{F}_{\mathsf{RandExp}}$, $\mathcal{F}_{\mathsf{Zero}}$, $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{RSS.Open}}$)-Hybrid Model

Players: Client C and Servers $S_1, \ldots, S_n$.
Input: Client C inputs $(x^{(1)}, \ldots, x^{(m)})$, and servers input $[\![k]\!]$.

1: Servers call $\mathcal{F}_{\mathsf{RandExp}}(e)$, $\mathcal{F}_{\mathsf{Zero}}$, and $\mathcal{F}_{\mathsf{Rand}}$ m times each to obtain $([\![(\alpha^{(j)})^e]\!])_{j\in[m]}$, $([z^{(j)}])_{j\in[m]}$, and the input masks $([\![r^{(j)}]\!])_{j\in[m]}$, respectively.
2: Client C and the servers run Π_{Input} (Protocol 7) on $(x^{(j)})_{j\in[m]}$ with the masks $([\![r^{(j)}]\!])_{j\in[m]}$, deferring its echo step to Step 6; the servers obtain $([\![x^{(j)}]\!])_{j\in[m]}$ and C aborts if the mask opening fails.
3: For every $j \in [m]$, let $[\![a^{(j)}]\!] \leftarrow [\![x^{(j)} + k]\!]$ $(= [\![x^{(j)}]\!] + [\![k]\!])$ and $[\![b^{(j)}]\!] \leftarrow [\![(\alpha^{(j)})^e]\!]$.
4: For every $i \in [n]$ and $j \in [m]$, let $a_i^{(j,\ell)} \leftarrow [a^{(j)}]_{u_\ell}$ and $b_i^{(j,\ell)} \leftarrow [b^{(j)}]_{v_\ell}$ for every $(u_\ell, v_\ell) \in \Lambda_i$, where $[\![a^{(j)}]\!]_i = \{[a^{(j)}]_\ell \mid i \notin T_\ell \in \mathcal{T}\}$, $[\![b^{(j)}]\!]_i = \{[b^{(j)}]_\ell \mid i \notin T_\ell \in \mathcal{T}\}$, and $\{\Lambda_i\}_{i\in[n]}$ is a balanced assignment (defined in §6.1).
5: Each server S_i locally computes $v_i^{(j)} = \sum_{\ell=1}^{L} a_i^{(j,\ell)} \cdot b_i^{(j,\ell)} + [z^{(j)}]_i$ for all $j \in [m]$.
6: Each server S_i computes $h_i \leftarrow H(v_i^{(1)} \| \cdots \| v_i^{(m)})$ and the deferred echo value $\chi_i \leftarrow H(u^{(1)} \| \cdots \| u^{(m)})$ of Π_{Input}, and sends (h_i, χ_i) to all other servers; S_i aborts unless $\chi_{i'} = \chi_i$ for all $i' \in [n]$.
7: Servers derive $\rho_\epsilon \leftarrow H(h_1 \| \cdots \| h_n)$, and then compute $\epsilon_j \leftarrow H(\rho_\epsilon \| j)$ for every $j \in [m]$.
8: Servers locally obtain $[\![a_i^{(j,\ell)}]\!]$ and $[\![b_i^{(j,\ell)}]\!]$ via local share conversion for all $i \in [n]$, $j \in [m]$, and $\ell \in [L]$.
9: Servers locally compute scaled inputs $[\![\hat{a}_i^{(j,\ell)}]\!] = \epsilon_j \cdot [\![a_i^{(j,\ell)}]\!]$ and $[\![\hat{b}_i^{(j,\ell)}]\!] = [\![b_i^{(j,\ell)}]\!]$ for all $i \in [n]$, $j \in [m]$, and $\ell \in [L]$.
10: Servers invoke $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ with the scaled inputs to obtain proof $([\![W]\!], [\![U]\!], [\![V]\!], [\![\Sigma]\!], [\![c]\!])$.
11: Client C calls $\mathcal{F}_{\mathsf{RSS.Open}}$ to robustly open $[\![W]\!]$, $[\![U]\!]$, $[\![V]\!]$, $[\![\Sigma]\!]$, and $[\![c]\!]$. If at least one opening fails, C aborts the protocol.
12: Client C computes $v^{(j)} = \sum_{i\in[n]} v_i^{(j)}$ for every $j \in [m]$, and re-derives $\{\epsilon_j\}_{j\in[m]}$ from $\{v_i^{(j)}\}_{i\in[n],\, j\in[m]}$.
13: Client C aborts the protocol if at least one of the following equations does not hold: $W = U \cdot V$; $\Sigma = 0$; and $c = \sum_{j=1}^{m} (\epsilon_j \sum_{i=1}^{n} v_i^{(j)})$.

Output: Client C outputs $((v^{(1)})^g, \ldots, (v^{(m)})^g)$.

servers confirm that they all received the same $u^{(j)}$ by exchanging the hash χ_i, which we fold into the hash exchange of Step 6 so that it costs no additional round. By Lemma 6.3, this fixes a unique input that the simulator can extract, keeps it hidden from any t servers, and—because the mask is opened with abort rather than reconstructed—prevents corrupted servers from shifting an honest client's input.

We prove that Π^m_{dVOPRF} securely realizes $\mathcal{F}^{\mathsf{F}_k}_{\mathsf{dVOPRF}}$ for the Gold PRF F. We sketch the proof below; the full proof is in Appendix D.7.

Theorem 6.4. *Let $p = eg + 1$ be a prime of $O(\lambda)$ bits, where p, e, g are public, and $\mathsf{F}_k(x)$ be Gold PRF over $\mathbb{F}_p$. Then, assuming random oracles, Π^m_{dVOPRF} in Protocol 8 securely realizes $\mathcal{F}^{\mathsf{F}_k}_{\mathsf{dVOPRF}}$ over $\mathbb{F}_p$, with statistical error at most $\varepsilon_{\mathsf{dVOPRF}}$ in the ($\mathcal{F}_{\mathsf{RandExp}}$, $\mathcal{F}_{\mathsf{Zero}}$, $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{RSS.Open}}$)-hybrid model, against a malicious adversary corrupting up to $t < n/2$ servers and, possibly, the client C, where $\varepsilon_{\mathsf{dVOPRF}} = n(2\lceil\log(mL)\rceil + 1)/(|\mathbb{F}_p| - 3) + 2/|\mathbb{F}_p| + 2n/(|\mathbb{F}_p| - 2n - 1) + m/|\mathbb{F}_p|$.*[5]

Proof sketch. Correctness follows from the balanced assignment, the zero masks, and the correctness of $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ (Lemma 6.2): the client's check $c = \sum_{j=1}^{m} \epsilon_j v^{(j)}$ holds, and $((\alpha^{(j)})^e)^g = 1$ whenever $\alpha^{(j)} \neq 0$. For security, the simulator emulates all hybrid functionalities and first extracts the client's input: playing the honest servers, it knows every mask $r^{(j)}$ and recovers $x^{(j)} = u^{(j)} - r^{(j)}$ from the broadcast $u^{(j)}$, which by Lemma 6.3 is exactly the input that the honest servers' sharings encode, and the input an honest client supplied. A corrupted client is a pure receiver. Given $y^{(j)} = \mathsf{F}_k(x^{(j)})$, the simulator samples $v^{(j)}$ uniformly from $\{v \in \mathbb{F}_p^* : v^g = y^{(j)}\}$, which matches the real distribution because $(\alpha^{(j)})^e$ is uniform over the kernel of $z \mapsto z^g$; it splits $v^{(j)}$ into server messages exactly as the zero masks do, and simulates the opened proof tuple using the designated-verifier privacy of Lemma 6.2. Corrupted servers see only unauthorized sets of RSS shares and uniformly distributed hybrid outputs. Finally, an honest client accepts an incorrect output only if $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ is unsound (Lemma 6.2), if the random-oracle batching coefficients satisfy a nontrivial linear equation (probability $1/|\mathbb{F}_p|$), or if some $\alpha^{(j)} = 0$ (probability at most $m/|\mathbb{F}_p|$); summing these terms gives $\varepsilon_{\mathsf{dVOPRF}}$. □

6.7 Efficiency Analysis

We analyze the communication costs of the online computations in our dVOPRF protocol Π^m_{dVOPRF} with m inputs and compare them with those of the naive protocol $\Pi^m_{\mathsf{naive\text{-}dVOPRF}}$.[6] Tab. 2 shows an efficiency comparison between them for concrete values of n, t, and m. Our protocol reduces per-server communication in every (n, t) regime—by ≈1.2–1.5× at $m = 1$ and up to ≈2.9× at $m = 100$—and saves four rounds (13 → 9). Moreover, if we allow a slight increase in rounds and use more efficient building blocks (Protocols 9 and 10 in Appendix A), the efficiency gap between them widens significantly (4.8× smaller for $(n, t, m) = (9, 4, 100)$). A detailed analysis and comparison are given in Appendix F.

Optimizations and Extensions. Appendix G presents two optimizations of Π^m_{dVOPRF}: a stratified share assignment that lets each server share its top-stratum terms by local share conversion, cutting the VIP protocol from two iterations to one for $n = 3$; and a variant with key-dependent preprocessing whose online phase runs no secure multiplication or VIP protocol and takes 4 rounds, independently of m.

7 Implementation and Evaluation

For the offline phase, we implement our protocols Π_{DegGen}, $\Pi^{\mathsf{VitH}}_{\mathsf{ZKPGen}}$, $\Pi^{\mathsf{Lig}}_{\mathsf{ZKPGen}}$ alongside the baseline Π_{AlyGen} [4, 30]. For the online phase, we implement Π^m_{dVOPRF} and the baseline $\Pi^m_{\mathsf{naive\text{-}dVOPRF}}$ [9]. Both take the client's input through Π_{Input} (Protocol 7). For end-to-end comparison, we compare with the Legendre dOPRF $\Pi_{\mathsf{L\text{-}dOPRF}}$ [28],

[5] The final term accounts for the event that some $\alpha^{(j)} = 0$, in which case C receives 0 and outputs 0 rather than $\mathsf{F}_k(x^{(j)})$; since $\mathcal{F}_{\mathsf{RandExp}}$ samples $\alpha \xleftarrow{\$} \mathbb{F}_p$, this occurs with probability at most $m/|\mathbb{F}_p|$.
[6] We assume the FS transform [19] is applied to both schemes; the FS variant of Π^m_{dVOPRF} is given in Appendix E.

Table 2: Online-phase efficiency comparison; Comm. is per-server communication in field elements.

(n,t)	Protocol (online)	$m=1$ Comm.	$m=1$ Rnds	$m=100$ Comm.	$m=100$ Rnds
$(3,1)$	$\Pi^m_{\text{naive-dVOPRF}}$	109	13	1 224.5	13
$(3,1)$	Our Π^m_{dVOPRF}	**72**	**9**	**544**	**9**
$(5,2)$	$\Pi^m_{\text{naive-dVOPRF}}$	840	13	4 957.5	13
$(5,2)$	Our Π^m_{dVOPRF}	**633**	**9**	**1 731**	**9**
$(7,3)$	$\Pi^m_{\text{naive-dVOPRF}}$	5 485	13	23 888.5	13
$(7,3)$	Our Π^m_{dVOPRF}	**4 470**	**9**	**8 964**	**9**
$(9,4)$	$\Pi^m_{\text{naive-dVOPRF}}$	32 056	13	113 230.5	13
$(9,4)$	Our Π^m_{dVOPRF}	**27 373**	**9**	**46 128**	**9**

the closest available dOPRF, benchmarked on $(4,1)$ and $(7,2)$ since theirs requires $t < n/3$.

Setup. All protocols are implemented in Rust. Protocols are executed in a simulated network adopting a WAN model with RTT = 100 ms and bandwidth BW = 100 Mbps as $T_{\text{WAN}} = T_{\text{comp}} + \text{Rounds} \cdot \text{RTT} + \frac{8 \cdot \text{TotalBytes}}{\text{BW}}$, where T_{comp} is the mean local computation time over 10 runs. The same setup applies to all our benchmarks, each averaged over 10 runs, executed on a machine running Ubuntu 22.04 with a 12th-gen Intel Core i7-12700K and 32 GB RAM. Our implementation, the benchmarks behind every number in this section, and build instructions and configuration files for each (n,t) regime we evaluate are publicly available.[7]

Parameters. We set $\kappa = 40$ (per MP-SPDZ [29]) for statistical checks. The dual-share proofs are non-interactive, so their challenges hashed from small sets are sized for the computational parameter $\lambda = 128$ (Appendix C): Π_{VitH} uses GGM tree arity $\tau = 16$, giving $R = \lceil 128/3 \rceil = 43$ repetitions; Π_{Lig} picks its layout (how many instances share a row), n_c and the number q of sampled points by the rule in Appendix C.2, giving $16384 \le n_c \le 32768$ and $364 \le q \le 588$ across our settings, with up to 148 instances per row at m=100. Benchmarks span $(n,t) \in \{(3,1),(5,2),(7,3),(9,4)\}$ for Gold PRF and $(4,1),(7,2)$ for Legendre OPRF (which requires $t < n/3$). For matched post-quantum security at $\lambda = 128$, both PRFs are configured at $|p| = 3\lambda = 384$ bits—ours via DSPRS [34], Legendre via DSLSP [28] after its classical 2λ bound is inflated by the 3/2 post-quantum factor—so all benchmarks share the same $\mathbb{F}_p$.

Metrics. Tab. 3, Tab. 4, and Tab. 5 report the detailed numerical results of our experimental runs. Each table reports computation and WAN time in milliseconds and communication in KB. Comp denotes the local computation time and WAN the total wall-clock time over the network. In Tab. 3, P2P refers to data sent to a specific receiver and Bcast to data sent to all $n-1$ remaining receivers. In Tab. 4, S↔S refers to communication between servers and S↔C to client input/output. For all tables, Rnds denotes the required rounds and Total the total communication.

[7] https://github.com/John-Katis/Verifiable-Distributed-OPRF

7.1 Offline Phase

Fig. 6 compares the offline protocols on computation overhead and communication volume at $m = 1$ (left) and $m = 100$ (right). Π_{AlyGen} is omitted from the right panel as it exceeds our memory budget. The full numerical breakdown is given in Tab. 3.

Scaling. As (n,t) grows from $(3,1)$ to $(9,4)$, Π_{AlyGen} has the worst compute scaling. Its RSS-multiplication count grows fastest, reaching ≈11.4 s of local time at $(9,4)$, m=1 and exceeding our memory budget at $(9,4)$, m=100. All three of our generators cut this by an order of magnitude or more. Π_{DegGen} has the lowest communication at every profile, but its sequential per-share check inflates its round count super-linearly ($11 \to 257$ rounds across the sweep at m=1; cf. Tab. 3), so its WAN time grows steepest and overtakes Π_{AlyGen}'s from $(7,3)$ onward. Both Π_{ZKPGen} variants are constant-round. On communication, $\Pi^{\text{Lig}}_{\text{ZKPGen}}$'s fixed code length dominates at small (n,t)—costing ≈169× Π_{DegGen}'s bytes at $(3,1)$, m=1—but amortizes rapidly as (n,t) and m grow, falling to ≈1.2×–≈2.6× by $(7,3)$–$(9,4)$; beyond the fixed code length, its row padding and per-verifier binding polynomials are the only remaining source of overhead. $\Pi^{\text{VitH}}_{\text{ZKPGen}}$'s per-leaf commitment volume instead grows linearly in m and steeply in (n,t) throughout—at $(9,4)$, m=100 it sends ≈3,165 MB (≈3.09 GB), the most of any generator, reflecting its $R = 43$ repetitions and the per-verifier authentication material each of the $n-1$ non-dealer servers independently checks.

Per-setting winners. At $(n,t) = (3,1)$, Π_{DegGen} has the lowest communication and the lowest WAN time at both batch sizes; $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ is the worst of the three on both axes here, its fixed code length far outweighing the m=1–100 instance count, and $\Pi^{\text{VitH}}_{\text{ZKPGen}}$ trails Π_{DegGen} at m=1 by ≈7.5× the bytes despite being constant-round. At $(n,t) = (5,2)$, $\Pi^{\text{VitH}}_{\text{ZKPGen}}$ has the lowest WAN time at m=1, trading ≈4.8× the communication of Π_{DegGen} for it, while $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ again trails both on every axis. At m=100, Π_{DegGen} dominates both axes at both $(3,1)$ and $(5,2)$, though $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ has already overtaken $\Pi^{\text{VitH}}_{\text{ZKPGen}}$ on both communication and WAN time at $(5,2)$ (e.g., WAN: ≈20.4 s vs ≈22.8 s), since VitH's repetition-driven broadcast material scales with m faster than Ligero's batched proof. From $(n,t) = (7,3)$ onward, the two ZKP variants show trade-offs between communication and computation. At m=1: $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ sends less data (e.g., $(7,3)$: ≈3.7 MB vs $\Pi^{\text{VitH}}_{\text{ZKPGen}}$'s ≈5.8 MB), but $\Pi^{\text{VitH}}_{\text{ZKPGen}}$ has the lower WAN time (e.g., $(7,3)$: ≈2.9 s vs ≈6.5 s); at m=100, $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ wins both axes outright, by a wide margin on communication (e.g., $(7,3)$: ≈8.3 MB vs ≈430 MB). **We argue that $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ will demonstrate clear efficiency advantage in scaling when deployed at a computation-performant host**.

7.2 Online Phase

Fig. 7 compares Π^m_{dVOPRF} against $\Pi^m_{\text{naive-dVOPRF}}$ at m=1 (left) and m=100 (right). Detailed numbers are given in Tab. 4.

Scaling. On communication, Π^m_{dVOPRF} scales more gracefully: the naive RSS share/open routine sends $O(N)$ shares per opening ($N = \binom{n}{t}$), while our fused $\Pi^{\text{Prl}}_{\text{VIP}}$ closes over a single opened element. Its server-to-server volume advantage therefore grows from ≈2× at $(3,1)$ to ≈9× at $(9,4)$, at both m=1 and m=100. On

Table 3: Offline phase benchmarks for $m \in \{1, 100\}$. Comm. in KB. Π_{AlyGen} at $(9, 4)$, m=100 is omitted.

		$m = 1$						$m = 100$					
(n, t)	Protocol	Comp (ms)	Rnds	P2P (KB)	Bcast (KB)	Total (KB)	WAN (ms)	Comp (ms)	Rnds	P2P (KB)	Bcast (KB)	Total (KB)	WAN (ms)
(3, 1)	Π_{AlyGen} [4, 30]	17.3	21	19.64	**2.25**	21.89	2 119.1	129.3	21	527.98	**2.25**	530.23	2 272.7
(3, 1)	Our Π_{DegGen}	**10.4**	**11**	**8.06**	**2.25**	**10.31**	**1 111.2**	**26.6**	**11**	**38.43**	**2.25**	**40.68**	**1 129.9**
(3, 1)	Our $\Pi^{\text{VitH}}_{\text{ZKPGen}}$	29.2	**11**	40.27	37.28	77.55	1 135.6	1 896.2	**11**	3 258.75	3 505.17	6 763.92	3 550.3
(3, 1)	Our $\Pi^{\text{Lig}}_{\text{ZKPGen}}$	2 240.5	**11**	199.91	1 538.26	1 738.16	3 482.9	6 188.7	**11**	225.64	1 684.03	1 909.67	7 445.1
(5, 2)	Π_{AlyGen} [4, 30]	56.5	25	249.09	**14.06**	263.16	2 578.1	1 106.6	25	4 202.00	**14.06**	4 216.04	3 952.0
(5, 2)	Our Π_{DegGen}	**34.6**	25	**144.38**	**14.06**	**158.44**	2 547.5	**125.3**	25	**412.68**	**14.06**	**426.74**	**2 660.3**
(5, 2)	Our $\Pi^{\text{VitH}}_{\text{ZKPGen}}$	198.8	**13**	627.00	139.17	766.17	**1 561.6**	16 500.2	**13**	48 692.06	12 524.12	61 216.18	22 815.0
(5, 2)	Our $\Pi^{\text{Lig}}_{\text{ZKPGen}}$	3 378.9	**13**	527.03	1 618.43	2 145.46	4 854.6	18 867.2	**13**	700.83	2 451.29	3 152.11	20 425.4
(7, 3)	Π_{AlyGen} [4, 30]	528.3	29	2 300.48	**78.75**	2 379.23	3 623.2	33 829.8	29	27 608.76	**78.75**	27 687.51	38 998.0
(7, 3)	Our Π_{DegGen}	**111.1**	75	**1 352.81**	**78.75**	**1 431.56**	7 728.4	**1 283.1**	75	3 126.28	**78.75**	**3 204.64**	**9 045.7**
(7, 3)	Our $\Pi^{\text{VitH}}_{\text{ZKPGen}}$	1 121.1	**13**	5 333.03	583.77	5 916.80	**2 905.8**	102 036.1	**13**	389 296.67	50 618.54	439 915.21	139 373.9
(7, 3)	Our $\Pi^{\text{Lig}}_{\text{ZKPGen}}$	4 894.6	**13**	2 038.70	1 751.87	3 789.78	6 505.1	87 954.0	**13**	**2 872.35**	5 591.72	8 464.07	89 947.4
(9, 4)	Π_{AlyGen} [4, 30]	11 361.1	31	16 548.17	**413.44**	16 961.61	15 850.6	(skipped: witness store > memory budget)					
(9, 4)	Our Π_{DegGen}	**562.3**	257	**10 922.24**	**413.44**	**11 335.68**	27 190.9	**15 318.8**	257	19 815.00	**413.44**	**20 228.43**	**42 675.9**
(9, 4)	Our $\Pi^{\text{VitH}}_{\text{ZKPGen}}$	6 375.1	**15**	40 978.60	2 555.81	43 534.42	**11 441.4**	598 505.5	**15**	3 025 671.76	214 747.63	3 240 809.36	865 492.6
(9, 4)	Our $\Pi^{\text{Lig}}_{\text{ZKPGen}}$	11 304.4	**15**	11 645.19	2 290.99	13 936.19	13 946.1	461 238.3	**15**	**16 829.17**	19 102.32	35 931.50	465 681.8

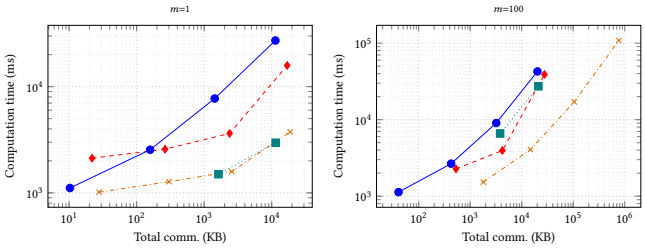


Figure 6: Offline phase: total communication vs. computation time. Each line traces $(n, t) \in \{(3, 1), (5, 2), (7, 3), (9, 4)\}$ left-to-right; $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ traces $(7, 3) \rightarrow (9, 4)$. Legend: Π_{AlyGen} [4, 30] (dashed, ♦); Π_{DegGen} (solid, •); $\Pi^{\text{VitH}}_{\text{ZKPGen}}$ (dash-dot, ×); $\Pi^{\text{Lig}}_{\text{ZKPGen}}$ (dotted, ■).

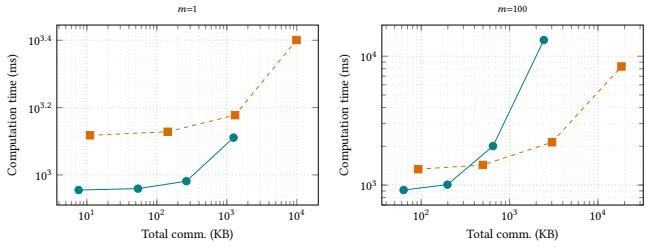


Figure 7: Offline phase: total communication vs. computation time. Each line traces $(n, t) \in \{(3, 1), (5, 2), (7, 3), (9, 4)\}$ left-to-right. Legend: $\Pi^{m}_{\text{naive-dVOPRF}}$ [9] (dashed, ■); our Π^{m}_{dVOPRF} (solid, •).

total bytes (including the client side communications) the advantage is smaller, ≈1.5× to ≈8× over the same range, because Π_{Input} adds client-facing traffic to both protocols. On computation, our local work grows super-linearly in (n, t) and linearly in m, since fusing multiplication with verification forces each server to evaluate $O(Lm)$ cross-products ($L = N^2/n$), whereas the naive baseline pays only a linear-in-m DZKP. At $(7, 3)$ and $(9, 4)$ for m=100, the baseline's per-server compute is the lower of the two.

Per-setting winners. Π^{m}_{dVOPRF} wins on both plotted axes (total communication and WAN time) at every (n, t) for m=1 and through $(7, 3)$ for m=100. At $(9, 4)$, m=100 the naive baseline's lighter per-server compute lets it finish in ≈8.3 s to our ≈13.3 s, though it still sends ≈7.6× more data (≈9× server-to-server).

7.3 End-to-End Performance

We construct a full deployment by composing one offline protocol with one online protocol. For our three end-to-end constructions, we write $\Pi^{\text{X}}_{\text{dVOPRF}}$ for the pairing of our online Π^{m}_{dVOPRF} with offline preprocessing $\text{X} \in \{\text{DegGen}, \text{VitH}, \text{Lig}\}$. The naive baseline $\Pi^{\text{Aly}}_{\text{DZKP}}$ pairs the Aly offline phase Π_{AlyGen} with the naive online protocol $\Pi^{m}_{\text{naive-dVOPRF}}$. Fig. 8 compares the full deployment cost (offline+online) of our three constructions against $\Pi^{\text{Aly}}_{\text{DZKP}}$ and $\Pi_{\text{L-dOPRF}}$ at m=1 (left) and m=100 (right). Detailed numbers are reported in Tab. 5.

The naive $\Pi^{\text{Aly}}_{\text{DZKP}}$ baseline is dominated on both axes by at least one of our three constructions at every profile: by $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$ at $(3, 1)$–$(5, 2)$, m=1 and every m=100 profile, and by $\Pi^{\text{VitH}}_{\text{dVOPRF}}$ or $\Pi^{\text{Lig}}_{\text{dVOPRF}}$ at $(7, 3)$–$(9, 4)$, m=1. Among our constructions, $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$ has the cheapest communication at every profile, and the lowest WAN time at every m=100 profile as well as at $(3, 1)$, m=1. From $(5, 2)$, m=1 onward, $\Pi^{\text{VitH}}_{\text{dVOPRF}}$ has the lowest WAN time instead. $\Pi^{\text{Lig}}_{\text{dVOPRF}}$ is now instantiable at every threshold, but at $(3, 1)$–$(5, 2)$ its fixed code length dominates and it is the most expensive of the three constructions on every axis at m=1; by m=100 it already sends less data than $\Pi^{\text{VitH}}_{\text{dVOPRF}}$ at both thresholds, mirroring the offline-phase crossover above. $\Pi_{\text{L-dOPRF}}$ has very low computation at $(4, 1)$ but its

Table 4: Online phase: our batched VIP (client-verified) vs. batched DZKP (server-verified) at $m \in \{1, 100\}$, both with Π_{Input}-verified client input. Communication split into server↔server (p2p + broadcast) and server↔client, in KB.

		$m = 1$						$m = 100$					
(n, t)	Protocol	Comp (ms)	Rnds	S↔S (KB)	S↔C (KB)	Total (KB)	WAN (ms)	Comp (ms)	Rnds	S↔S (KB)	S↔C (KB)	Total (KB)	WAN (ms)
$(3, 1)$	$\Pi^{m}_{\text{naive-dVOPRF}}$ [9]	10.6	13	10.45	**0.66**	11.11	1 311.5	21.1	13	36.19	56.35	92.53	1 328.7
$(3, 1)$	Our Π^{m}_{dVOPRF}	**1.2**	**9**	**5.25**	2.39	**7.64**	**901.9**	**9.9**	**9**	**19.03**	**44.16**	**63.19**	**915.1**
$(5, 2)$	$\Pi^{m}_{\text{naive-dVOPRF}}$ [9]	31.7	13	140.78	**2.27**	143.05	1 343.4	92.2	13	286.75	211.10	497.85	1 433.0
$(5, 2)$	Our Π^{m}_{dVOPRF}	**6.2**	**9**	**45.03**	8.91	**53.94**	**910.6**	**91.1**	**9**	**96.22**	**101.72**	**197.94**	**1 007.3**
$(7, 3)$	$\Pi^{m}_{\text{naive-dVOPRF}}$ [9]	97.8	13	1 305.61	**8.76**	1 314.37	1 505.4	**602.6**	13	2 174.25	853.35	3 027.60	2 150.6
$(7, 3)$	Our Π^{m}_{dVOPRF}	**36.4**	**9**	**228.16**	36.42	**264.58**	**958.1**	1 054.6	**9**	**385.00**	**263.81**	**648.81**	**2 007.7**
$(9, 4)$	$\Pi^{m}_{\text{naive-dVOPRF}}$ [9]	403.7	13	9 874.13	**36.15**	9 910.27	2 515.5	**5 505.6**	13	14 891.06	3 586.23	18 477.29	**8 319.2**
$(9, 4)$	Our Π^{m}_{dVOPRF}	**288.2**	**9**	**1 093.78**	156.09	**1 249.88**	**1 290.6**	12 249.5	**9**	**1 617.47**	**824.34**	**2 441.81**	13 349.5

per-query communication scales catastrophically, reaching ≈1.3 GB at t=2 and ≈132 GB at m=100.

At m=1, $\Pi^{\mathsf{DegGen}}_{\text{dVOPRF}}$ has the lowest WAN time only at $(3, 1)$. $\Pi^{\mathsf{VitH}}_{\text{dVOPRF}}$ has it from $(5, 2)$ onward. $\Pi_{\text{L-dOPRF}}$ outperforms our protocols only at $(4, 1)$, m=1, beating $\Pi^{\mathsf{VitH}}_{\text{dVOPRF}}$ by ≈3.5× on WAN time but at ≈39.7× more traffic; Everywhere else our protocols dominate by orders of magnitude (e.g., ≈2,200× on WAN time and ≈8,800× on communication at t=2, m=100). At $(7, 3)$–$(9, 4)$, $\Pi^{\mathsf{DegGen}}_{\text{dVOPRF}}$ has the lowest WAN time at m=100 and $\Pi^{\mathsf{VitH}}_{\text{dVOPRF}}$ at m=1. $\Pi^{\mathsf{Lig}}_{\text{dVOPRF}}$ sends the least data among the two ZKP variants at both batch sizes.

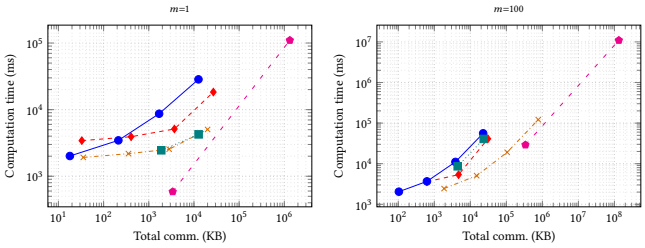


Figure 8: End-to-end phase: total communication vs. computation time. Our protocols and the Aly baseline trace $(n, t) \in \{(3, 1), (5, 2), (7, 3), (9, 4)\}$ left-to-right; $\Pi_{\text{L-dOPRF}}$ traces $(4, 1) \rightarrow (7, 2)$. Legend: $\Pi^{\mathsf{Aly}}_{\text{DZKP}}$ (dashed, ♦); $\Pi^{\mathsf{DegGen}}_{\text{dVOPRF}}$ (solid, •); $\Pi^{\mathsf{VitH}}_{\text{dVOPRF}}$ (dash-dot, ×); $\Pi^{\mathsf{Lig}}_{\text{dVOPRF}}$ (dotted, ■); $\Pi_{\text{L-dOPRF}}$ [28] (loosely dashed, pentagon).

8 Limitations and Open Problems

Replicated secret sharing gives each server $N' = \binom{n-1}{t}$ shares, a count that grows combinatorially in n; our constructions therefore target the small-(n, t) regime in which threshold deployments typically operate, and a Shamir-based variant with comparable concrete efficiency for large n remains open. In addition, the round count of Π_{DegGen} grows linearly in N, and the online computation of Π^{m}_{dVOPRF} grows super-linearly in (n, t); reducing either while keeping communication low is a natural next step.

Acknowledgments

This work was partially supported by the University of St. Gallen's Basic Research Funding Scheme (GFF). It was also partially supported by JST K Program Grant Number JPMJKP24U2, and by JSPS KAKENHI Grant Numbers JP26K02908, JP26K02909, and JP26K00966.

References

[1] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. 2024. Crypto Dark Matter on the Torus - Oblivious PRFs from Shallow PRFs and TFHE. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI (Lecture Notes in Computer Science)*, Marc Joye and Gregor Leander (Eds.). Springer, 447–476. doi:10.1007/978-3-031-58751-1_16

[2] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. 2021. Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices. In *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II (Lecture Notes in Computer Science)*, Juan A. Garay (Ed.). Springer, 261–289. doi:10.1007/978-3-030-75248-4_10

[3] Martin R. Albrecht and Kamil Doruk Gür. 2024. Verifiable Oblivious Pseudorandom Functions from Lattices: Practical-Ish and Thresholdisable. In *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part IV (Lecture Notes in Computer Science)*, Kai-Min Chung and Yu Sasaki (Eds.). Springer, 205–237. doi:10.1007/978-981-96-0894-2_7

[4] Abdelrahaman Aly, Aysajan Abidin, and Svetla Nikova. 2018. Practically Efficient Secure Distributed Exponentiation Without Bit-Decomposition. In *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers (Lecture Notes in Computer Science)*, Sarah Meiklejohn and Kazue Sako (Eds.). Springer, 291–309. doi:10.1007/978-3-662-58387-6_16

[5] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 2087–2104. doi:10.1145/3133956.3134104

[6] Andrea Basso. 2023. A Post-Quantum Round-Optimal Oblivious PRF from Isogenies. In *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers (Lecture Notes in Computer Science)*, Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen (Eds.). Springer, 147–168. doi:10.1007/978-3-031-53368-6_8

[7] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. 2023. Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V (Lecture Notes in Computer Science)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 581–615. doi:10.1007/978-3-031-38554-4_19

[8] Ward Beullens, Lucas Dodgson, Sebastian H. Faller, and Julia Hesse. 2025. The 2Hash OPRF Framework and Efficient Post-quantum Instantiations. In *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques Madrid, Spain, May 4-8, 2025, Proceedings, Part VIII (Lecture Notes in Computer Science)*, Serge Fehr and

Table 5: End-to-end benchmarks for $m \in \{1, 100\}$, with fresh α^e per input. Online phase is Π^m_{dVOPRF} throughout ($\Pi^m_{\text{naive-dVOPRF}}$ for $\Pi^{\text{Aly}}_{\text{DZKP}}$). *Total* sums p2p, broadcast, and server→client traffic; comm. in KB. $\Pi^{\text{Aly}}_{\text{DZKP}}$ at $(9, 4)$, m=100 is omitted. For $\Pi_{\text{L-dOPRF}}$, Comp is WAN − Rnds · RTT − 8 · Total/BW.

		$m = 1$				$m = 100$			
(n, t)	Protocol (offline + online)	Comp (ms)	Rnds	Total (KB)	WAN (ms)	Comp (ms)	Rnds	Total (KB)	WAN (ms)
$(3, 1)$	$\Pi^{\text{Aly}}_{\text{DZKP}}$	25.0	34	33.00	3 427.7	149.1	34	622.77	3 600.2
$(3, 1)$	Our $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$	11.6	20	**17.95**	2 013.1	**36.6**	20	**103.87**	**2 045.1**
$(3, 1)$	Our $\Pi^{\text{VitH}}_{\text{dVOPRF}}$	30.1	20	87.23	2 037.1	1 934.5	20	6 827.11	4 493.8
$(3, 1)$	Our $\Pi^{\text{Lig}}_{\text{dVOPRF}}$	2 233.3	20	1 745.81	4 376.3	6 214.2	20	1 972.85	8 375.8
$(4, 1)$	$\Pi_{\text{L-dOPRF}}$ [28]	**9.8**	**3**	3 381.00	**586.8**	1 040.7	**3**	338 100.00	29 037.8
$(5, 2)$	$\Pi^{\text{Aly}}_{\text{DZKP}}$	87.5	38	406.21	3 920.8	1 179.5	38	4 713.88	5 365.7
$(5, 2)$	Our $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$	**40.9**	34	**212.38**	3 458.3	**217.0**	34	**624.67**	**3 668.2**
$(5, 2)$	Our $\Pi^{\text{VitH}}_{\text{dVOPRF}}$	208.5	22	820.11	**2 475.6**	16 768.2	22	61 414.09	23 999.2
$(5, 2)$	Our $\Pi^{\text{Lig}}_{\text{dVOPRF}}$	3 392.3	22	2 199.40	5 772.5	18 877.4	22	3 350.06	21 351.8
$(7, 2)$	$\Pi_{\text{L-dOPRF}}$ [28]	1 235.2	**3**	1 323 617.90	109 966.0	123 690.4	**3**	132 361 785.90	10 967 067.9
$(7, 3)$	$\Pi^{\text{Aly}}_{\text{DZKP}}$	620.9	42	3 693.60	5 123.4	34 333.0	42	30 715.07	41 049.2
$(7, 3)$	Our $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$	**145.9**	84	**1 696.14**	8 684.9	**2 344.5**	84	**3 853.84**	**11 060.2**
$(7, 3)$	Our $\Pi^{\text{VitH}}_{\text{dVOPRF}}$	1 183.2	**22**	6 181.37	**3 889.6**	103 534.4	**22**	440 564.05	141 825.4
$(7, 3)$	Our $\Pi^{\text{Lig}}_{\text{dVOPRF}}$	4 964.2	**22**	4 054.36	7 496.3	88 994.0	**22**	9 112.87	91 940.5
$(9, 4)$	$\Pi^{\text{Aly}}_{\text{DZKP}}$	11 752.9	44	26 872.86	18 354.3	(skipped: witness store > memory budget)			
$(9, 4)$	Our $\Pi^{\text{DegGen}}_{\text{dVOPRF}}$	**847.8**	266	**12 585.56**	28 478.8	**27 591.9**	266	**22 670.23**	**56 049.0**
$(9, 4)$	Our $\Pi^{\text{VitH}}_{\text{dVOPRF}}$	6 738.1	**24**	44 784.29	**12 806.8**	616 425.4	**24**	3 243 251.21	884 512.6
$(9, 4)$	Our $\Pi^{\text{Lig}}_{\text{dVOPRF}}$	11 618.3	**24**	15 186.07	15 262.3	472 811.0	**24**	38 373.28	478 354.6

Pierre-Alain Fouque (Eds.). Springer, 332–362. doi:10.1007/978-3-031-91101-9_12

[9] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III (Lecture Notes in Computer Science)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 67–97. doi:10.1007/978-3-030-26954-8_3

[10] Dan Boneh, Dmitry Kogan, and Katharine Woo. 2020. Oblivious Pseudorandom Functions from Isogenies. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 520–550. doi:10.1007/978-3-030-64834-3_18

[11] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2020. Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III (Lecture Notes in Computer Science)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 244–276. doi:10.1007/978-3-030-64840-4_9

[12] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. 2020. Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1451. https://eprint.iacr.org/2020/1451

[13] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof. 2023. Fast large-scale honest-majority MPC for malicious adversaries. *Journal of Cryptology* 36, 3 (2023), 15. doi:10.1007/s00145-023-09453-7

[14] Ronald Cramer, Ivan Damgård, and Yuval Ishai. 2005. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings (Lecture Notes in Computer Science)*, Joe Kilian (Ed.). Springer, 342–362. doi:10.1007/978-3-540-30576-7_19

[15] Ivan Damgård. 1990. On the Randomness of Legendre and Jacobi Sequences. In *Advances in Cryptology — CRYPTO '88 (Lecture Notes in Computer Science)*. Springer New York, New York, NY, 163–172. doi:10.1007/0-387-34799-2_13

[16] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*. Springer Berlin Heidelberg, Berlin, Heidelberg, 572–590. doi:10.1007/978-3-540-74143-5_32

[17] Sebastian H. Faller, Astrid Ottenhues, and Johannes Ottenhues. 2023. Composable Oblivious Pseudo-random Functions via Garbled Circuits. In *Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings (Lecture Notes in Computer Science)*, Abdelrahaman Aly and Mehdi Tibouchi (Eds.). Springer, 249–270. doi:10.1007/978-3-031-44469-2_13

[18] Uriel Feige, Joe Kilian, and Moni Naor. 1994. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, Frank Thomson Leighton and Michael T. Goodrich (Eds.). ACM, 554–563. doi:10.1145/195058.195408

[19] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings (Lecture Notes in Computer Science)*, Andrew M. Odlyzko (Ed.). Springer, 186–194. doi:10.1007/3-540-47721-7_12

[20] L. R. Ford and D. R. Fulkerson. 1956. Maximal Flow Through a Network. *Canadian Journal of Mathematics* 8 (1956), 399–404. doi:10.4153/CJM-1956-045-5

[21] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *Theory of Cryptography*, Joe Kilian (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 303–324. doi:10.1007/978-3-540-30576-7_17

[22] Andrew V. Goldberg and Satish Rao. 1998. Beyond the Flow Decomposition Barrier. *J. ACM* 45, 5 (1998), 783–797. doi:10.1145/290179.290181

[23] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. 2021. ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. In *Advances in Cryptology - CRYPTO 2021 (Lecture Notes in Computer Science)*, Tal Malkin and Chris Peikert (Eds.). Springer, 244–274. doi:10.1007/978-3-030-84245-1_9

[24] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. 2016. MPC-Friendly Symmetric Key Primitives. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 430–443. doi:10.1145/2976749.2978332

[25] Lena Heimberger, Tobias Hennerbichler, Fredrik Meisingseth, Sebastian Ramacher, and Christian Rechberger. 2024. OPRFs from Isogenies: Designs and Analysis. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*, Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas (Eds.). ACM, 575–588. doi:10.1145/3634737.3645010

[26] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. 2014. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, 233–253. doi:10.1007/978-3-662-45608-8_13

[27] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. 2017. TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings (Lecture Notes in Computer Science)*, Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi (Eds.). Springer, 39–58. doi:10.1007/978-3-319-61204-1_3

[28] Novak Kaluđerović, Nan Cheng, and Aikaterini Mitrokotsa. 2025. A Post-quantum Distributed OPRF from the Legendre PRF. In *Computer Security – ESORICS 2025 (Lecture Notes in Computer Science)*. Springer Nature Switzerland, 205–225. doi:10.1007/978-3-032-07891-9_11 Preprint available as Cryptology ePrint Archive, Report 2024/544, https://eprint.iacr.org/2024/544.

[29] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1575–1590. doi:10.1145/3372297.3417872

[30] Bart Mennink. 2023. Secure Distributed Modular Exponentiation: Systematic Analysis and New Results. *IEEE Transactions on Information Forensics and Security* 18 (2023), 4188–4197. doi:10.1109/TIFS.2023.3293396

[31] Stephen C. Pohlig and Martin E. Hellman. 1978. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.). *IEEE Trans. Inf. Theory* 24, 1 (1978), 106–110. doi:10.1109/TIT.1978.1055817

[32] István András Seres, Máté Horváth, and Péter Burcsi. 2025. The Legendre pseudorandom function as a multivariate quadratic cryptosystem: security and applications. *Appl. Algebra Eng. Commun. Comput.* 36, 2 (2025), 223–253. doi:10.1007/S00200-023-00599-2

[33] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2986–3001. doi:10.1145/3460120.3484556

[34] Yibin Yang, Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, and Tal Rabin. 2025. Gold OPRF: Post-Quantum Oblivious Power-Residue PRF. In *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, Marina Blanton, William Enck, and Cristina Nita-Rotaru (Eds.). IEEE, 259–278. doi:10.1109/SP61157.2025.00116

A Building Block Protocols for RSS

This appendix gives concrete realizations of the RSS building-block functionalities listed in §3.2.

Protocol 9 $\Pi_{\text{RSS.Share}}$ – (t, n)-RSS Sharing

Players: A dealer D and servers $S_1, \ldots, S_n$.

Input: Dealer D holds a secret $s \in \mathbb{F}_p$; servers hold a pre-computed double sharing $([r], [\![r]\!])$ of a random $r \in \mathbb{F}_p$.

1: Each server S_i sends $[r]_i$ to D.
2: D reconstructs $r \leftarrow \sum_{i=1}^{n} [r]_i$ and sends $\delta \leftarrow s - r$ to each server S_j for $j \in Q$, where $Q \subset [n]$ with $|Q| = t + 1$ is a fixed public subset.
3: Each server S_j for $j \in Q$ locally updates $[\![s]\!]_j$ by adding δ to the appropriate share component of $[\![r]\!]_j$. Each server S_i for $i \notin Q$ sets $[\![s]\!]_i \leftarrow [\![r]\!]_i$.

Output: Each server S_i holds a share $[\![s]\!]_i$.

Protocol 9 requires 2 rounds of communication: one round where each server sends 1 field element (its additive share $[r]_i$) to the dealer D, and one round where D sends 1 field element δ to the $t + 1$ servers in Q.

Protocol 10 requires 1 round of communication. For RSS, each server S_j receives $\binom{n-1}{t-1}$ share components from other servers, plus

Protocol 10 $\Pi_{\text{RSS.Open}}$ – (t, n)-RSS Opening

Players: Servers $S_1, \ldots, S_n$.

Input: A public access policy $\mathcal{A}$. Each server S_i holds $[\![s]\!]_{\mathcal{T}_i}$ which is an ordered list of S_i's local shares.

1: $\forall i \in [n]$, server S_i computes $h_i \leftarrow H([\![s]\!]_{\mathcal{T}_i})$, where $H\colon \{0,1\}^* \to \{0,1\}^\lambda$.
2: Each server S_i sends h_i and $[\![s]\!]_{\mathcal{A}_i}$ (a list of shares defined by the public access policy $\mathcal{A}$) to each S_j for $j \in [n] \setminus \{i\}$.
3: Each server S_j reconstructs s from the received shares together with its own local shares.
4: Each server S_j verifies $h_i \stackrel{?}{=} H([\![s]\!]_{\mathcal{T}_i})$ for all $i \in [n]$. `// abort if any check fails`

Output: All servers learn s, or $\perp$.

$n - 1$ hash values. The hash verification enables security with abort: after reconstruction, each server S_j can determine every other server's full local share from the complete share information and check it against the received hash, detecting any inconsistency.

Protocol 11 Π_{A2R} – Additive-to-Replicated Sharing Conversion

Players: Servers $S_1, \ldots, S_n$.

Input: Servers hold an additive sharing $[s]$; a pre-computed double sharing $([r], [\![r]\!])$ of a random $r \in \mathbb{F}_p$.

1: Each server S_i locally computes $[s - r]_i \leftarrow [s]_i - [r]_i$.
2: Each server S_i sends $[s - r]_i$ to a designated aggregator S_q, where $q \in Q$ and $Q \subset [n]$ with $|Q| = t + 1$ is a fixed public subset.
3: S_q reconstructs $\delta \leftarrow \sum_{i=1}^{n} [s - r]_i$ and sends δ to each server S_j for $j \in Q \setminus \{q\}$.
4: Each server S_j for $j \in Q$ locally updates $[\![s]\!]_j$ by adding δ to the appropriate share component of $[\![r]\!]_j$. Each server S_i for $i \notin Q$ sets $[\![s]\!]_i \leftarrow [\![r]\!]_i$.

Output: Each server S_i holds $[\![s]\!]_i$.

Protocol 11 requires 2 rounds of communication: in the first round, each server sends 1 field element to the aggregator S_q; in the second round, S_q sends 1 field element δ to each of the t other servers in Q. Only the $t + 1$ servers in Q update their shares; the remaining $n - t - 1$ servers keep their shares of r unchanged. The double sharing $([r], [\![r]\!])$ is assumed to be pre-computed in the offline phase.

Protocol 12 $\Pi_{\text{RSS.Mul}}$ – Multiplication over (t, n)-Replicated Secret Sharing

Players: Servers $S_1, \ldots, S_n$.

Input: Servers hold $[\![a]\!]$ and $[\![b]\!]$; a pre-computed double sharing $([r], [\![r]\!])$ of a random $r \in \mathbb{F}_p$.

1: Each server S_i locally computes an additive share $[ab]_i$ from its local shares $[\![a]\!]_i$ and $[\![b]\!]_i$ via cross-multiplication.
2: Servers invoke Π_{A2R} over $[ab]$ to obtain $[\![ab]\!]$.

Output: Each server S_i holds $[\![ab]\!]_i$.

Protocol 12 requires 2 rounds of communication (from the invocation of Π_{A2R}): each server sends 1 field element to the aggregator

S_q, who then forwards δ to the t other servers in Q. The overall communication overhead is $\frac{3t+1}{2t+1} \approx 1.5$ elements per party.

Generating Double Sharings. Both $\Pi_{\text{RSS.Share}}$ and $\Pi_{\text{RSS.Mul}}$ consume a pre-computed double sharing $([r], [\![r]\!])$ of a random value $r \in \mathbb{F}_p$. We describe a non-interactive offline protocol for generating such pairs, following the approach of Cramer et al. [14].

Protocol 13 $\Pi_{\text{DoubleRand}}$ – Non-Interactive Double Sharing Generation

Players: Servers $S_1, \ldots, S_n$.

Pre-shared material: Replicated PRF keys $\{k_T\}_{T \in \mathcal{T}}$, where each key k_T is held by all servers S_i with $i \notin T$; $\forall i, j \in [n]$, pairwise PRG seeds $\{s_{i,j}\}$ shared between every pair of servers S_i, S_j.

Input: A public counter ctr.

`// Step 1 – RSS component`

1: For each $T \in \mathcal{T}$ with $i \notin T$, server S_i locally computes $[r]_T \leftarrow \mathsf{PRF}_{k_T}(\text{ctr})$.

2: Each server S_i sets its threshold share $[\![r]\!]_i \leftarrow \{[r]_T \mid i \notin T\}$.

`// Step 2 – Re-randomized additive share`

3: Let $P\colon \mathcal{T} \to [n]$ be a public *covering policy* satisfying $P(T) \notin T$ for every $T \in \mathcal{T}$. Each server S_i locally computes:

$$[r]_i \leftarrow \sum_{T \in \mathcal{T}:\ P(T)=i} [r]_T + \sum_{\substack{j \in [n] \\ j \neq i}} (-1)^{\mathbf{1}[j<i]} \mathsf{PRG}(s_{i,j}, \text{ctr}).$$

Output: Servers hold $([r], [\![r]\!])$ where $r = \sum_{T \in \mathcal{T}} [r]_T$.

Protocol 13 is non-interactive: all computations are local. Each server S_i evaluates the PRF for each $T \in \mathcal{T}$ with $i \notin T$, yielding $\binom{n-1}{t}$ PRF evaluations per server. Write b_i for the first sum above and z_i for the second, so that $[r]_i = b_i + z_i$. The base additive share b_i is obtained by summing the RSS components assigned to S_i via the covering policy P: component $[r]_T$ is assigned to $S_{P(T)}$, which holds key k_T since $P(T) \notin T$, and each component is counted exactly once since P is a function. Since the PRF keys are replicated (all parties outside T hold k_T), the base shares b_i are deterministic and could be recomputed by other parties holding the same keys. The zero-sharing $\{z_i\}$ from pairwise PRG seeds re-randomizes the additive shares: $\sum_i z_i = 0$ by construction (each pairwise PRG term appears once with + and once with −), so correctness is preserved ($\sum_i [r]_i = \sum_i b_i = r$). For privacy, let $C \subset [n]$ with $|C| \leq t$ be the corrupted set and let $\mathcal{H} \coloneqq [n] \setminus C$ be the honest parties, so that $|\mathcal{H}| = n - t \geq t + 1 \geq 2$. A corrupted party holds k_T exactly when it lies outside T, so the adversary can recompute $[r]_T$ precisely for those T with $C \not\subseteq T$; for any $T \supseteq C$—at least one exists, since $|C| \leq t = |T|$—the value $[r]_T$ is pseudorandom in its view, and the covering policy assigns it to an *honest* party, because $P(T) \notin T \supseteq C$. Writing e_i for the standard basis vector of coordinate i, that term shifts the honest shares $([r]_h)_{h \in \mathcal{H}}$ along $e_{P(T)}$, while the unknown PRG term of each honest pair $\{h, h'\} \subseteq \mathcal{H}$ shifts them along $e_h - e_{h'}$. Since $|\mathcal{H}| \geq 2$, the pair directions span all zero-sum vectors, and $e_{P(T)}$ has coordinate sum 1 and so lies outside that subspace; together they span $\mathbb{F}_p^{|\mathcal{H}|}$. Hence the honest parties' additive shares $[r]_h = b_h + z_h$ are jointly computationally indistinguishable from uniform, even though the adversary can recompute every other $[r]_T$ from its replicated keys.

B Aly et al.'s Protocol 7 in the RSS Setting

Protocol 14 gives the full pseudocode for Π_{AlyGen}, the honest-majority RSS instantiation of Aly et al.'s public-exponent construction [4, Protocol 7] (with the corrected complexity analysis of [30]) used as the state-of-the-art baseline in §5.

Protocol 14 Π_{AlyGen} – Baseline offline generation via the public-exponent protocol of [4].

Players: Servers $S_1, \ldots, S_n$.

Parameters: Public exponent e with $\gcd(e, p-1) = 1$; fixed generator g of $\mathbb{F}_p^*$.

1: Servers invoke $\mathcal{F}_{\text{Rand}}$ to obtain $[r']_{p-1} \in \mathbb{Z}_{p-1}^*$ and $[\![\alpha]\!]_p \in \mathbb{F}_p$.

2: Servers invoke $\Pi_{\exp(b,[\cdot])}$ [4, Prot. 6] to compute $[\![\bar{r}]\!]_p \leftarrow \exp(g, [r']_{p-1})$, yielding $\bar{r} = g^{r'}$.

3: Servers invoke $\Pi_{\text{RSS.Mul}}$ to compute $[\![c]\!]_p \leftarrow [\![\bar{r}]\!]_p \cdot [\![\alpha]\!]_p$; the batched DZKP verification $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ is deferred to the end.

4: $c \leftarrow \text{Open}([\![c]\!]_p)$; if $c = 0$, abort.

5: Each server locally computes $c' \leftarrow c^e$, $[\bar{e}]_{p-1} \leftarrow -e \cdot [r']_{p-1}$.

6: Servers invoke $\Pi_{\exp(b,[\cdot])}$ to compute $[\![\rho]\!]_p \leftarrow \exp(g, [\bar{e}]_{p-1})$, yielding $\rho = g^{-er'}$.

7: Each server locally computes $[\![\alpha^e]\!]_p \leftarrow c' \cdot [\![\rho]\!]_p$.

Output: $[\![\alpha^e]\!]_p$, satisfying $\alpha^e = c^e \cdot g^{-er'} = (g^{r'}\alpha)^e \cdot g^{-er'}$.

C Full Dual-Share ZKP Protocols

This appendix presents the full protocol descriptions for the two dual-share consistency ZKP instantiations introduced in §5.2.2 and §5.2.3: the VOLEitH-based Π_{VitH} and the Ligero-based Π_{Lig}.

C.1 Full VOLEitH Protocol

We now give the protocol descriptions and supporting derivations for Π_{VitH} (cf. §5.2.2).

Prove workflow. In Protocol 15, the prover expands a random seed into τ virtual VOLE shares via a GGM tree, uses them to mask the extended witness, and commits to all shares through a two-level Merkle tree. A Fiat–Shamir challenge χ, hashed from the commitments, batches the multiplication checks (QuickSilver) over the VOLE correlation, and the prover masks its check values; only then does it hash them, together with δ, rt and the masked witness, into the leaf Δ to hide, and it opens $\text{cp}(\Delta)$ for all other leaves. The commitment, masked witness, check values and co-path form a single broadcast, sent in the same round as the sub-tree authentication paths that each verifier receives for the input-wire shares it holds locally.

Verify workflow. In Protocol 16, each verifier re-derives χ and Δ from the broadcast, then reconstructs and checks all opened GGM leaves against the Merkle root. For the single hidden leaf, each verifier recovers the missing shares at the input-wire positions it knows, using its local shares and the masked witness, and verifies them against the sub-tree authentication paths received from the prover. Finally, all verifiers perform the QuickSilver multiplication check to confirm that the circuit relation holds.

QuickSilver multiplication-gate check. For each multiplication gate ℓ with left input $a^{(\ell)}$, right input $b^{(\ell)}$, and output $c^{(\ell)}$ (so the circuit enforces $a^{(\ell)} \cdot b^{(\ell)} = c^{(\ell)}$), denote the corresponding VOLE masks by $v_a^{(\ell)}, v_b^{(\ell)}, v_c^{(\ell)}$. The verifier computes:

$$\begin{aligned} B_\ell &= \mathsf{Commit}(a^{(\ell)})\,\mathsf{Commit}(b^{(\ell)}) \;-\; \mathsf{Commit}(c^{(\ell)})\,\Delta \\ &= \big(\underbrace{a^{(\ell)}b^{(\ell)}}_{=\,c^{(\ell)}} - c^{(\ell)}\big)\,\Delta^2 \;+\; v_a^{(\ell)} v_b^{(\ell)} \\ &\quad +\; \big(a^{(\ell)} v_b^{(\ell)} + b^{(\ell)} v_a^{(\ell)} - v_c^{(\ell)}\big)\,\Delta \\ &\overset{?}{=} v_a^{(\ell)} v_b^{(\ell)} \;+\; \big(a^{(\ell)} v_b^{(\ell)} + b^{(\ell)} v_a^{(\ell)} - v_c^{(\ell)}\big)\,\Delta \end{aligned} \tag{4}$$

When the gate is satisfied, the Δ^2 terms cancel. To batch all M gates, both parties sample a random challenge $\chi \xleftarrow{\$} \mathbb{F}_p$ (via Fiat–Shamir) and check the combined equation

$$V \;=\; \textstyle\sum_\ell \chi^\ell B_\ell \;=\; A_0 + A_1 \cdot \Delta \;\overset{?}{=}\; 0,$$

where $A_0 = \sum_\ell \chi^\ell v_a^{(\ell)} v_b^{(\ell)}$ and $A_1 = \sum_\ell \chi^\ell \big(v_c^{(\ell)} - b^{(\ell)} v_a^{(\ell)} - a^{(\ell)} v_b^{(\ell)}\big)$. Since the prover knows (v_j, w_j) but not Δ, it computes A_0 and A_1 and masks them with a fresh dedicated check-wire VOLE pair (u_0, v_0): it sends $\tilde{A}_0 = A_0 + v_0$ and $\tilde{A}_1 = A_1 + u_0$. The verifier then checks $V + q_0 \overset{?}{=} \tilde{A}_0 + \tilde{A}_1 \cdot \Delta$, where $q_0 = u_0 \cdot \Delta + v_0$ is the check-wire VOLE tag.

Output gate. The last gate $\ell = N - 1$ has no output wire. Since commitments are linear and a public constant c is committed as $c \cdot \Delta$ with mask 0, both parties use

$$\mathsf{Commit}(c^{(N-1)}) \;=\; \textstyle\sum_{k=1}^{N} \mathsf{Commit}(a_k) \;+\; \delta \cdot \Delta,$$

a commitment to $\sum_k a_k + \delta$ with VOLE mask $v_c^{(N-1)} = \sum_{k=1}^{N} v_{a_k}$, which the prover uses in A_1. The check on this gate enforces $w_{N-1} \cdot M_N = \sum_k a_k + \delta$, which binds the public δ to the witness.

Challenge order and soundness. The check values must be fixed before Δ is known: given Δ, any prover could pass by setting $\tilde{A}_0 \leftarrow V + q_0 - \tilde{A}_1 \cdot \Delta$. The prover therefore derives Δ by hashing $(\delta, \mathsf{rt}, \tilde{\mathbf{w}}, \tilde{A}_0, \tilde{A}_1)$ after computing the check values; χ is hashed earlier, from $(\delta, \mathsf{rt}, \tilde{\mathbf{w}})$. For a false witness, $V + q_0 - \tilde{A}_0 - \tilde{A}_1 \cdot \Delta$ is then a fixed polynomial of degree 2 in Δ whose leading coefficient $\sum_\ell \chi^\ell (a^{(\ell)} b^{(\ell)} - c^{(\ell)})$ is nonzero except with probability $M/|\mathbb{F}_p|$ over χ. A cheating prover can place both of its roots among the τ possible values of Δ, so a single repetition has soundness error $2/\tau + M/|\mathbb{F}_p|$.

From single repetition to the full protocol. Protocols 15 and 16 describe a single repetition with soundness error $2/\tau + M/|\mathbb{F}_p|$. Because Δ is hashed from a set of only τ values, a cheating prover can re-randomize its commitments and re-hash offline: modelling H as a random oracle, a prover making q_H queries succeeds with probability $O(q_H((2/\tau)^R + RM/|\mathbb{F}_p|))$ over R repetitions. The repetitions must therefore meet the computational security parameter λ rather than the statistical κ: we run $R = \lceil \lambda/(\log_2 \tau - 1) \rceil$ repetitions, so that $(2/\tau)^R \le 2^{-\lambda}$ (the batching term is negligible over our 384-bit field), with the following adaptations:

(1) **BAVC tree [7].** Naively, R independent GGM trees would cost $R \cdot \log_2 \tau$ co-path hash values in communication. The *Batch All-But-One Vector Commitment* (BAVC) technique replaces them with a single GGM tree of $R{\cdot}\tau$ leaves: each repetition $\rho \in [R]$ corresponds to leaves $\{(\rho-1)\tau + 1, \ldots, \rho\tau\}$, yielding an independent VOLE correlation $(u_j^{(\rho)}, v_j^{(\rho)})$. In our two-level variant, the BAVC applies to the outer Merkle tree over all $R \cdot \tau$ sub-tree roots h_i; the sub-Merkle-tree structure within each leaf remains unchanged.

(2) **Masked witness and challenges.** The prover computes one masked witness $\tilde{\mathbf{w}}^{(\rho)}$ per repetition using the corresponding VOLE masks: $\tilde{w}_j^{(\rho)} = w_j + u_j^{(\rho)}$. The batching challenges $(\chi_\rho)_{\rho \in [R]}$ are hashed as $\mathsf{H}(\delta, \mathsf{rt}, \tilde{\mathbf{w}}^{(1)}, \ldots, \tilde{\mathbf{w}}^{(R)})$. Once all check values $(\tilde{A}_0^{(\rho)}, \tilde{A}_1^{(\rho)})_{\rho \in [R]}$ are computed, all hidden leaves $(\Delta_\rho)_{\rho \in [R]}$ are hashed together from δ, rt, the masked witnesses and the check values of all repetitions, so no repetition can be ground on its own.

(3) **Per-repetition co-paths and checks.** For each repetition ρ, the prover computes its own check values $(\tilde{A}_0^{(\rho)}, \tilde{A}_1^{(\rho)})$ and, once all Δ_ρ are derived, opens $\mathsf{cp}(\Delta_\rho)$. Each verifier independently performs the GGM reconstruction, input binding, and QuickSilver check for every repetition; the proof is accepted only if all R checks pass.

Protocol 15 $\Pi_{\mathsf{VitH.Prove}}$ – VOLEitH Proof Generation

Players: Prover $\mathcal{P}$ (dealer S_i).

Input: Extended witness $\mathbf{w} \in \mathbb{F}_p^L$; a collision resistant hash function H, public statement $\delta \in \mathbb{F}_p$.

`// GGM tree expansion`

1: Generate a GGM tree from sd and obtain τ leaf seeds $\{\mathsf{sd}_i\}_{i \in \tau}$, where $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$.

2: $\forall i \in [\tau]$, derive shares $\{s_j^{(i)}\}_{j=0}^{L}$ and salts $\{r_j^{(i)}\}_{j=0}^{L}$ from sd_i (j=0: check wire); compute $u_j = \sum_i s_j^{(i)}$, $v_j = -\sum_i i \cdot s_j^{(i)}$.

`// Extended witness masking`

3: Compute the masked witness $\tilde{\mathbf{w}} \in \mathbb{F}_p^L$ with $\tilde{\mathbf{w}}_j = \mathbf{w}_j + u_j$ for all $j \in [L]$.

`// Two-level GGM commitment`

4: $\forall i \in [\tau]$, build a sub-Merkle-tree with leaves $\mathsf{H}(j, s_j^{(i)}, r_j^{(i)})$ for $j = 0, \ldots, L$; let h_i be its root.

5: Build the outer Merkle tree over $\{\mathsf{h}_i\}_{i \in [\tau]}$ and derive root commitment rt.

`// Batched multiplication check (QuickSilver)`

6: Derive χ from $\mathsf{H}(\delta, \mathsf{rt}, \tilde{\mathbf{w}})$ via Fiat–Shamir.

7: Compute A_0, A_1 from the gate-by-gate VOLE check (4), batched with challenge χ, using $v_c^{(N-1)} = \sum_k v_{a_k}$ for the output gate.

8: Mask with check-wire VOLE: $\tilde{A}_0 = A_0 + v_0$,

9: $\tilde{A}_1 = A_1 + u_0$.

`// Hidden leaf and opening`

10: Derive $\Delta \in [\tau]$ from $\mathsf{H}(\delta, \mathsf{rt}, \tilde{\mathbf{w}}, \tilde{A}_0, \tilde{A}_1)$ via Fiat–Shamir.

11: Open $\mathsf{cp}(\Delta)$ (outer tree).

Broadcast: $\pi_{\mathsf{bc}} = (\mathsf{rt}, \tilde{\mathbf{w}}, \tilde{A}_0, \tilde{A}_1, \mathsf{cp}(\Delta))$.

$\mathcal{P}$ **sends to each** S_j: the salts and sub-tree authentication paths in leaf Δ for every input-wire position that S_j holds locally.

Proof size and complexity. For $R = \lceil \lambda/(\log_2 \tau - 1) \rceil$ repetitions over $\mathbb{F}_p$ with $L = 2N + M - 1$ witness wires ($M = N - 1$ multiplication gates, the last of which has no output wire) and $d = \lceil \log_2 \tau \rceil$:

Protocol 16 $\Pi_{\mathsf{VitH.Verify}}$ – VOLEitH Proof Verification

Players: Verifier S_j, $j \in [n] \setminus \{i\}$.

Input: Broadcast proof π_{bc} from $\mathcal{P}$; P2P proof π_j for each S_j; public $\delta, \mathbb{F}, \mathsf{H}$. Each S_j holds its local shares $\{(a_T, m_T)\}$ for every subset T it belongs to.

`// Challenges`

Re-derive χ from $\mathsf{H}(\delta, \mathsf{rt}, \tilde{\mathbf{w}})$ and Δ from $\mathsf{H}(\delta, \mathsf{rt}, \tilde{\mathbf{w}}, \tilde{A}_0, \tilde{A}_1)$.

`// GGM reconstruction`

Reconstruct $\{\mathsf{sd}_i\}_{i \neq \Delta}$ from the GGM sibling seeds $(\mathsf{sd}_1^*, \ldots, \mathsf{sd}_d^*)$ in $\mathsf{cp}(\Delta)$.

$\forall i \neq \Delta$: expand sd_i to obtain shares $\{s_j^{(i)}\}_{j=0}^{L}$ and salts $\{r_j^{(i)}\}_{j=0}^{L}$.

`// Input binding and commitment check for hidden leaf Δ`

for *each* S_j **do**

  for *each input wire* k *that* S_j *holds locally* **do**

    Compute $u_k = \tilde{w}_k - w_k^{\text{local}}$ (VOLE mask from masked witness and local share).

    Recover hidden-leaf share: $s_k^{(\Delta)} = u_k - \sum_{i \neq \Delta} s_k^{(i)}$.

    Verify $\mathsf{H}(k, s_k^{(\Delta)}, r_k^{(\Delta)})$ against h_Δ using the salt and sub-tree authentication path from π_j.

  end

end

Verify h_Δ against rt using the Merkle sibling hashes $(\mathsf{h}_1^*, \ldots, \mathsf{h}_d^*)$ in $\mathsf{cp}(\Delta)$.

`// VOLEitH multiplication check (QuickSilver)`

Re-derive VOLE tags: $q_j = \sum_{i \neq \Delta} s_j^{(i)} (\Delta - i)$ for all wires j.

Compute wire commitments $\mathsf{Commit}(w_j) = \tilde{w}_j \cdot \Delta - q_j$ and the output-gate commitment $\mathsf{Commit}(c^{(N-1)}) = \sum_k \mathsf{Commit}(a_k) + \delta \cdot \Delta$.

Evaluate the batched gate check (4): $V = \sum_\ell \chi^\ell B_\ell$.

Compute check-wire tag $q_0 = \sum_{i \neq \Delta} s_0^{(i)} (\Delta - i)$; check $V + q_0 \stackrel{?}{=} \tilde{A}_0 + \tilde{A}_1 \cdot \Delta$.

Output: Accept iff all checks pass.

- **Broadcast proof size** (per proof): root commitment rt (2λ bits via BAVC); masked witnesses $\tilde{\mathbf{w}}^{(1)}, \ldots, \tilde{\mathbf{w}}^{(R)}$ (RL field elements); check values $\tilde{A}_0, \tilde{A}_1$ ($2R$ field elements); co-paths (Rd GGM sibling seeds and Rd Merkle sibling hashes).
- **P2P proof size** (per verifier S_j): R sub-tree authentication paths for input-wire shares, each of depth $\lceil \log_2(L+1) \rceil$ hashes, with a λ-bit salt.
- **Prover:** $O(R\tau L)$ hash evaluations, $O(R\tau)$ PRG evaluations, $O(RM)$ field operations.
- **Verifier:** $O(R\tau)$ PRG evaluations, $O(R(L+d))$ hash evaluations, $O(RM)$ field operations.

In summary, substituting $R = O(\lambda)$, $L = O(N)$, $M = N-1$, and $d = O(1)$, a single invocation of Π_{VitH} has broadcast cost $O(\lambda N)$ field elements plus $O(\lambda^2)$ bits, P2P cost $O(\lambda \log N)$ hashes per verifier, prover computation $O(\lambda N)$ hashes, and verifier computation $O(\lambda N)$ hashes plus $O(\lambda N)$ field operations. When invoked $t+1$ times in Π_{ZKPGen}, the total broadcast is $O(t\lambda N)$ and the total P2P per verifier is $O(t\lambda \log N)$ hashes—all from efficient symmetric-key primitives. Each proof is non-interactive and sent in a single round.

C.2 Full Ligero Protocol

We now give the full description of Π_{Lig} (cf. §5.2.3) for B instances with N share pairs each.

Layout. The prover groups the instances into $\ell = \lceil B/c \rceil$ segments of c instances each, filling the last segment with dummy instances ($M = 1$, $a = 0$, $\delta = 1$), which satisfy every constraint. A segment σ has four rows of $L = cN$ entries: the shares M_σ and a_σ, the running products p_σ and the running sums s_σ of its instances, with its ν-th instance at positions $\nu N, \ldots, \nu N + N - 1$. We write $W = \{0, \ldots, L-1\}$ for the positions, $S = \{\nu N\}_{\nu < c}$ for the instance starts, and $E = \{\nu N + N - 1\}_{\nu < c}$ for the instance ends. Sharing rows keeps opened columns short; how c is chosen is part of the parameters below.

Encoding. Let $H = \langle \eta \rangle \subset D = \langle \omega \rangle$ be multiplicative subgroups of $\mathbb{F}_p^*$ of sizes k' and n_c (powers of two), where $k' \geq L + 2q$, $n_c \geq 4k'$, and q is the number of points the verifiers will sample. The polynomial f_r of a row r has degree $< k'$, takes the row's L entries at $\eta^0, \ldots, \eta^{L-1}$, and takes uniformly random values at the remaining $k' - L \geq 2q$ points of H; its codeword $\hat{W}_r$ is its evaluation on D. The committed matrix $\hat{W}$ also contains a mask row $Z_j = Z_W \cdot g_j$ for each verifier S_j, where $Z_W(x) = \prod_{i \in W}(x - \eta^i)$ and g_j is uniform of degree $< 2k' - 1 - L$, and one blinding row Z_0, uniform of degree $< K^* \coloneqq 2k' - 1$. Its n_c columns, each hashed with a fresh λ-bit salt, are Merkle-hashed into $\mathsf{rt}_{\hat{W}}$; the verifiers only ever see columns at points of $D \setminus H$, with their salts.

Constraints. For a set $A \subseteq W$, let Z_A be the vanishing polynomial of $\{\eta^i\}_{i \in A}$, and say that a polynomial vanishes *on* A if it vanishes at η^i for all $i \in A$. For each segment σ, the relations $\delta^{(b)} = p_N^{(b)} - s_N^{(b)}$ of its instances become five constraints:

(1) $h_1^\sigma(x) = f_{p_\sigma}(\eta x) - f_{p_\sigma}(x) \cdot f_{M_\sigma}(\eta x)$ (on $W \setminus E$)
(2) $h_2^\sigma(x) = f_{s_\sigma}(\eta x) - f_{s_\sigma}(x) - f_{a_\sigma}(\eta x)$ (on $W \setminus E$)
(3) $h_3^\sigma(x) = f_{p_\sigma}(x) - f_{M_\sigma}(x)$ (on S)
(4) $h_4^\sigma(x) = f_{s_\sigma}(x) - f_{a_\sigma}(x)$ (on S)
(5) $h_5^\sigma(x) = f_{p_\sigma}(x) - f_{s_\sigma}(x) - \Delta_\sigma(x)$ (on E)

Here Δ_σ is the public polynomial of degree $< c$ that takes the δ of each instance of the segment at that instance's end. Let

$$\begin{aligned} \mathcal{N}^\sigma(x) \;=\; & Z_E(x)\left[h_1^\sigma(x) + \alpha\, h_2^\sigma(x)\right] \\ & +\; Z_{W \setminus S}(x)\left[\alpha^2 h_3^\sigma(x) + \alpha^3 h_4^\sigma(x)\right] \\ & +\; \alpha^4 Z_{W \setminus E}(x)\, h_5^\sigma(x), \end{aligned}$$

and $\mathcal{N}(x) = \sum_{\sigma=1}^{\ell} \alpha^{5(\sigma-1)} \mathcal{N}^\sigma(x)$. Each term of $\mathcal{N}^\sigma$ vanishes on W exactly when its constraints hold, so Z_W divides $\mathcal{N}$ when all constraints hold, and otherwise fails to divide it except with probability $5\ell/|\mathbb{F}_p|$ over α. The quotient $C = \mathcal{N}/Z_W$ has degree $< 2k' + c - 1 - L \leq K^*$. The prover commits the salted pairs $(C(y), Z_C(y))$ for $y \in D$ into rt_C, where Z_C is a second blinding row, uniform of degree $< K^*$.

Input binding (our contribution). Verifier S_j holds the shares $(M_k^{(b)}, a_k^{(b)})$ for k in a set $K_j^{(b)} \subseteq [N]$; let $K_{j,\sigma} \subseteq W$ be the positions of these shares within segment σ. After both commitments, the prover sends S_j the *binding polynomial*

$$q_j(x) \;=\; \sum_\sigma \Lambda_{j,\sigma}(x)\left(\gamma_{j,\sigma,M}\, f_{M_\sigma}(x) + \gamma_{j,\sigma,a}\, f_{a_\sigma}(x)\right) + Z_j(x)$$

of degree $< 2k' - 1$, and broadcasts its digest $h_j = \mathsf{H}(q_j)$. Here γ_j are hashed challenges, and $\Lambda_{j,\sigma}$ is the polynomial of degree $< k'$ that equals the hashed challenge $\beta_{j,i}$ at η^i for $i \in K_{j,\sigma}$ and 0 on the rest of H. Since Z_j vanishes on W, S_j can compute $q_j(\eta^i) = \beta_{j,i} \sum_{\sigma: i \in K_{j,\sigma}} (\gamma_{j,\sigma,M} M_{\sigma,i} + \gamma_{j,\sigma,a}\, a_{\sigma,i})$ from its own shares for every position i it holds; it checks these values and h_j, and it checks q_j against the opened columns at every sampled point. The first check pins the committed rows to S_j's shares, and the second pins q_j to the committed codewords. Every share position is held by at least $n - t - 1 \geq t$ servers other than the dealer, so if the dealer is corrupted, some honest server checks every position.

Challenges. All challenges are derived by hashing, so Π_{Lig} is non-interactive and takes a single round. The challenge α is hashed from $\mathsf{rt}_{\hat{W}}$ and the $\delta^{(b)}$, and the coefficients $\mathbf{v}$ of the consistency polynomial u and (γ_j, β_j) from $\mathsf{rt}_{\hat{W}}$, rt_C and the $\delta^{(b)}$. These range over $\mathbb{F}_p$: a prover making q_H hash queries gains only $O(q_H(n_c + \ell + n)/|\mathbb{F}_p|)$, which is negligible for our 384-bit field. The q sampled points are hashed from $\mathsf{rt}_{\hat{W}}$, rt_C, u, the digests $\{h_j\}_j$ and the $\delta^{(b)}$, so every message the checks depend on is fixed before the points are known. They range over only $n_c - k'$ values, however, so a prover can re-randomize its commitments and re-hash offline, multiplying its chance of passing by up to q_H; the parameters below therefore bound that chance by $2^{-\lambda}$ rather than $2^{-\kappa}$.

Soundness and parameters. Let $T = n_c - k'$ and fix an integer e with $4e < T - K^* + 1$. First suppose the committed rows, including C and Z_C and restricted to the T columns in $D \setminus H$, are e-far from every matrix whose rows are evaluations of polynomials of degree $< K^*$. By the proximity lemma of Ligero [5], a random linear combination of the rows is then e-far from that code, except with probability $O(n_c/|\mathbb{F}_p|)$. So u differs from that combination at more than e points of $D \setminus H$, and each sampled point avoids all of them with probability at most $1 - e/T$. Otherwise, the rows agree with polynomials $\tilde{f}_r$ of degree $< K^*$ outside a common set of at most e columns. Suppose the witness these polynomials encode on W violates a constraint, or differs from an honest verifier's shares at a position it holds. Then, except with probability $O((\ell + n)/|\mathbb{F}_p|)$ over the hashed challenges, the constraint identity $Z_W\, C = \mathcal{N}$ or that verifier's binding identity fails as a polynomial identity of degree at most $2K^* + c - 2$. A sampled point x passes the check only if x is one of its roots, or if x or ηx is among the e corrupted columns. Hence a false claim is accepted with probability at most

$$\left(1 - \frac{e}{T}\right)^q \;+\; \left(\frac{2K^* + c - 2 + 2e}{T}\right)^q \;+\; O\!\left(\frac{n_c + \ell + n}{|\mathbb{F}_p|}\right).$$

The layout and parameters follow from this bound. For each power of two k', we search n_c (a power of two with $n_c \geq 4k'$ dividing $p - 1$, smallest first) and q (smallest first), with $c = \min(B, \lfloor (k' - 2q)/N \rfloor)$, until the first two terms are at most $2^{-\lambda}$ for an admissible e. Among the candidates for the different k', we choose the one sending the fewest field elements (see the complexity below). The sampling budget q does not depend on N or B: zero knowledge comes from the $k' - L \geq 2q$ padding values, not from limiting q.

Zero knowledge. The corrupted verifiers see the opened columns at x and ηx for the q sampled points (at most $2q$ positions of $D \setminus H$), C and Z_C at the sampled points, u, all digests h_j, and their own binding polynomials. Each data row has $k' - L \geq 2q$ uniform padding values, so its values at any $2q$ points outside H are uniform; each mask row and the blinding row keep at least $2q$ degrees of freedom. $C(x)$ is determined by the opened values at x and ηx. The value $u(\eta x)$ involves $C(\eta x)$, which depends on unopened values at $\eta^2 x$; this is why u also contains Z_C, which is opened only at the points x. After the openings, Z_0 and Z_C make u uniform subject to its values at the points x, which the openings determine. Likewise, q_j is determined on W by S_j's shares (it is 0 at the positions S_j does not hold) and at x and ηx by the openings, and g_j leaves exactly enough freedom to make q_j uniform subject to these values. The salts make the Merkle hashes of unopened columns independent of their contents in the random-oracle model. The digest h_j of an honest verifier's q_j is simulated by programming the random oracle, since g_j leaves q_j at least $k' - 1$ uniform degrees of freedom beyond these values. A simulator therefore samples all these values uniformly subject to the consistency equations, using only the corrupted servers' shares.

Protocol 17 $\Pi_{\mathsf{Lig.Prove}}$ – Ligero Proof Generation (B instances)

Players: Prover $\mathcal{P}$ (dealer S_i) and verifiers S_j, $j \in [n] \setminus \{i\}$.

Input: Private: $\{(M_k^{(b)}, a_k^{(b)})\}_{k \in [N]}$ for $b \in [B]$. Public: $\{\delta^{(b)}\}_{b \in [B]}$; the layout (c, ℓ, L, k', n_c, q) with subgroups $H = \langle \eta \rangle \subset D = \langle \omega \rangle$; for each S_j and b, the positions $K_j^{(b)} \subseteq [N]$ whose shares S_j holds.

`// Lay out, encode and commit`

1: Group the instances into ℓ segments of c, padding the last segment with dummy instances.
2: **foreach** *segment* σ *and row* $r \in \{M_\sigma, a_\sigma, p_\sigma, s_\sigma\}$ **do**
3: Let f_r have degree $< k'$, take the row's L entries at $\eta^0, \ldots, \eta^{L-1}$ and uniformly random values on the rest of H.
4: **end**
5: **foreach** *verifier* S_j **do**
6: Set $Z_j = Z_W \cdot g_j$ with g_j uniform of degree $< 2k' - 1 - L$.
7: **end**
8: Sample Z_0 uniformly of degree $< K^*$.
9: Evaluate all rows on D and Merkle-commit the columns, each with a fresh λ-bit salt $\to \mathsf{rt}_{\hat{W}}$.

`// Composition`

10: Hash α from $(\mathsf{rt}_{\hat{W}}, \{\delta^{(b)}\}_b)$; form $\mathcal{N}$ and $C = \mathcal{N}/Z_W$.
11: Sample Z_C uniformly of degree $< K^*$; evaluate C and Z_C on D and Merkle-commit the pairs $(C(y), Z_C(y))$, each with a fresh salt $\to \mathsf{rt}_C$.

`// Consistency and binding polynomials`

12: Hash $\mathbf{v}$ and each (γ_j, β_j) from $(\mathsf{rt}_{\hat{W}}, \mathsf{rt}_C, \{\delta^{(b)}\}_b)$.
13: Broadcast $\mathsf{rt}_{\hat{W}}$, rt_C and $u = \sum_r v_r f_r + v_C\, C + v_Z\, Z_C$.
14: **foreach** *verifier* S_j **do**
15: Send S_j its binding polynomial q_j and broadcast $h_j = \mathsf{H}(q_j)$.
16: **end**

`// Sample and open`

17: Hash q distinct points $Q \subset D \setminus H$ from $(\mathsf{rt}_{\hat{W}}, \mathsf{rt}_C, u, \{h_j\}_j, \{\delta^{(b)}\}_b)$.
18: For each $x \in Q$, broadcast the committed columns at x and ηx and $(C(x), Z_C(x))$, with their salts and Merkle paths.

Protocol 18 $\Pi_{\text{Lig.Verify}}$ – Ligero Proof Verification (B instances)

Players: Verifier S_j, $j \in [n] \setminus \{i\}$.

Input: The broadcast $(\mathrm{rt}_{\hat{W}}, \mathrm{rt}_C, u, \{h_j\}_j)$ with the openings, and the binding polynomial q_j; public values as in Protocol 17. S_j holds $(M_k^{(b)}, a_k^{(b)})$ for $k \in K_j^{(b)}$.

// Challenges and openings

1: Check $h_j = \mathsf{H}(q_j)$; re-derive α, $\mathbf{v}$, (γ_j, β_j) and Q; check $\deg u < K^*$ and $\deg q_j < 2k' - 1$.

2: Verify the Merkle paths of all opened columns against $\mathrm{rt}_{\hat{W}}$ and of all $(C(x), Z_C(x))$ against rt_C.

3: **foreach** $x \in Q$ **do**

// Proximity

4: Check $u(x) = \sum_r v_r \hat{W}_r(x) + v_C\, C(x) + v_Z\, Z_C(x)$.

// Constraints

5: Compute $\mathcal{N}(x)$ from the opened values at x and ηx and the public $\Delta_\sigma(x)$; check $Z_W(x)\, C(x) = \mathcal{N}(x)$.

// Input binding

6: Check $q_j(x) = \sum_\sigma \Lambda_{j,\sigma}(x)\big(\gamma_{j,\sigma,M} \hat{W}_{M_\sigma}(x) + \gamma_{j,\sigma,a} \hat{W}_{a_\sigma}(x)\big) + \hat{W}_{Z_j}(x)$.

7: **end**

8: **foreach** *position i that S_j holds in some segment* **do**

9: Check $q_j(\eta^i) = \beta_{j,i} \sum_{\sigma:\, i \in K_{j,\sigma}} \big(\gamma_{j,\sigma,M} M_{\sigma,i} + \gamma_{j,\sigma,a}\, a_{\sigma,i}\big)$.

10: **end**

Output: Accept iff all checks pass.

Complexity of Π_{Lig}. One invocation over B instances commits $4\ell + n$ rows to $\mathrm{rt}_{\hat{W}}$ and the two rows C, Z_C to rt_C, where $\ell = \lceil B/c \rceil$.

- **Communication.** In its single round, the prover broadcasts two roots, u ($K^* = 2k' - 1$ field elements) and $n - 1$ digests, and sends each verifier its binding polynomial ($2k' - 1$ field elements). The openings add at most $2q$ columns of $4\ell + n$ field elements, $2q$ values of C and Z_C, and $O(q \log n_c)$ hashes and salts. In total this is $O(q(B/c + n) + nk')$ field elements with $k' = O(cN + q)$. Choosing $c \approx \sqrt{qB/(nN)}$ balances the two terms, so with $q = O(\lambda)$ the proof has $O(\sqrt{\lambda nNB} + \lambda n)$ field elements and grows with $\sqrt{B}$.
- **Prover.** RS encoding of the rows and the composition: $O((\ell + n)\, n_c \log n_c)$ field operations with $n_c = O(k')$, plus $O(n_c)$ hashes.
- **Verifier.** $O(q(B + cN))$ field operations and $O(q \log n_c)$ hashes.

In Π_{ZKPGen}, each of the $t + 1$ dealers runs one invocation over all of its instances.

D Deferred Proofs

This appendix proves Theorems 5.2 and 5.3 and Lemmas 5.4 and 5.5 for the offline phase, and Lemmas 6.1 and 6.2 and Theorem 6.4 for the online phase.

D.1 Proof of Theorem 5.2

Proof of Theorem 5.2. Let $C \subset [n]$ with $|C| \le t$ be the corrupted set, and fix $T^* \in \mathcal{T}$ with $T^* \supseteq C$. We first assume that all values derived from replicated keys and pairwise seeds are uniformly random; replacing them by PRF and PRG outputs changes every distribution below by at most the corresponding distinguishing advantages.

Simulator. $\mathcal{S}$ emulates $\mathcal{F}_{\text{Setup}}$, $\mathcal{F}_{\text{Coin}}$ and $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ toward the adversary and plays the honest servers following Π_{DegGen}. Since it knows every input to the emulated $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$, it evaluates the checked relation exactly as the functionality would. If the emulated check rejects or the adversary aborts, $\mathcal{S}$ sends abort to $\mathcal{F}_{\text{RandExp}}$. Otherwise it computes the components of $[\![q_N]\!]$ held by corrupted servers as the honest servers hold them, from the emulated keys and the broadcast corrections, sends them to $\mathcal{F}_{\text{RandExp}}$, and replies continue.

Messages. In the invocation of Π_{DegMul} with target $\mathcal{A}_j = [n] \setminus T_j$, the adversary receives $d_i = [av]_i - [r]_i$ from each honest $S_i \in \mathcal{A}_j$ when S_{agg} is corrupted, and the broadcast $\delta = av - r$. By the privacy of $\Pi^{\text{deg}}_{\text{DoubleRand}}$ (§5.1.1), the honest servers' additive shares $([r]_i)_{i \in \mathcal{A}_j \setminus C}$ are jointly uniform in the adversary's view, and each invocation uses a fresh counter. Hence these messages are jointly uniform subject only to the linear relations the adversary can compute from its own shares, independently of the honest servers' inputs, and the simulated messages are identically distributed.

Correctness. All honest servers hold consistent shares, because every correction δ is broadcast. $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ reconstructs from the honest servers' shares, and the random combination $\sum_{j=2}^{N} \rho_j (q_j - q_{j-1} M_{T_j})$ with coefficients from $\mathcal{F}_{\text{Coin}}$ is nonzero except with probability $1/|\mathbb{F}_p|$ whenever some residual is nonzero. Therefore, except with probability $1/|\mathbb{F}_p|$, acceptance implies $q_N = \prod_{T \in \mathcal{T}} M_T = \big(\prod_{T \in \mathcal{T}} m_T\big)^e$, and rejection leads to abort in both executions.

Output distribution. No corrupted server holds m_{T^*}, and by the previous paragraphs the adversary's view is independent of it. Given that view, $\alpha := \prod_{T \in \mathcal{T}} m_T = m_{T^*} \cdot \prod_{T \ne T^*} m_T$ is therefore uniform in $\mathbb{F}_p$ unless one of the other $N - 1$ components is zero, which happens with probability at most $(N-1)/|\mathbb{F}_p|$. For the honest servers' outputs, consider the last invocation of Π_{DegMul} and let $\mathcal{H}$ be the honest servers in its target set $\mathcal{A}_N$. Each component r_T with $T \supseteq C$ of its double sharing is unknown to the adversary; it enters the output component of $[\![q_N]\!]$ indexed by T with coefficient $+1$ and, since $P_{\mathcal{A}_N}(T) \in \mathcal{H}$, the honest message $d_{P_{\mathcal{A}_N}(T)}$ with coefficient -1, while the pairwise PRG term of each honest pair in $\mathcal{H}$ enters two honest messages with opposite signs. These directions span every vector of coordinate sum zero over the honest messages and the honest-only output components, so these values are jointly uniform subject to a single linear relation whose right-hand side is q_N minus values the adversary knows. This is exactly how $\mathcal{F}_{\text{RandExp}}$ samples the honest servers' components once the corrupted servers' components are fixed.

Combining these steps, the joint distribution of the adversary's view and the honest servers' outputs differs between the real and ideal executions by at most $(N-1)/|\mathbb{F}_p| + 1/|\mathbb{F}_p| = N/|\mathbb{F}_p|$. □

D.2 Proof of Theorem 5.3

Proof of Theorem 5.3. Let C be the corrupted set, $|C| \le t$. Since $|\mathcal{A}| = t + 1$, some dealer $S_{i^*} \in \mathcal{A}$ is honest; fix $T^* \supseteq C$. As before we first treat seed-derived values as uniformly random.

Simulator. $\mathcal{S}$ emulates $\mathcal{F}_{\text{Setup}}$, $\mathcal{F}_{\text{DoubleRand}}$, $\mathcal{F}_{\text{Coin}}$, $\mathcal{F}_{\text{DualShareZKP}}$ and $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$, and plays the honest servers following Π_{ZKPGen}. It

sends abort to $\mathcal{F}_{\mathsf{RandExp}}$ whenever an emulated functionality rejects or the adversary aborts; otherwise it sends the components of $[\![q_{t+1}]\!]$ held by corrupted servers, computed as the honest servers hold them, and replies continue.

Honest dealers. For S_{i^*}, the pair (m_{T^*}, a_{T^*}) is derived from sd_{i^*,T^*}, which only S_{i^*} and the servers in $[n] \setminus T^* \subseteq [n] \setminus C$ hold, so it is uniform in the adversary's view. The sum A^{i^*} contains the uniform summand a_{T^*}, so the broadcast $\delta_{i^*} = M^{i^*} - A^{i^*}$ is uniform and independent of M^{i^*}. Because the dealer is honest, $\mathcal{F}_{\mathsf{DualShareZKP}}$ outputs accept unless the adversary aborts, and it leaks only the dealer's values for pairs that corrupted servers already hold.

Corrupted dealers. For a corrupted $S_i \in \mathcal{A}$, every $T \in \mathcal{T}$ has an honest holder in $[n] \setminus T$, since $n - t > t$. $\mathcal{F}_{\mathsf{DualShareZKP}}$ therefore accepts only if the dealer's values (M_T, a_T) equal (m_T^e, a_T) for the setup-determined pairs held by honest servers and $\delta_i = \prod_T m_T^e - \sum_T a_T$; in that case $[\![M^i]\!] = \delta_i + [\![A^i]\!]$ shares $M^i = \prod_T m_T^e$.

Multiplication tree. $\Pi_{\mathsf{RSS.Mul}}$ reveals only values masked by fresh double sharings from $\mathcal{F}_{\mathsf{DoubleRand}}$, which are uniform in the adversary's view, and the honest servers' shares stay consistent because corrections are delivered over the broadcast channel. If the adversary causes $q_j \neq q_{j-1} M^{i_j}$ for some j, the random combination checked by $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$ is nonzero except with probability $1/|\mathbb{F}_p|$, and both executions abort.

Output distribution. On acceptance, $q_{t+1} = \prod_{i \in \mathcal{A}} M^i$, where $M^{i^*} = m_{T^*}^e \cdot \prod_{T \neq T^*} m_T^e$ and m_{T^*} is uniform and independent of the adversary's view. Unless one of the other $(t+1)N - 1$ derived components m_T is zero, which happens with probability at most $((t+1)N - 1)/|\mathbb{F}_p|$, the product equals $m_{T^*}^e \cdot c$ for a nonzero e-th power c independent of m_{T^*}. The map $x \mapsto x^e$ sends a uniform element of $\mathbb{F}_p$ to 0 with probability $1/|\mathbb{F}_p|$ and otherwise to a uniform nonzero e-th power, and multiplication by c permutes the nonzero e-th powers, so the shared value is distributed exactly as α^e for $\alpha \xleftarrow{\$} \mathbb{F}_p$. The honest servers' output components indexed by $T \supseteq C$ contain the honest-only components of the last double sharing and, by the argument in the proof of Theorem 5.2, are jointly uniform with the adversary's view subject to the shared value. The two executions therefore differ by at most $((t+1)N - 1)/|\mathbb{F}_p| + 1/|\mathbb{F}_p| = (t+1)N/|\mathbb{F}_p|$. □

D.3 Proof of Lemma 5.4

Proof of Lemma 5.4. Let C, $|C| \leq t$, be the corrupted set and S_i the dealer. The simulator $\mathcal{S}$ answers the adversary's oracle queries with fresh uniform values, records them, and programs the oracle at the points named below. Every server sees the same broadcast and aborts unless all verdicts accept, so it suffices to simulate the adversary's view together with the common verdict. For an honest dealer we first treat the GGM expansion and the leaf expansions as truly random; each repetition hides one leaf behind $\lceil \log_2 \tau \rceil$ PRG evaluations and one expansion, so the PRG costs at most $R(\lceil \log_2 \tau \rceil + 1)\, \epsilon_{\mathsf{PRG}}$.

Honest dealer: simulator. $\mathcal{S}$ receives b and the dealer's pairs for all T held by a corrupted server, and hence the input wires that corrupted servers hold. In each repetition ρ it samples $\Delta_\rho \xleftarrow{\$} [\tau]$, generates every other leaf honestly, and samples $\tilde{\mathbf{w}}^{(\rho)}$ uniformly. In the sub-Merkle tree of the hidden leaf, at an input wire k held by a corrupted server it sets $s_k^{(\Delta_\rho)} = \tilde{w}_k - w_k - \sum_{i \neq \Delta_\rho} s_k^{(i)}$ with a fresh salt and hashes honestly; at every other position it uses a uniform 2λ-bit string. It builds rt, samples $\chi_\rho, \tilde{A}_1^{(\rho)} \xleftarrow{\$} \mathbb{F}_p$, computes V and q_0 as a verifier does, and sets $\tilde{A}_0^{(\rho)} = V + q_0 - \tilde{A}_1^{(\rho)} \Delta_\rho$. It programs the two hashes of the protocol to return $(\chi_\rho)_\rho$ and $(\Delta_\rho)_\rho$, sends the broadcast and the corrupted servers' salts and paths, and replies continue to $\mathcal{F}_{\mathsf{DualShareZKP}}$ unless a corrupted server broadcasts reject.

Honest dealer: indistinguishability. In the real proof the hidden leaf's shares are uniform, so every $\tilde{w}_j = w_j + u_j$ is uniform, and so is $\tilde{A}_1 = A_1 + u_0$; the honest $\tilde{A}_0$ satisfies the verification equation, which fixes it as in the simulation. The adversary can recompute the hidden leaf's share only at the input wires it holds, where both executions agree. Every other leaf of the hidden sub-tree hashes a uniform salt unknown to the adversary, which it queries with probability at most $q_H/2^\lambda$. The programmed inputs contain rt and hence fresh uniform strings, so programming fails with probability at most $q_H/2^{2\lambda}$. By completeness, the honest servers accept in both executions exactly when $b = 1$ and no corrupted server rejects: an honest verifier whose pair differs from the dealer's recovers a wrong share and rejects.

Corrupted dealer: extraction. Call leaf i of repetition ρ *extractable* if the recorded queries contain a path of preimages from rt down to all $L + 1$ salted shares of the leaf. Without an oracle collision, which occurs with probability at most $q_H^2/2^{2\lambda}$, the extracted shares $s_j^{(i)}$ are unique. An opened leaf must hash to its committed sub-tree root, so a repetition with two non-extractable leaves passes with probability $O(q_H/2^{2\lambda})$, and one with a single non-extractable leaf passes only if Δ_ρ is that leaf. If all leaves of repetition ρ are extractable, set $u_j = \sum_i s_j^{(i)}$, $v_j = -\sum_i i \cdot s_j^{(i)}$ and the effective witness $w_j = \tilde{w}_j - u_j$, all fixed before χ_ρ and Δ_ρ are derived. Since $\sum_{i \neq \Delta} s_j^{(i)} (\Delta - i) = u_j \Delta + v_j$ for every $\Delta \in [\tau]$, the verifiers' commitments are commitments to the effective witness. $\mathcal{S}$ inputs to $\mathcal{F}_{\mathsf{DualShareZKP}}$ the input wires (M_T, a_T) of the effective witness of the first repetition whose leaves are all extractable, or arbitrary values if there is none.

Corrupted dealer: soundness. Fix a repetition whose leaves are all extractable. (i) Suppose the effective witness differs at an input wire k from the value held by an honest holder S_h, which exists since $n - 2t \geq 1$. Then the share S_h recovers differs from the extracted $s_k^{(\Delta)}$ for every Δ, and its check fails except with probability $O(q_H/2^{2\lambda})$. (ii) Otherwise, if $b = 0$, the honest servers' inputs make the statement false, so the effective witness violates a gate; recall that the last gate enforces $w_{N-1} M_N = \sum_k a_k + \delta$. Then $V + q_0 - \tilde{A}_0 - \tilde{A}_1 \Delta$ is a polynomial of degree at most 2 in Δ whose leading coefficient $\sum_\ell \chi_\rho^\ell (a^{(\ell)} b^{(\ell)} - c^{(\ell)})$ vanishes with probability at most $(N-1)/|\mathbb{F}_p|$ over χ_ρ; otherwise the check passes for at most two values of Δ_ρ. Honest servers that were given different values of δ derive their challenges from different oracle inputs and reject as in case (ii). So, given good batching challenges, each repetition passes with probability at most $2/\tau$ over Δ_ρ in every case, and since one oracle output yields all Δ_ρ, all R repetitions pass with probability at most $(2/\tau)^R$ per query. A query yields a bad batching challenge with probability at most $R(N-1)/|\mathbb{F}_p|$. Summing over the q_H queries and the final output, the honest servers accept while $b = 0$ with probability at most $(q_H + 1)\big((2/\tau)^R + R(N-1)/|\mathbb{F}_p|\big) + O(q_H^2/2^\lambda)$.

Corrupted dealer: simulation. $\mathcal{S}$ runs every check that does not use the honest servers' shares, checks the salts and paths sent to each honest server against the extracted shares, and replies continue exactly when all of these pass and no corrupted server rejects. If $b = 1$, the effective witness agrees with the honest holders' values, so the honest servers' own checks agree with $\mathcal{S}$'s; if $b = 0$, both executions reject except with the probability above. The adversary's view consists of its own messages and the oracle answers, which $\mathcal{S}$ reproduces exactly. Adding the three parts gives the bound of Lemma 5.4. □

D.4 Proof of Lemma 5.5

Proof of Lemma 5.5. Let C, $|C| \le t$, be the corrupted set and S_i the dealer. $\mathcal{S}$ handles oracle queries as in the proof of Lemma 5.4, and the verdict broadcast again makes the verdict common to all servers.

Honest dealer: simulator. $\mathcal{S}$ receives b and the dealer's pairs held by corrupted servers, which fix the witness entries that corrupted servers hold. It samples α, $\mathbf{v}$, $(\gamma_j, \beta_j)_j$ and the points Q itself. For every $x \in Q$ it samples the committed columns at x and ηx and the value $Z_C(x)$ uniformly, and computes $C(x)$ from them by the constraint identity. It samples u uniformly of degree $< K^*$ subject to the consistency check at every $x \in Q$. For each corrupted verifier S_j it samples q_j uniformly of degree $< 2k' - 1$ subject to the binding check at every $x \in Q$ and to its values on W, which S_j's shares determine. Unopened Merkle leaves, the roots and the digests of honest verifiers' q_j are built from uniform strings. $\mathcal{S}$ programs the oracle at the four challenge derivations and replies continue unless a corrupted server rejects.

Honest dealer: indistinguishability. (1) A data row is determined by its L entries and $k' - L \ge 2q$ uniform padding values on H. The Lagrange coefficients that map the padding values to the row's values at $2q$ distinct points outside H form, up to nonzero row and column scalings, a Cauchy matrix, which has full rank; the opened data values are therefore uniform and independent of the witness. (2) $Z_j = Z_W g_j$, where Z_W has no root outside H and g_j is uniform of degree $< 2k' - 1 - L$, a number of coefficients at least $2q$; its opened values are uniform. Moreover $q_j - Z_j$ is determined by the data rows, and conditioned on the $2q$ opened values, $Z_W g_j$ ranges over all polynomials of degree $< 2k' - 1$ that vanish on W and at the opened points. So q_j is uniform subject to its values on W and at the opened points, exactly as sampled. (3) Z_0 and Z_C are uniform of degree $< K^*$ and independent of all other rows. Z_0 is opened at $2q$ points and Z_C only at the points $x \in Q$, so unless $v_0 v_Z = 0$, which happens with probability at most $2/|\mathbb{F}_p|$, u is uniform subject to its values at these points, and the openings determine those values. In particular $u(\eta x)$, which involves the unopened $C(\eta x)$, is masked by $Z_C(\eta x)$. (4) $C(x)$ is fixed by the constraint identity, which an honest proof satisfies. (5) Every unopened Merkle leaf contains a uniform salt, queried with probability at most $q_H/2^\lambda$. An honest q_j keeps at least $2k' - 1 - L - 2q \ge k' - 1$ uniform coefficients given the view, so its digest is queried with probability at most $q_H/|\mathbb{F}_p|$. Each programmed input contains a fresh root, so programming fails with probability at most $4q_H/2^{2\lambda}$. By completeness, the honest servers accept in both executions exactly when $b = 1$ and no corrupted server rejects.

Corrupted dealer: extraction. $\mathcal{S}$ extracts from the recorded queries every column of $\mathrm{rt}_{\hat{W}}$ and every pair of rt_C that has a path of preimages; extracted values are unique unless the oracle has a collision, and an opening without such a path passes its Merkle check with probability $O(q_H/2^{2\lambda})$. Set unextracted entries to 0. Let U be the extracted matrix restricted to the T columns in $D \setminus H$, with C and Z_C as two further rows. The challenge α is derived after $\mathrm{rt}_{\hat{W}}$, the challenges $\mathbf{v}$, γ_j, β_j after both roots, and Q after u and all digests; each honest S_h checks its digest, so q_h is fixed before Q except with probability $O(q_H^2/2^{2\lambda})$.

Corrupted dealer: soundness. Let $d = T - K^* + 1$ be the minimum distance of the Reed–Solomon code of degree $< K^*$ on $D \setminus H$. (a) If U is e-far from the interleaved code, then since $4e < d$, the proximity lemma of Ligero [5] shows that $\mathbf{v}^\top U$ is e-far from the code except with probability $d/|\mathbb{F}_p|$ over $\mathbf{v}$. The polynomial u of degree $< K^*$ then differs from $\mathbf{v}^\top U$ at more than e of the T points, and q distinct sampled points avoid all of them with probability at most $(1 - e/T)^q$. (b) Otherwise there is a set E of at most e columns outside which every row of U agrees with a polynomial $\tilde{f}_r$ of degree $< K^*$, unique since $2e < d$, and these polynomials fix an effective witness $(\tilde{f}_r(\eta^i))_{i \in W}$ for every segment. If some instance's statement is false for the honest servers' inputs, then either the effective witness violates a constraint while matching all honest shares, or it differs from the share of some honest holder S_h, which exists since $n - 2t \ge 1$. In the first case, $\tilde{\mathcal{N}}$ is divisible by Z_W with probability at most $5\ell/|\mathbb{F}_p|$ over α, so $Z_W \tilde{C} - \tilde{\mathcal{N}}$ is a nonzero polynomial of degree at most $2K^* + c - 2$. In the second case, let $P_h = \sum_\sigma \Lambda_{h,\sigma}(\gamma_{h,\sigma,M} \tilde{f}_{M_\sigma} + \gamma_{h,\sigma,a} \tilde{f}_{a_\sigma}) + \tilde{Z}_h$, of degree at most $k' + K^* - 2$. At a mismatching position η^i, the value $P_h(\eta^i)$ minus the value that S_h expects is $\beta_{h,i} D + \tilde{Z}_h(\eta^i)$, where D is a linear form in the γ_h with a nonzero coefficient; it vanishes with probability at most $2/|\mathbb{F}_p|$ over γ_h and $\beta_{h,i}$. So if S_h's check on W passes, $q_h - P_h$ is a nonzero polynomial of degree at most $2K^* + c - 2$. In both cases a sampled x passes only if it is a root of that polynomial or x or ηx lies in E, which happens with probability at most $(2K^* + c - 2 + 2e)/T$ per point and at most $((2K^* + c - 2 + 2e)/T)^q$ for q distinct points. A query yields a bad α, $\mathbf{v}$, γ_h or β_h with probability $O((n_c + \ell + n)/|\mathbb{F}_p|)$; summing over the q_H queries and the final output gives the first term of the bound.

Corrupted dealer: simulation. $\mathcal{S}$ replies abort if U is e-far from the code, which it detects by unique decoding. Otherwise it inputs the share pairs of the effective witness to the functionality. It then runs every check that does not use the honest servers' shares, and each honest verifier's check on W with the effective witness in place of its shares. It replies continue exactly when all of these pass and no corrupted server rejects. If every instance has $b = 1$, the effective witness carries the honest shares and these checks coincide with the honest servers' own; otherwise both executions reject except with the probability bounded above. Adding the parts gives the bound of Lemma 5.5. □

D.5 Properties of Our VIP Protocol

Proof of Lemma 6.1. Fix an instance indexed by $i \in [n]$, and let $\gamma \coloneqq \lceil \log L \rceil$. Throughout the proof, dummy zero pairs appended when the current length is odd are ignored in the notation, since they do not change any inner product.

Correctness. Assume first that the sharing server S_i is honest. Every value explicitly distributed by S_i is shared using VSS, and every subsequent shared value is obtained from already shared values by local linear operations and Lagrange interpolation. Hence, unless the protocol aborts, all sharings generated by the protocol are consistent. For each round $k \in [\gamma]$, let $D^{(k-1)}$ denote the inner product of the current folded vectors at the beginning of round k. In particular, $D^{(0)} = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$. Let $q^{(k)}$ be the quadratic polynomial honestly defined in round k from the current folded vectors: $q^{(k)}(x) = \sum_{\ell=1}^{L/2} f_{2\ell-1}(x) f_{2\ell}(x)$, where L is the current folded length in that round. By construction of the linear polynomials, the two evaluation points 1 and 2 correspond to the two halves of the current folded vectors. Therefore, $q^{(k)}(1) + q^{(k)}(2) = D^{(k-1)}$. In the first round, the protocol defines $[\![c_i]\!] \leftarrow [\![q^{(1)}(1)]\!] + [\![q^{(1)}(2)]\!]$. Hence, $c_i = q^{(1)}(1) + q^{(1)}(2) = D^{(0)} = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$. For each round k, after the challenge $r_k \xleftarrow{\$} \mathbb{F}_p \setminus \{1,2,3\}$ is sampled, the protocol sets $d_i^{(k)} = q^{(k)}(r_k)$ and updates the folded vectors to $a_i^{(\ell)} \leftarrow f_{2\ell-1}(r_k)$ and $b_i^{(\ell)} \leftarrow f_{2\ell}(r_k)$. Thus the next folded inner product is exactly $D^{(k)} = q^{(k)}(r_k) = d_i^{(k)}$. By induction over k, after the final round the folded vectors have length one and $d_i^{(\gamma)} = D^{(\gamma)} = a_i^{(1)} b_i^{(1)}$. It remains to show that $\sigma_i = 0$. For each $k \geq 2$, the protocol checks the residual $d_i^{(k-1)} - q^{(k)}(1) - q^{(k)}(2)$. In the honest execution, $d_i^{(k-1)} = D^{(k-1)}$ and $q^{(k)}(1) + q^{(k)}(2) = D^{(k-1)}$, so every such residual is zero. Hence $\sigma_i = \sum_{k=2}^{\gamma} \epsilon_k \left(d_i^{(k-1)} - q^{(k)}(1) - q^{(k)}(2) \right) = 0$. Therefore the output sharings reconstruct to $c_i = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$, $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$, and $\sigma_i = 0$. This proves correctness.

Soundness. Assume now that the sharing server S_i is corrupted, and condition on the event that the protocol does not abort. By VSS, each value shared by S_i is fixed consistently before the corresponding challenge r_k is sampled. Thus, for each round k, the three shared values determine a unique quadratic polynomial. We denote this actually shared polynomial by $\tilde{q}^{(k)}$. We also define an ideal honest folding process. Let $q_*^{(k)}$ denote the quadratic polynomial that would be obtained in round k from the honestly folded vectors determined by the original inputs and the previously sampled challenges $r_1, \ldots, r_{k-1}$. Let $D^{(0)} \coloneqq \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$ be the true initial inner product, and for $k \in [\gamma]$ define $D^{(k)} \coloneqq q_*^{(k)}(r_k)$. Thus $D^{(k)}$ is the honest folded inner product after round k. On the other hand, the actual values produced by the possibly corrupted dealer are $c_i = \tilde{q}^{(1)}(1) + \tilde{q}^{(1)}(2)$, and, for $k \in [\gamma]$, $d^{(k)} \coloneqq \tilde{q}^{(k)}(r_k)$. For notational convenience, define $d^{(0)} \coloneqq c_i$. For each $k \in [\gamma]$, define the polynomial error $e_k(x) \coloneqq \tilde{q}^{(k)}(x) - q_*^{(k)}(x)$, and define the folded error $\delta^{(k)} \coloneqq d^{(k)} - D^{(k)}$ for $k \in [\gamma]$. For the initial claim, define $\delta^{(0)} \coloneqq d^{(0)} - D^{(0)} = c_i - \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$. Since $c_i = \tilde{q}^{(1)}(1) + \tilde{q}^{(1)}(2)$ and $D^{(0)} = q_*^{(1)}(1) + q_*^{(1)}(2)$, we have $\delta^{(0)} = e_1(1) + e_1(2)$. Moreover, for every $k \in [\gamma]$, $\delta^{(k)} = d^{(k)} - D^{(k)} = \tilde{q}^{(k)}(r_k) - q_*^{(k)}(r_k) = e_k(r_k)$. For each $k \in \{2, \ldots, \gamma\}$, define the actual residual $\tilde{s}^{(k)} \coloneqq d^{(k-1)} - \tilde{q}^{(k)}(1) - \tilde{q}^{(k)}(2)$. Since the honest polynomial satisfies $q_*^{(k)}(1) + q_*^{(k)}(2) = D^{(k-1)}$, we obtain $\tilde{s}^{(k)} = \delta^{(k-1)} - (e_k(1) + e_k(2))$. The value computed by the protocol is $\sigma_i = \sum_{k=2}^{\gamma} \epsilon_k \tilde{s}^{(k)}$, where the coefficients $\epsilon_2, \ldots, \epsilon_\gamma$ are sampled only after all folding rounds are completed. We now prove that, except with the claimed probability, it is impossible to have an incorrect initial claim c_i while also satisfying the two later checks $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$ and $\sigma_i = 0$. Let Bad be the event that $\delta^{(0)} \neq 0$, $\sigma_i = 0$, and $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$. Since $\delta^{(0)} = c_i - \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$, ruling out Bad is exactly the desired soundness statement. We distinguish two cases.

Case 1: Some residual from rounds $2, \ldots, \gamma$ is nonzero. Suppose there exists $k \in \{2, \ldots, \gamma\}$ such that $\tilde{s}^{(k)} \neq 0$. Let $j = \max\{k \in \{2, \ldots, \gamma\} : \tilde{s}^{(k)} \neq 0\}$. All values $\tilde{s}^{(2)}, \ldots, \tilde{s}^{(\gamma)}$ are fixed before $\epsilon_2, \ldots, \epsilon_\gamma$ are sampled. Conditioned on all coefficients except ϵ_j, the value $\sigma_i = \sum_{k=2}^{\gamma} \epsilon_k \tilde{s}^{(k)}$ is a nonconstant affine function of the uniform variable ϵ_j. Therefore, $\Pr[\sigma_i = 0 \mid \exists k \geq 2 : \tilde{s}^{(k)} \neq 0] \leq 1/|\mathbb{F}_p|$. Consequently, $\Pr[\text{Bad} \wedge (\exists k \in \{2, \ldots, \gamma\} : \tilde{s}^{(k)} \neq 0)] \leq 1/|\mathbb{F}_p|$.

Case 2: All residuals from rounds $2, \ldots, \gamma$ are zero. Assume now that $\tilde{s}^{(k)} = 0$ for all $k \in \{2, \ldots, \gamma\}$. We show that, on Bad, some nonzero error polynomial must vanish at one of the random folding challenges. First, not all e_k can be the zero polynomial. Indeed, if $e_k \equiv 0$ for all $k \in [\gamma]$, then in particular $e_1 \equiv 0$, and hence $\delta^{(0)} = e_1(1) + e_1(2) = 0$, contradicting Bad. Thus, on Bad, there exists at least one round k with $e_k \not\equiv 0$. Let $\tau = \max\{k \in [\gamma] : e_k \not\equiv 0\}$. Then e_τ is a nonzero polynomial of degree at most 2.

We claim that Bad and the condition $\tilde{s}^{(k)} = 0$ for all $k \geq 2$ imply $\delta^{(\tau)} = 0$. If $\tau = \gamma$, the condition $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$ implies $d^{(\gamma)} = D^{(\gamma)}$, because $D^{(\gamma)}$ is exactly the honest final folded inner product, namely $D^{(\gamma)} = a_i^{(1)} b_i^{(1)}$. Therefore $\delta^{(\gamma)} = 0$. If instead $\tau < \gamma$, then by maximality of τ we have $e_{\tau+1} \equiv 0$. Since all residuals from round 2 onward are zero, in particular $\tilde{s}^{(\tau+1)} = 0$. Using $\tilde{s}^{(\tau+1)} = \delta^{(\tau)} - (e_{\tau+1}(1) + e_{\tau+1}(2))$, we get $0 = \delta^{(\tau)}$. Hence in all cases, $\delta^{(\tau)} = 0$. But $\delta^{(\tau)} = e_\tau(r_\tau)$, so on this event one must have $e_\tau(r_\tau) = 0$ for a nonzero polynomial e_τ of degree at most 2.

For each $j \in [\gamma]$, let E_j be the event that Bad holds, all residuals $\tilde{s}^{(k)}$ for $k \in \{2, \ldots, \gamma\}$ are zero, and $j = \max\{k \in [\gamma] : e_k \not\equiv 0\}$. The events $E_1, \ldots, E_\gamma$ are pairwise disjoint. Fix $j \in [\gamma]$. Condition on any transcript up to, but not including, the sampling of r_j. At that point, the dealer has already VSS-shared $\tilde{q}^{(j)}(1), \tilde{q}^{(j)}(2), \tilde{q}^{(j)}(3)$, and therefore e_j is already fixed. On E_j, this polynomial is nonzero and has degree at most 2, and the event E_j implies $e_j(r_j) = 0$. Since $r_j \xleftarrow{\$} \mathbb{F}_p \setminus \{1,2,3\}$, the Schwartz–Zippel lemma gives $\Pr[E_j] \leq 2/(|\mathbb{F}_p| - 3)$. Summing over all $j \in [\gamma]$ yields $\Pr[\text{Bad} \wedge (\forall k \in \{2, \ldots, \gamma\},\ \tilde{s}^{(k)} = 0)] \leq 2\gamma/(|\mathbb{F}_p| - 3)$. Combining the two cases, $\Pr[\text{Bad}] \leq 1/|\mathbb{F}_p| + 2\gamma/(|\mathbb{F}_p| - 3) \leq (2\gamma + 1)/(|\mathbb{F}_p| - 3)$. Since $\gamma = \lceil \log L \rceil$, this is exactly $\varepsilon_{\mathsf{VIP}} = (2\lceil \log L \rceil + 1)/(|\mathbb{F}_p| - 3)$.

Therefore, except with probability at most $\varepsilon_{\mathsf{VIP}}$, either the protocol aborts, or the output sharings are such that if the reconstructed values satisfy $d_i^{(\gamma)} = a_i^{(1)} b_i^{(1)}$ and $\sigma_i = 0$, then necessarily $c_i = \sum_{\ell=1}^{L} a_i^{(\ell)} b_i^{(\ell)}$. This proves soundness, and hence the lemma. □

D.6 Properties of Our Batched VIP Protocol

Proof of Lemma 6.2. We prove correctness, designated-verifier privacy, and soundness separately. For readability, write d_i for $d_i^{(\gamma)}$, a_i for $a_i^{(1)}$, and b_i for $b_i^{(1)}$ in the proof.

Correctness. Assume that all underlying Π_{VIP} instances are executed honestly. By Lemma 6.1, for every $i \in [n]$, the i-th instance outputs consistent secret sharings of values satisfying $c_i = C_i^*$, $d_i = a_i b_i$, and $\sigma_i = 0$. Protocol 6 computes $[\![c]\!] \leftarrow \sum_{i=1}^n [\![c_i]\!]$ by local linear operations. Hence the reconstructed value satisfies $c = \sum_{i=1}^n c_i = \sum_{i=1}^n C_i^* = C^*$. Moreover, Protocol 6 computes $[\![\Sigma]\!] \leftarrow \sum_{i=1}^n \epsilon_i' [\![\sigma_i]\!]$. Since $\sigma_i = 0$ for every i, we have $\Sigma = 0$. It remains to prove that $W(\rho) = U(\rho)V(\rho)$. By construction, $U(x)$ and $V(x)$ are polynomials of degree at most n such that $U(i) = a_i$ and $V(i) = b_i$ for every $i \in [n]$. For each $i \in [n]$, the polynomial $W(x)$ is interpolated so that $W(i) = d_i$. Hence, by correctness of each VIP instance, $W(i) = d_i = a_i b_i = U(i)V(i)$ for every $i \in [n]$. For each $i \in \{0, n+1, \ldots, 2n\}$, Protocol 6 obtains $[\![W(i)]\!] = [\![U(i)V(i)]\!]$ from $\mathcal{F}_{\mathsf{RSS.Mul}}$, and therefore $W(i) = U(i)V(i)$ at these points as well. Thus $W(x)$ and $U(x)V(x)$, both of degree at most $2n$, agree on the $2n+1$ distinct points $0, 1, \ldots, 2n$. Therefore they are identical polynomials, and in particular $W(\rho) = U(\rho)V(\rho)$. Hence the output tuple is accepting. All output sharings are consistent because they are obtained from consistent outputs of the VIP instances, $\mathcal{F}_{\mathsf{Rand}}$, and $\mathcal{F}_{\mathsf{RSS.Mul}}$ by local operations and interpolation. This proves correctness.

Designated-verifier privacy. Recall that $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$ denotes the joint view of the corrupted sharing servers in the underlying Π_{VIP} instances, including their internal views and local outputs. We show that the opened tuple $(U(\rho), V(\rho), W(\rho), \Sigma, c)$ can be simulated from c, the public coins, and $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$. Let $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ with $|\mathcal{I}_{\mathsf{corr}}| \le t$ denote the set of corrupted sharing servers. For each $i \in [n]$, define the local multiplication residual $\Delta_i := d_i - a_i b_i$. For every honest sharing server S_i, Lemma 6.1 gives $d_i = a_i b_i$ and $\sigma_i = 0$, and hence $\Delta_i = 0$ and $\sigma_i = 0$. For every corrupted sharing server S_i, the values Δ_i and σ_i are determined by $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$ and the public coins. Let $E(x) := W(x) - U(x)V(x)$. Then $E(x)$ has degree at most $2n$. By the construction of $W(x)$, for every $i \in [n]$ we have $E(i) = d_i - a_i b_i = \Delta_i$. For every $i \in \{0, n+1, \ldots, 2n\}$, $W(i)$ is obtained from $\mathcal{F}_{\mathsf{RSS.Mul}}$ as $W(i) = U(i)V(i)$, so $E(i) = 0$. Therefore $E(x)$ is completely determined by the values $\{\Delta_i\}_{i\in[n]}$ and the public interpolation points. Since $\Delta_i = 0$ for every honest i, the value $E(\rho)$ is determined by the public coin ρ and the corrupted instances' residuals $\{\Delta_i\}_{i\in\mathcal{I}_{\mathsf{corr}}}$. Similarly, since $\Sigma = \sum_{i=1}^n \epsilon_i' \sigma_i$ and $\sigma_i = 0$ for every honest i, the value Σ is determined by the public coins $\{\epsilon_i'\}_{i\in[n]}$ and the corrupted instances' values $\{\sigma_i\}_{i\in\mathcal{I}_{\mathsf{corr}}}$. It remains to argue that $U(\rho)$ and $V(\rho)$ are fresh uniform masks, even conditioned on c, the public coins, and $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$. Let $\lambda_0(\rho)$ be the Lagrange coefficient of the interpolation point 0 in the degree-n interpolation of $U(\rho)$ from the points $0, 1, \ldots, n$. Since $\rho \notin \{0, 1, \ldots, 2n\}$, in particular $\rho \notin \{1, \ldots, n\}$, and hence $\lambda_0(\rho) \neq 0$. Thus $U(\rho)$ can be written as $U(\rho) = \lambda_0(\rho) a_0 + \Delta_U$, where Δ_U is determined by ρ and the values $a_1, \ldots, a_n$. The value a_0 is sampled by $\mathcal{F}_{\mathsf{Rand}}$ independently of all VIP outputs and of c. Moreover, by privacy of the RSS sharing produced by $\mathcal{F}_{\mathsf{Rand}}$, a_0 remains uniformly distributed over $\mathbb{F}_p$ conditioned on the corrupted parties' view. Since $\lambda_0(\rho) \neq 0$, it follows that $U(\rho)$ is uniformly distributed over $\mathbb{F}_p$ conditioned on c, the public coins, and $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$. The same argument applies to $V(\rho)$, using the independently sampled random value b_0. Therefore $U(\rho)$ and $V(\rho)$ are independent uniform field elements conditioned on c, the public coins, and $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$.

The simulator works as follows. Given c, the public coins, and $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$, it computes $\Delta_\Sigma := \sum_{i\in\mathcal{I}_{\mathsf{corr}}} \epsilon_i' \sigma_i$ and computes $\Delta_W := E(\rho)$ from the residuals $\{\Delta_i = d_i - a_i b_i\}_{i\in\mathcal{I}_{\mathsf{corr}}}$ and the public interpolation coefficients. It then samples $u, v \xleftarrow{\$} \mathbb{F}_p$ independently and outputs $(u, v, uv + \Delta_W, \Delta_\Sigma, c)$. By the discussion above, in the real execution we have $U(\rho)$ and $V(\rho)$ distributed as independent uniform values u, v, while $W(\rho) = U(\rho)V(\rho) + E(\rho)$ and $\Sigma = \Delta_\Sigma$. Hence the simulator's output distribution is identical to the real distribution of $(U(\rho), V(\rho), W(\rho), \Sigma, c)$. This proves the designated-verifier privacy statement.

Finally, suppose the output tuple is accepting. Then $\Sigma = 0$ and $W(\rho) = U(\rho)V(\rho)$, i.e., $\Delta_\Sigma = 0$ and $\Delta_W = 0$. In this case the simulator does not need $\mathsf{View}_{\mathsf{corr}}^{\mathsf{VIP}}$: given only c and the fact that the tuple is accepting, it samples $u, v \xleftarrow{\$} \mathbb{F}_p$ and outputs $(u, v, uv, 0, c)$. This is identically distributed to the real opened tuple conditioned on acceptance. Thus, in accepting executions, the opened tuple is simulatable from c and the fact that the tuple is accepting.

Soundness. Let Bad be the event that the protocol does not abort, the output tuple is accepting, and yet $c \neq C^*$. We bound the probability of Bad. For each $i \in [n]$, let Good_i denote the following condition: either the i-th VIP instance aborts, or it outputs consistent sharings such that if $d_i = a_i b_i$ and $\sigma_i = 0$, then $c_i = C_i^*$. If S_i is honest, then Good_i holds by Lemma 6.1 correctness. If S_i is corrupted, then by Lemma 6.1 soundness, Good_i fails with probability at most $\varepsilon_{\mathsf{VIP}}$. By a union bound over all n instances, the event $\mathsf{Good} := \bigwedge_{i=1}^n \mathsf{Good}_i$ fails with probability at most $n\varepsilon_{\mathsf{VIP}}$. We now condition on Good and on the event that the protocol does not abort. All output sharings are then consistent: the VIP outputs are consistent by Good, $\mathcal{F}_{\mathsf{RSS.Mul}}$ outputs consistent sharings, and all remaining values are obtained by local linear operations and interpolation. Assume toward Bad that the output tuple is accepting but $c \neq C^*$. Since Protocol 6 computes $c = \sum_{i=1}^n c_i$ and $C^* = \sum_{i=1}^n C_i^*$, the inequality $c \neq C^*$ implies that there exists at least one index $j \in [n]$ such that $c_j \neq C_j^*$. By Good_j, this implies that the local acceptance condition of the j-th VIP instance fails, namely $d_j \neq a_j b_j$ or $\sigma_j \neq 0$. We distinguish two cases. First, suppose that $\sigma_i \neq 0$ for at least one $i \in [n]$. The values $\sigma_1, \ldots, \sigma_n$ are fixed before the coefficients $\epsilon_1', \ldots, \epsilon_n'$ are sampled by $\mathcal{F}_{\mathsf{Coin}}$. Since the vector $(\sigma_1, \ldots, \sigma_n)$ is nonzero, the value $\Sigma = \sum_{i=1}^n \epsilon_i' \sigma_i$ is a nonconstant affine function of at least one uniform coefficient ϵ_i'. Therefore $\Pr[\Sigma = 0] \le 1/|\mathbb{F}_p|$. Since an accepting output tuple requires $\Sigma = 0$, the probability of Bad in this case is at most $1/|\mathbb{F}_p|$. Second, suppose that $\sigma_i = 0$ for every $i \in [n]$. Since $c \neq C^*$ and Good holds, there must exist an index j such that $d_j \neq a_j b_j$. Define $E(x) := W(x) - U(x)V(x)$. As above, $E(x)$ has degree at most $2n$, satisfies $E(i) = d_i - a_i b_i$ for every $i \in [n]$, and satisfies $E(i) = 0$ for every $i \in \{0, n+1, \ldots, 2n\}$. Because $E(j) = d_j - a_j b_j \neq 0$, the polynomial $E(x)$ is nonzero. The polynomial $E(x)$ is fixed before ρ is sampled: the values defining U, V, W are fixed before the final call to $\mathcal{F}_{\mathsf{Coin}}$ that samples ρ. The challenge ρ is sampled uniformly from $\mathbb{F}_p \setminus \{0, 1, \ldots, 2n\}$. Since $E(x)$ is a nonzero polynomial of degree at most $2n$, the Schwartz–Zippel lemma gives $\Pr[E(\rho) = 0] \le 2n/(|\mathbb{F}_p| - 2n - 1)$. But $E(\rho) = 0$ is exactly the condition $W(\rho) = U(\rho)V(\rho)$. Thus, in this second case, the probability that the output tuple is accepting is at most $2n/(|\mathbb{F}_p| - 2n - 1)$.

Combining the failure probability of Good with the two cases above, we obtain $\Pr[\mathsf{Bad}] \le n\varepsilon_{\mathsf{VIP}} + 2n/(|\mathbb{F}_p| - 2n - 1) + 1/|\mathbb{F}_p|$. Therefore, except with probability at most $\varepsilon^{\mathsf{Prl}}_{\mathsf{VIP}} = n\varepsilon_{\mathsf{VIP}} + 2n/(|\mathbb{F}_p| - 2n - 1) + 1/|\mathbb{F}_p|$, either the protocol aborts, or the output sharings are consistent and satisfy the implication that if the output tuple is accepting, then $c = C^*$. This proves soundness and completes the proof. □

D.7 Proof of Theorem 6.4

Let $I \subseteq [n]$ be the set of corrupted servers, with $|I| \le t$. We write $\mathcal{H} = [n] \setminus I$ for the honest servers (reserving H for the hash function). For each client input $x^{(j)}$, define $a^{(j)} := x^{(j)} + k$ and $b^{(j)} := (\alpha^{(j)})^e$. Thus the desired opened value before the final exponentiation is $v^{(j)} = a^{(j)}b^{(j)} = (x^{(j)} + k)(\alpha^{(j)})^e$.

We first prove correctness. Assume that all parties are honest and that no ideal functionality aborts. By construction of the balanced assignment $\{\Lambda_i\}_{i\in[n]}$, for every $j \in [m]$, the product $a^{(j)}b^{(j)}$ is decomposed as $a^{(j)}b^{(j)} = \sum_{i=1}^{n}\sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$. Protocol 8 samples additive zero shares $[z^{(j)}]_1, \dots, [z^{(j)}]_n$ satisfying $\sum_{i=1}^{n}[z^{(j)}]_i = 0$ and each server computes $v_i^{(j)} = \sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)} + [z^{(j)}]_i$. Therefore, $\sum_{i=1}^{n} v_i^{(j)} = a^{(j)}b^{(j)}$. The batching coefficients $\epsilon_1, \dots, \epsilon_m$ are derived from the hashes of the values $v_i^{(j)}$. Protocol 8 then invokes $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ on the flattened vectors $\{\hat{a}_i^{(j,\ell)}, \hat{b}_i^{(j,\ell)}\}_{j\in[m],\ell\in[L]}$, where $\hat{a}_i^{(j,\ell)} = \epsilon_j a_i^{(j,\ell)}$ and $\hat{b}_i^{(j,\ell)} = b_i^{(j,\ell)}$. Thus, for each $i \in [n]$, the target inner product in the i-th underlying VIP instance is $C_i^* = \sum_{j=1}^{m}\epsilon_j \sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$. The effective input length of each VIP instance is mL. By Lemma 6.2 correctness, the output tuple of $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ is accepting and its value c satisfies $c = \sum_{i=1}^{n} C_i^* = \sum_{j=1}^{m}\epsilon_j \sum_{i=1}^{n}\sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$. Since the zero masks sum to zero for every j, we also have $\sum_{i=1}^{n} v_i^{(j)} = \sum_{i=1}^{n}\sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$. Hence $c = \sum_{j=1}^{m}\epsilon_j \sum_{i=1}^{n} v_i^{(j)}$. Therefore the client accepts all checks. Finally, since $p = eg+1$, we have $eg = p-1$. For each j, $((\alpha^{(j)})^e)^g = (\alpha^{(j)})^{eg} = 1$ because $\alpha^{(j)} \in \mathbb{F}_p^*$. Thus the client outputs $(v^{(j)})^g = ((x^{(j)} + k)(\alpha^{(j)})^e)^g = (x^{(j)} + k)^g = F_k(x^{(j)})$. This proves correctness.

We construct a simulator $\mathcal{S}$ for any static adversary $\mathcal{A}$ corrupting at most t servers and, possibly, the client C. The simulator internally emulates the hybrid functionalities toward $\mathcal{A}$ and interacts with $\mathcal{F}^{F_k}_{\mathsf{dVOPRF}}$.

Extracting the client's input. Whether or not C is corrupted, $\mathcal{S}$ must hand $\mathcal{F}^{F_k}_{\mathsf{dVOPRF}}$ a well-defined input vector. Playing the honest servers, $\mathcal{S}$ knows every mask $r^{(j)}$: the honest servers hold $[\![r^{(j)}]\!]_i$ for $n - t \ge t + 1$ indices i, and these cover all N additive components. At Step 2 of Protocol 8 it observes the values $u^{(j)}$ that C sends to the honest servers. If the deferred echo checked at Step 6 fails—that is, if C sent different $u^{(j)}$ to two honest servers—or if C aborts or sends nothing, then $\mathcal{S}$ sends $\perp$ to $\mathcal{F}^{F_k}_{\mathsf{dVOPRF}}$ and halts, matching the $\perp$ branch of Fig. 3. Otherwise all honest servers hold a common $u^{(j)}$ and $\mathcal{S}$ submits $x^{(j)} := u^{(j)} - r^{(j)}$. By Lemma 6.3 (consistency and extractability), this is exactly the value that the honest servers' sharings $[\![x^{(j)}]\!]$ encode, so the ideal outputs $y^{(j)} = F_k(x^{(j)})$ agree with the real execution. When C is honest, Lemma 6.3 (input integrity) guarantees in addition that $x^{(j)}$ is the value C actually supplied: a corrupted server that lies during the mask opening is caught by the hash check, and C aborts.

We first treat the case in which $\mathcal{A}$ corrupts C, and argue that the corrupted client's view is simulatable from the extracted input and the ideal output. Observe that nothing C sends influences what it receives: within $\Pi^{m}_{\mathsf{dVOPRF}}$ the client is a pure *receiver*. It cannot bias the batching coefficients ϵ_j, which the servers derive by hashing their own h_i, nor the VIP challenge ρ, which $\mathcal{F}_{\mathsf{Coin}}$ produces among the servers; and the additive zero shares from $\mathcal{F}_{\mathsf{Zero}}$ that mask the $v_i^{(j)}$ are independent of C's messages. The client's only influence is its choice of $u^{(j)}$, which the extraction step above has already translated into an ideal input. The argument below therefore never uses semi-honesty of C. For each $j \in [m]$, the simulator receives the ideal output $y^{(j)} = F_k(x^{(j)})$ from $\mathcal{F}^{F_k}_{\mathsf{dVOPRF}}$. It samples $v^{(j)}$ uniformly from the fiber $\{v \in \mathbb{F}_p^* : v^g = y^{(j)}\}$. This matches the real distribution: in the real protocol, $v^{(j)} = (x^{(j)} + k)(\alpha^{(j)})^e$, and since $\alpha^{(j)}$ is uniform over $\mathbb{F}_p^*$, $(\alpha^{(j)})^e$ is uniform over the kernel of the map $z \mapsto z^g$. Hence $v^{(j)}$ is uniform over the same fiber of $y^{(j)}$. The simulator then samples the values $\{v_i^{(j)}\}_{i\in[n],j\in[m]}$ subject to the aggregate constraints $\sum_{i=1}^{n} v_i^{(j)} = v^{(j)}$ for all j. If some servers are corrupted, their values are fixed consistently with the adversary's view, and the simulator samples the honest servers' values uniformly subject to the same aggregate constraints. This is identical to the real distribution because the additive zero shares mask the individual local products and reveal only their sum. The simulator computes the hashes h_i, the random-oracle-derived coefficients ϵ_j, and sets $c := \sum_{j=1}^{m}\epsilon_j v^{(j)}$. It then simulates the proof-opening tuple $(U(\rho), V(\rho), W(\rho), \Sigma, c)$ using Lemma 6.2 designated-verifier privacy. Namely, the opened tuple is statistically simulatable from c, the public coins, and the joint view of the corrupted VIP instances. In accepting executions, it is simulatable from c and the fact that the tuple is accepting alone. Therefore, the client's entire view consisting of the opened values $v_i^{(j)}$ and the proof tuple $(U(\rho), V(\rho), W(\rho), \Sigma, c)$ is statistically identical to the real execution.

If the client is honest and only servers are corrupted, the simulator uses the honest client's inputs in the ideal functionality and simulates the corrupted servers' shares and messages using the privacy of the underlying RSS sharings and the ideal hybrid functionalities. In particular, at most t corrupted servers see only forbidden sets of RSS shares, and the values generated by $\mathcal{F}_{\mathsf{RandExp}}$, $\mathcal{F}_{\mathsf{Zero}}$, $\mathcal{F}_{\mathsf{Coin}}$, and $\mathcal{F}_{\mathsf{Rand}}$ are simulated exactly according to their ideal distributions. The proof values produced by $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ are simulated using Lemma 6.2 privacy. Thus the corrupted servers' view is statistically identical to the real view, except for the bad events analyzed below, which account for the statistical simulation error.

It remains to bound the bad event in which a malicious adversary makes an honest client accept an incorrect output. This event is the only remaining way in which the real execution can differ from the ideal simulation. Let $L_{\mathsf{VIP}} := mL$ be the effective input length of each underlying VIP instance in Protocol 8. By Lemma 6.2 soundness applied with input length L_{VIP}, except with probability at most $\varepsilon^{\mathsf{Prl}}_{\mathsf{VIP}}(mL) = n\cdot(2\lceil\log(mL)\rceil+1)/(|\mathbb{F}_p|-3)+2n/(|\mathbb{F}_p|-2n-1)+1/|\mathbb{F}_p|$, either $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ aborts, or whenever its output tuple is accepting, the reconstructed value c satisfies $c = \sum_{i=1}^{n}\sum_{j=1}^{m}\epsilon_j \sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$.

Equivalently, letting $y^{(j)} \coloneqq \sum_{i=1}^{n}\sum_{\ell=1}^{L} a_i^{(j,\ell)} b_i^{(j,\ell)}$, we have $c = \sum_{j=1}^{m} \epsilon_j y^{(j)}$. Now consider the values $\tilde{v}^{(j)}$ reconstructed by the client from the server-provided values $v_i^{(j)}$, namely $\tilde{v}^{(j)} = \sum_{i=1}^{n} v_i^{(j)}$. The client accepts only if $c = \sum_{j=1}^{m} \epsilon_j \tilde{v}^{(j)}$. Combining this with the conclusion of Lemma 6.2, an accepting execution with some incorrect opened value $\tilde{v}^{(j)} \neq y^{(j)}$ must satisfy $\sum_{j=1}^{m} \epsilon_j(\tilde{v}^{(j)} - y^{(j)}) = 0$. If the error vector $(\tilde{v}^{(1)} - y^{(1)}, \ldots, \tilde{v}^{(m)} - y^{(m)})$ is nonzero, then this is a nontrivial linear equation in the random-oracle-derived batching coefficients $\epsilon_1, \ldots, \epsilon_m$. Because these coefficients are derived after the values are committed by the hashes h_i, the probability that this equation holds is at most $1/|\mathbb{F}_p|$. Here $a^{(j)} = x^{(j)} + k$ holds by Lemma 6.3: on non-abort the honest servers' shares are consistent components of a sharing of a single well-defined $x^{(j)}$, and when C is honest that value is the one it supplied. This is what licenses the substitution below, and is precisely the step a non-verifiable input sharing would leave unjustified. Therefore, except with probability at most $\varepsilon_{\mathsf{VIP}}^{\mathsf{Prl}}(mL) + 1/|\mathbb{F}_p|$, if the honest client accepts, then $\tilde{v}^{(j)} = y^{(j)} = a^{(j)}b^{(j)} = (x^{(j)} + k)(\alpha^{(j)})^e$ for every $j \in [m]$. Consequently, the client's final outputs satisfy $(\tilde{v}^{(j)})^g = ((x^{(j)} + k)(\alpha^{(j)})^e)^g = (x^{(j)} + k)^g = F_k(x^{(j)})$ for every $j \in [m]$. Combining these error sources with the event that some $\alpha^{(j)} = 0$, which occurs with probability at most $m/|\mathbb{F}_p|$ (Theorem 6.4), gives $\varepsilon_{\mathsf{dVOPRF}} = \varepsilon_{\mathsf{VIP}}^{\mathsf{Prl}}(mL) + 1/|\mathbb{F}_p| + m/|\mathbb{F}_p|$, namely

$$\varepsilon_{\mathsf{dVOPRF}} = \frac{n(2\lceil\log(mL)\rceil + 1)}{|\mathbb{F}_p| - 3} + \frac{2}{|\mathbb{F}_p|} + \frac{2n}{|\mathbb{F}_p| - 2n - 1} + \frac{m}{|\mathbb{F}_p|}.$$

Finally, we record what a corrupted client can learn about the key. Fig. 3 evaluates F_k on m points of $\mathcal{A}$'s choosing and returns nothing else, so a malicious client's view across all sessions is exactly that of an oracle for F_k. Key privacy against such a client therefore reduces to the pseudorandomness of Gold PRF, i.e. to the DSPRS assumption (§4.2). No further leakage arises from the protocol itself: by Lemma 6.2 (designated-verifier privacy), everything the client receives is statistically simulatable from the ideal outputs alone.

This completes the simulation proof. □

E Fiat–Shamir Variants of Our Protocols

We can apply the Fiat–Shamir transform [19] to reduce the number of rounds in our VIP protocol Π_{VIP} (Protocol 5) as in [11], though our VIP protocol is not a (public-coin) zero-knowledge proof. Suppose that $\{[\![s_1^{(k)}]\!], [\![s_2^{(k)}]\!], [\![s_3^{(k)}]\!]\}_{k\in[\lceil\log L\rceil]}$ are generated during the offline phase, and S_i knows all $s_1^{(k)}, s_2^{(k)}, s_3^{(k)}$. This can be done via random sharing and secure opening protocols. Let H be a random oracle. We give the formal description of the FS-applied VIP protocol $\widehat{\Pi}_{\mathsf{VIP}}$ in Protocol 20. The Fiat–Shamir transform allows S_i to share $(q^{(k)}(1), q^{(k)}(2), q^{(k)}(3))_{k\in[\gamma]}$ once, and therefore, the resulting protocol achieves constant rounds. Note that the Fiat–Shamir transform is not applicable to the batched VIP protocol (Protocol 6) and our dVOPRF protocol (Protocol 8).

F Detailed Efficiency Analysis for Online Phases

We analyze the communication costs and the number of rounds of our dVOPRF protocol (Protocol 8) and compare it with the naive dVOPRF construction built from the ideal functionalities for secure multiplication $\mathcal{F}_{\mathsf{RSS.Mul}}$ and correctness verification of multiplications $\mathcal{F}_{\mathsf{vrfy}}$ [11]. We assume that in both protocols, ideal functionalities independent of the client's input, i.e., $\mathcal{F}_{\mathsf{RandExp}}$, $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{Rand}}$, and $\mathcal{F}_{\mathsf{Zero}}$, are called in the offline phases if the protocols allow. Note that some of them must remain fixed at those points to ensure security; for example, $\mathcal{F}_{\mathsf{Coin}}$ at the last step in Π_{VIP} (Protocol 5) must be called.

Let $N = \binom{n}{t}$, $N' = \binom{n-1}{t}$, $N'' = \binom{n-1}{t-1}$, and $L = N^2/n$. We abstract the communication costs per server and the number of rounds for several protocols as follows:

- Let $\mathsf{c}_{\mathsf{Share}}$ and $\mathsf{r}_{\mathsf{Share}}$ denote the communication cost per server and the number of rounds for (t, n)-RSS sharing *by a server dealer* (as inside Π_{VIP} and $\mathcal{F}_{\mathsf{vrfy}}$); the client's input is handled separately by $\mathsf{c}_{\mathsf{Input}}$ below. In this analysis, we consider two instantiations of $\mathcal{F}_{\mathsf{RSS.Share}}$. The first one is the traditional RSS sharing protocol; a dealer generates RSS shares of the input x and distributes them to n parties. The communication cost is $\mathsf{c}_{\mathsf{Share}} = N'$ field elements per

---

Players: Servers $S_1, \ldots, S_n$, adversary $\mathcal{A}$ corrupting up to t servers, and $\mathcal{F}_{\mathsf{vrfy}}$.

Input: Honest servers $\{S_i\}_{i\in\mathcal{H}}$ input their shares $\{([\![a^{(j)}]\!]_{\mathcal{H}}, [\![b^{(j)}]\!]_{\mathcal{H}}, [\![c^{(j)}]\!]_{\mathcal{H}})\}_{j=1}^{m}$.

Computation: $\mathcal{F}_{\mathsf{vrfy}}$ computes $\{(a^{(j)}, b^{(j)}, c^{(j)})\}$ and corrupted servers' shares $\{([\![a^{(j)}]\!]_{[n]\setminus\mathcal{H}}, [\![b^{(j)}]\!]_{[n]\setminus\mathcal{H}}, [\![c^{(j)}]\!]_{[n]\setminus\mathcal{H}})\}_{j=1}^{m}$ which are sent to $\mathcal{A}$. Then, $\mathcal{F}_{\mathsf{vrfy}}$ checks $c^{(j)} = a^{(j)} \cdot b^{(j)}$ holds for all $j \in [m]$.

Output: If the above holds, $\mathcal{F}_{\mathsf{vrfy}}$ sends accept to $\mathcal{A}$. In this case, $\mathcal{F}_{\mathsf{vrfy}}$ waits for $\mathcal{A}$ to send $\mathsf{out}_i \in \{\text{accept}, \text{reject}\}$ which is then handed to honest server S_i. Otherwise, $\mathcal{F}_{\mathsf{vrfy}}$ sends reject to $\mathcal{A}$ and the honest servers. In addition, $\mathcal{F}_{\mathsf{vrfy}}$ sends $d^{(j)} = c^{(j)} - a^{(j)} \cdot b^{(j)}$ for each $j \in [m]$ for which $d^{(j)} \neq 0$ to $\mathcal{A}$.

Figure 9: $\mathcal{F}_{\mathsf{vrfy}}$ – Verifying Correctness of Multiplications

---

Protocol 19 $\Pi^{m}_{\mathsf{naive\text{-}dVOPRF}}$ – Naive dVOPRF Protocol in the $(\mathcal{F}_{\mathsf{RandExp}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{RSS.Mul}}, \mathcal{F}_{\mathsf{vrfy}}, \mathcal{F}_{\mathsf{RSS.Open}})$-Hybrid Model

Players: Client C and Servers $S_1, \ldots, S_n$.
Input: Client C inputs $(x^{(1)}, \ldots, x^{(m)})$, and servers inputs $[\![k]\!]$.

1. Servers call $\mathcal{F}_{\mathsf{RandExp}}(e)$ and $\mathcal{F}_{\mathsf{Rand}}$ m times each to obtain $([\![(\alpha^{(j)})^e]\!])_{j\in[m]}$ and the input masks $([\![r^{(j)}]\!])_{j\in[m]}$.
2. Client C and the servers run Π_{Input} (Protocol 7) to obtain $([\![x^{(j)}]\!])_{j\in[m]}$.
3. For every $j \in [m]$, let $[\![a^{(j)}]\!] \leftarrow [\![x^{(j)} + k]\!]$ $(= [\![x^{(j)}]\!] + [\![k]\!])$ and $[\![b^{(j)}]\!] \leftarrow [\![(\alpha^{(j)})^e]\!]$.
4. For every $j \in [m]$, servers call $\mathcal{F}_{\mathsf{RSS.Mul}}$ with $([\![a^{(j)}]\!], [\![b^{(j)}]\!])$ to obtain $[\![c^{(j)}]\!]$.
5. Servers call $\mathcal{F}_{\mathsf{vrfy}}$ with $\{([\![a^{(j)}]\!], [\![b^{(j)}]\!], [\![c^{(j)}]\!])\}_{j=1}^{m}$.
6. If its output is reject, servers abort the protocol.
7. For every $j \in [m]$, client C and servers call $\mathcal{F}_{\mathsf{RSS.Open}}$ with $[\![c^{(j)}]\!]$ to robustly reconstruct $c^{(j)}$. If at least one opening fails, C aborts the protocol.

Output: Client C outputs $((c^{(1)})^g, \ldots, (c^{(m)})^g)$.

Protocol 20 $\widehat{\Pi}_{\mathsf{VIP}}$ – Fiat–Shamir Variant of Verifiable Computation for Inner Products over RSS in the $\mathcal{F}_{\mathsf{Coin}}$-Hybrid Model

Players: Servers $S_1, \ldots, S_n$, where S_i is a dealer.

Input: Servers hold $\{[\![a_i^{(\ell)}]\!], [\![b_i^{(\ell)}]\!]\}_{\ell\in[L]}$ and $\{[\![s_1^{(k)}]\!], [\![s_2^{(k)}]\!], [\![s_3^{(k)}]\!]\}_{k\in[\lceil\log L\rceil]}$, where S_i also knows $\{a_i^{(\ell)}, b_i^{(\ell)}\}_{\ell\in[L]}$ and $\{s_1^{(k)}, s_2^{(k)}, s_3^{(k)}\}_{k\in[\lceil\log L\rceil]}$.

1: Let $\gamma \leftarrow \lceil \log L \rceil$.
2: **for** $k = 1, \ldots, \gamma$ **do**
3: If L is odd, let $L \leftarrow L + 1$.
4: Servers define (or update) polynomials $f_1, f_2, \ldots, f_L$, and $q^{(k)}$ as defined in Eqs. (2) and (3).
5: S_i computes $\eta_j^{(k)} = q^{(k)}(j) - s_j^{(k)}$ for $j = 1, 2, 3$.
6: S_i computes $r_k = H(\eta_1^{(1)}||\eta_2^{(1)}||\eta_3^{(1)}||\cdots||\eta_1^{(k)}||\eta_2^{(k)}||\eta_3^{(k)})$.
7: S_i sets $\{a_i^{(\ell)}, b_i^{(\ell)}\} \leftarrow \{f_{2\ell-1}(r_k), f_{2\ell}(r_k)\}$ for all $\ell \in [L/2]$, and $d_i \leftarrow q^{(k)}(r_k)$, and sets $L \leftarrow L/2$.
8: **end**
9: S_i shares $(q^{(k)}(1), q^{(k)}(2), q^{(k)}(3))_{k\in[\gamma]}$ to all servers using VSS and sends $(\eta_1^{(k)}, \eta_2^{(k)}, \eta_3^{(k)})_{k\in[\gamma]}$.
10: Servers locally compute $[\![c_i]\!] \leftarrow [\![q_1(1)]\!] + [\![q_1(2)]\!]$ and $[\![d_i]\!] \leftarrow [\![c_i]\!]$.
11: Servers initialize L.
12: **for** $k = 1, \ldots, \gamma$ **do**
13: If L is odd, let $L \leftarrow L + 1$.
14: Servers compute r_k as above.
15: Servers use Lagrange interpolation with $\{[\![a_i^{(j)}]\!], [\![b_i^{(j)}]\!]\}_{j\in[L]}$ to locally compute $\{[\![a_i^{(\ell)}]\!], [\![b_i^{(\ell)}]\!]\} \leftarrow \{[\![f_{2\ell-1}(r_k)]\!], [\![f_{2\ell}(r_k)]\!]\}$ for all $\ell \in [L/2]$.
16: Servers use Lagrange interpolation with $(q^{(k)}(1), q^{(k)}(2), q^{(k)}(3))$ to locally compute $[\![d_i]\!] \leftarrow [\![q^{(k)}(r_k)]\!]$.
17: Servers set $L \leftarrow L/2$.
18: **end**
19: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\epsilon_2, \ldots, \epsilon_\gamma \in \mathbb{F}_p$ and locally updates $[\![\sigma_i]\!] \leftarrow \sum_{k=2}^{\gamma} \epsilon_k([\![d_i]\!] - [\![q^{(k)}(1)]\!] - [\![q^{(k)}(2)]\!])$.

Output: Servers record $([\![d_i]\!], [\![a_i^{(1)}]\!], [\![b_i^{(1)}]\!], [\![\sigma_i]\!], [\![c_i]\!])$.

party and it requires $\mathsf{r}_{\mathsf{Share}} = 1$ round. The second one is Protocol 9. The communication cost for the online phase is at most $\mathsf{c}_{\mathsf{Share}} \approx 1.5$ elements per party, and it requires $\mathsf{r}_{\mathsf{Share}} = 2$ rounds.

- Let $\mathsf{c}_{\mathsf{Input}}$ and $\mathsf{r}_{\mathsf{Input}}$ denote the communication cost per server and the number of rounds of Π_{Input} (Protocol 7) for all m client inputs. Each additive component of each mask is sent to C by exactly one designated server and each server sends a single hash covering all m masks, so the servers send $mN + n$ field elements to C in total; C then sends m elements to each server. Hence $\mathsf{c}_{\mathsf{Input}} = m(N/n + 1) + 1$ elements per party, and $\mathsf{r}_{\mathsf{Input}} = 3$ rounds standalone. Note that a direct client sharing would instead cost $m \cdot \mathsf{c}_{\mathsf{Share}} = mN'$ elements per party, so Π_{Input} is cheaper whenever $N/n + 1 < N'$, which holds for every (n, t) we report. In Π^m_{dVOPRF} the echo of Π_{Input} is merged into the hash exchange the protocol already performs at Step 6, so it contributes only $\mathsf{r}_{\mathsf{Input}} - 1 = 2$ rounds there; $\Pi^m_{\mathsf{naive\text{-}dVOPRF}}$ has no such exchange and pays all 3.
- Let $\mathsf{c}_{\mathsf{Open}}$ and $\mathsf{r}_{\mathsf{Open}}$ denote the communication cost per server and the number of rounds for (t, n)-RSS Opening. In this analysis, we consider two instantiations of $\mathcal{F}_{\mathsf{RSS.Open}}$. The first one is the traditional RSS opening protocol; all parties send their shares $[\![x]\!]$ to an opening party, and the honest opening party can reconstruct x by checking the consistency of each additive share in RSS shares, even if there are at most $t < n/2$ corrupted parties. The communication cost is $\mathsf{c}_{\mathsf{Open}} = N'$ field elements per party and it requires $\mathsf{r}_{\mathsf{Open}} = 1$ round. The second one is Protocol 10. The communication cost for the online phase is at most $\mathsf{c}_{\mathsf{Open}} \approx (N'' + 2t)$ elements per party,[8] and it requires only $\mathsf{r}_{\mathsf{Open}} = 1$ round.
- Let $\mathsf{c}_{\mathsf{Mul}}$ and $\mathsf{r}_{\mathsf{Mul}}$ denote the communication cost per server and the number of rounds during secure multiplication protocols. The communication cost of Protocol 12 (or the DN multiplication protocol [16]) for the online phase is at most $\mathsf{c}_{\mathsf{Mul}} \approx 1.5$ elements per party, and it requires $\mathsf{r}_{\mathsf{Mul}} = 2$ rounds.
- Let $\mathsf{c}_{\mathsf{Coin}}$ and $\mathsf{r}_{\mathsf{Coin}}$ denote the communication cost per server and the number of rounds for $\mathcal{F}_{\mathsf{Coin}}$. One of its popular instantiations is that parties call $\mathcal{F}_{\mathsf{Rand}}$ to get $[\![r]\!]$ and opens it [13]. Therefore, its communication cost for the online phase is $\mathsf{c}_{\mathsf{Open}}$ per party, since $\mathcal{F}_{\mathsf{Rand}}$ can be realized without any interactions if we allow its setup phase [14]. When applying the above RSS opening procedure, the communication cost for the online phase is at most $\mathsf{c}_{\mathsf{Coin}} = N' \cdot (n-1)$ elements per party, and it requires $\mathsf{r}_{\mathsf{Coin}} = 1$ rounds. When applying Protocol 10, the communication cost for the online phase is at most $\mathsf{c}_{\mathsf{Coin}} \approx N'' + 2t$ elements per party, and it requires $\mathsf{r}_{\mathsf{Coin}} = 1$ rounds.

[8] We here regard a hash value as a single field element for simplicity, though it could be smaller, say, 256 bits.

We give the detailed efficiency comparisons between the naive protocol $\Pi^m_{\text{naive-dVOPRF}}$ (Protocol 19) and our protocol Π^m_{dVOPRF} in Tabs. 6 and 7. We elaborate on how we count them below.

F.1 Communication Costs

Naive dVOPRF protocol $\Pi^m_{\text{naive-dVOPRF}}$. At the beginning of the online phase, client C and the servers run Π_{Input} to produce $(\llbracket x^{(j)} \rrbracket)_{j\in[m]}$, costing $\mathsf{c}_{\mathsf{Input}}$ field elements per server. Servers then call m instances of $\mathcal{F}_{\mathsf{RSS.Mul}}$ in parallel, so each server sends $m \cdot \mathsf{c}_{\mathsf{Mul}}$ field elements to other servers. Servers call $\mathcal{F}_{\mathsf{vrfy}}$ (Fig. 9) with $3m$ RSS shares $\{(\llbracket a^{(j)} \rrbracket, \llbracket b^{(j)} \rrbracket, \llbracket c^{(j)} \rrbracket)\}_{j=1}^m$. An instantiation of $\mathcal{F}_{\mathsf{vrfy}}$ is given in [12, Protocol 4.2], which is the full version of [11]. In this instantiation, (a) servers call $\mathcal{F}_{\mathsf{Coin}}$ m times, which can be computed in the offline phase, (b) each server sends to the other servers an RSS sharing of its additive share of the input products, (c) servers call n instances of $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ with $2m+1$ RSS shares in parallel, and (d) each server calls $\mathcal{F}_{\mathsf{RSS.Open}}$ to reconstruct a certain RSS share $\llbracket \beta \rrbracket$ and use it to check the validity of the computation. Step (b) requires $n\mathsf{c}_{\mathsf{Share}}$ field elements per server, and Step (d) requires $n \cdot \mathsf{c}_{\mathsf{Open}}$ field elements per server. Regarding Step (c), Boyle et al.'s protocol that securely realizes $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ [12, (FS-applied) Protocol 3.3] is executed as follows.

(1) A sharing server S_i repeats $\lceil \log mL \rceil$ iterations for folding polynomials (with the output-randomization procedure at the last iteration). Each iteration calls $\mathcal{F}_{\mathsf{RSS.Share}}$ to share 3 polynomial evaluations $(q^{(k)}(1), q^{(k)}(2), q^{(k)}(3))$ (plus, $q^{(k)}(0)$ and $q^{(k)}(4)$ at the last iteration). Separately, the FS transform contributes $3/n$ additional field elements of communication per server (except for the last iteration, in which $5/n$ additional field elements are sent).
(2) Servers call $\mathcal{F}_{\mathsf{Coin}}$ $\lceil \log mL \rceil$ times to securely update the RSS sharing of $b = 0$.
(3) The servers batch the values $\llbracket q(r) \rrbracket$, $\llbracket f_1(r) \rrbracket$, $\llbracket f_2(r) \rrbracket$, and $\llbracket b \rrbracket$ obtained from all instances into four shared verification values, using m calls to $\mathcal{F}_{\mathsf{Coin}}$ and n invocations of semi-honest secure multiplication. They then call $\mathcal{F}_{\mathsf{RSS.Open}}$ to reconstruct the four batched values and check whether the corresponding batched relations hold.

Note that $\mathcal{F}_{\mathsf{Coin}}$ after the iterations cannot be moved to the offline phase for security reasons. Therefore, the communication cost of an instance of $\mathcal{F}^{\text{abort}}_{\text{provDeg2Rel}}$ is $(3\mathsf{c}_{\mathsf{Share}} + 3/n)(\lceil \log mL \rceil - 1) + 5\mathsf{c}_{\mathsf{Share}} + 5/n + \lceil \log mL \rceil \cdot \mathsf{c}_{\mathsf{Coin}} + 4n \cdot \mathsf{c}_{\mathsf{Open}}$.[9] Therefore, $\mathcal{F}_{\mathsf{vrfy}}$ for m secret-shared multiplication triples, instantiated as in [12, Protocol 4.2] requires $(3n\mathsf{c}_{\mathsf{Share}} + 3)(\lceil \log mL \rceil - 1) + 6n\mathsf{c}_{\mathsf{Share}} + 5 + (\lceil \log mL \rceil + m) \cdot \mathsf{c}_{\mathsf{Coin}} + n\mathsf{c}_{\mathsf{Mul}} + 4n \cdot \mathsf{c}_{\mathsf{Open}}$ field elements for the communication cost per server. Finally, client C opens RSS shares, $(\llbracket c^{(1)} \rrbracket, \ldots, \llbracket c^{(m)} \rrbracket)$, among servers, which requires $m \cdot \mathsf{c}_{\mathsf{Open}}$ field elements for communication cost per server.

Overall, the communication cost per party of the naive dVOPRF protocol $\Pi^m_{\text{naive-dVOPRF}}$ is

$$\mathsf{c}_{\mathsf{Input}} + 6n \cdot \mathsf{c}_{\mathsf{Share}} + (m+n) \cdot \mathsf{c}_{\mathsf{Mul}} + (\lceil \log mL \rceil + m) \cdot \mathsf{c}_{\mathsf{Coin}} + (3n\mathsf{c}_{\mathsf{Share}} + 3)(\lceil \log mL \rceil - 1) + (4n+m) \cdot \mathsf{c}_{\mathsf{Open}} + 5$$

field elements. When we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ as the naive protocols, and $\mathcal{F}_{\mathsf{Coin}}$, and $\mathcal{F}_{\mathsf{RSS.Mul}}$ as stated at the beginning of Appendix F, it turns to

$$N'((4n-1)\lceil \log mL \rceil + (n+1)m + 7n) + 3\lceil \log mL \rceil + 1.5(m+n) + 2$$

field elements.

On the other hand, if we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocols 9 and 10, then we have

$$(N'' + 2t)(\lceil \log mL \rceil + 2m + 4n) + (4.5n + 3)\lceil \log mL \rceil + 3m + 6n + 2$$

field elements.

Our dVOPRF protocol Π^m_{dVOPRF}. We start with the sub-protocols.

The VIP Protocol Π_{VIP}. For each iteration $k \in [\lceil \log L \rceil]$, server S_i shares 3 field elements $(q^{(k)}(1), q^{(k)}(2), q^{(k)}(3))$ with 3 more field elements $(\eta_1^{(k)}, \eta_2^{(k)}, \eta_3^{(k)})$, where $\eta_j^{(k)} \coloneqq q^{(k)}(j) - s_j^{(k)}$ as in Protocol 20, and then servers call $\mathcal{F}_{\mathsf{Coin}}$ to generate $\epsilon_2, \ldots, \epsilon_{\lceil \log L \rceil}$. Hence, the communication cost is $(3\mathsf{c}_{\mathsf{Share}} + 3/n) \cdot \lceil \log L \rceil + \mathsf{c}_{\mathsf{Coin}} \cdot (\lceil \log L \rceil - 1)$ field elements per server.

The Batched VIP Protocol $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$. Servers invoke n instances of Π_{VIP} in parallel. For every $k \in [\lceil \log L \rceil]$, the challenge r_k is generated synchronously across all instances (by FS transform), and the common randomness ϵ_k is used in all instances. We suppose that $(\epsilon_k)_{k\in[\lceil \log L \rceil]}$ are shared among all servers by calling $\mathcal{F}_{\mathsf{Coin}}$ after the iteration steps of all the instances of the single VIP protocol Π_{VIP}. Therefore, those n instances of Π_{VIP} requires $(3n\mathsf{c}_{\mathsf{Share}}+3) \cdot \lceil \log L \rceil + \mathsf{c}_{\mathsf{Coin}} \cdot (\lceil \log L \rceil - 1)$ field elements for the communication cost per server. Servers also call $n+1$ instances of $\mathcal{F}_{\mathsf{RSS.Mul}}$ in parallel, so each server sends $(n+1)\mathsf{c}_{\mathsf{Mul}}$ field elements to other servers. Finally, servers call $\mathcal{F}_{\mathsf{Coin}}$ to get randomness ρ, which requires $\mathsf{c}_{\mathsf{Coin}}$ field elements per server for the communication cost. Overall, the communication cost of $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ is $(3n\mathsf{c}_{\mathsf{Share}}+3) \cdot \lceil \log L \rceil + \mathsf{c}_{\mathsf{Coin}} \cdot \lceil \log L \rceil + (n+1)\mathsf{c}_{\mathsf{Mul}}$ field elements per server. An important observation here is that, as can be seen, the communication cost of $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ is logarithmic in the number of terms L to be shared.

Our dVOPRF protocol Π^m_{dVOPRF}. Now, we are ready to analyze the communication cost of Π^m_{dVOPRF}. Client C and the servers run Π_{Input} to produce $(\llbracket x^{(j)} \rrbracket)_{j\in[m]}$, costing $\mathsf{c}_{\mathsf{Input}}$ field elements per server. Each server S_i sends a hash value h_i to the other servers, incurring $n-1$ communication costs per server.[10] Servers invoke $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ for mL terms, which requires $(3n\mathsf{c}_{\mathsf{Share}}+3) \cdot \lceil \log mL \rceil + \mathsf{c}_{\mathsf{Coin}} \cdot \lceil \log mL \rceil + (n+1)\mathsf{c}_{\mathsf{Mul}}$ field elements for communications. Finally, client C opens RSS shares, $(\llbracket W(\rho) \rrbracket, \llbracket U(\rho) \rrbracket, \llbracket V(\rho) \rrbracket, \llbracket \Sigma \rrbracket, \llbracket c \rrbracket)$, among servers, and receives additive shares $(v_i^{(1)}, \ldots, v_i^{(m)})_{i=1}^n$ from all servers. Therefore, each server sends $(5 \cdot \mathsf{c}_{\mathsf{Open}} + m)$ field elements to C.

Overall, the communication cost of our dVOPRF protocol Π^m_{dVOPRF} is

$$\mathsf{c}_{\mathsf{Input}} + (3n\mathsf{c}_{\mathsf{Share}} + \mathsf{c}_{\mathsf{Coin}} + 3) \cdot \lceil \log mL \rceil + (n+1)\mathsf{c}_{\mathsf{Mul}} + 5 \cdot \mathsf{c}_{\mathsf{Open}} + m + n - 1$$

field elements. When we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ as the naive protocols, and $\mathcal{F}_{\mathsf{Coin}}$, and $\mathcal{F}_{\mathsf{RSS.Mul}}$ as stated at the beginning

[9] Although we may apply the batching technique [11, Section 3] to reduce the communication cost, we do not consider it here as it requires $\mathcal{F}_{\mathsf{RSS.Mul}}$ that increases the number of rounds.

[10] We here regard a hash value as a single field element for simplicity, though it could be smaller, say, 256 bits.

Table 6: Efficiency comparison for online phases when instantiating $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by simple ones (i.e., generating/receiving shares and distributing/reconstructing them).

(n,t)	Protocol (online)	$m=1$ Comm.	$m=1$ Rnds	$m=35$ Comm.	$m=35$ Rnds	$m=100$ Comm.	$m=100$ Rnds
$(3,1)$	$\Pi^m_{\text{naive-dVOPRF}}$	109	13	557	13	1 224.5	13
$(3,1)$	Our Π^m_{dVOPRF}	**72**	**9**	**299**	**9**	**544**	**9**
$(5,2)$	$\Pi^m_{\text{naive-dVOPRF}}$	840	13	2 598	13	4 957.5	13
$(5,2)$	Our Π^m_{dVOPRF}	**633**	**9**	**1 354**	**9**	**1 731**	**9**
$(7,3)$	$\Pi^m_{\text{naive-dVOPRF}}$	5 485	13	13 215	13	23 888.5	13
$(7,3)$	Our Π^m_{dVOPRF}	**4 470**	**9**	**7 423**	**9**	**8 964**	**9**
$(9,4)$	$\Pi^m_{\text{naive-dVOPRF}}$	32 056	13	66 302	13	113 230.5	13
$(9,4)$	Our Π^m_{dVOPRF}	**27 373**	**9**	**40 182**	**9**	**46 128**	**9**

Table 7: Efficiency comparison for online phases when instantiating $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocols 9 and 10.

(n,t)	Protocol (online)	$m=1$ Comm.	$m=1$ Rnds	$m=35$ Comm.	$m=35$ Rnds	$m=100$ Comm.	$m=100$ Rnds
$(3,1)$	$\Pi^m_{\text{naive-dVOPRF}}$	105.5	15	526	15	1 182.5	15
$(3,1)$	Our Π^m_{dVOPRF}	**66**	**10**	**265.5**	**10**	**499.5**	**10**
$(5,2)$	$\Pi^m_{\text{naive-dVOPRF}}$	381	15	1 245.5	15	2 611.5	15
$(5,2)$	Our Π^m_{dVOPRF}	**225.5**	**10**	**529**	**10**	**822.5**	**10**
$(7,3)$	$\Pi^m_{\text{naive-dVOPRF}}$	1 126.5	15	3 087	15	6 415.5	15
$(7,3)$	Our Π^m_{dVOPRF}	**575**	**10**	**1 090.5**	**10**	**1 656.5**	**10**
$(9,4)$	$\Pi^m_{\text{naive-dVOPRF}}$	3 688	15	9 138.5	15	18 746	15
$(9,4)$	Our Π^m_{dVOPRF}	**1 542.5**	**10**	**2 624**	**10**	**3 879**	**10**

of Appendix F, it turns to

$$N'((4n-1)\lceil \log mL\rceil + m + 5) + 3\lceil \log mL\rceil + m + 2.5n + 0.5$$

field elements.

On the other hand, if we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocols 9 and 10, then we have

$$(N'' + 2t)(\lceil \log mL\rceil + 5) + (4.5n + 3)\lceil \log mL\rceil + 2.5(m+n) + 0.5$$

field elements.

F.2 Round Complexity

We here consider the same instantiations of $\mathcal{F}_{\mathsf{RSS.Share}}$, $\mathcal{F}_{\mathsf{RSS.Open}}$, $\mathcal{F}_{\mathsf{RSS.Mul}}$, $\mathcal{F}_{\mathsf{vrfy}}$, and $\mathcal{F}_{\mathsf{Coin}}$ as in the previous section.

Naive dVOPRF protocol $\Pi^m_{\text{naive-dVOPRF}}$. For the online phase, it runs Π_{Input} at Step 2, and calls $\mathcal{F}_{\mathsf{RSS.Mul}}$ at Step 4, $\mathcal{F}_{\mathsf{vrfy}}$ at Step 5, and $\mathcal{F}_{\mathsf{RSS.Open}}$ at the last step. Since this protocol performs no server-to-server hash exchange of its own, the echo of Π_{Input} cannot be absorbed and costs a separate round. In particular, $\mathcal{F}_{\mathsf{vrfy}}$ can be instantiated with invocations of $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{RSS.Share}}$, $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$, and $\mathcal{F}_{\mathsf{RSS.Open}}$, though $\mathcal{F}_{\mathsf{Coin}}$ can be moved to the offline phase, so we do not count its rounds. Boyle et al.'s protocol that realizes $\mathcal{F}^{\mathsf{abort}}_{\mathsf{provDeg2Rel}}$ [12, Protocol 3.3] invokes $\mathcal{F}_{\mathsf{RSS.Share}}$, $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{RSS.Mul}}$, and $\mathcal{F}_{\mathsf{RSS.Open}}$. Therefore, Boyle et al.'s instantiation of $\mathcal{F}_{\mathsf{vrfy}}$ [12, Protocol 4.2] requires $(2\mathsf{r}_{\mathsf{Share}} + \mathsf{r}_{\mathsf{Coin}} + \mathsf{r}_{\mathsf{Mul}} + 2\mathsf{r}_{\mathsf{Open}})$ rounds. Overall, the number of rounds of the online phase of $\Pi^m_{\text{naive-dVOPRF}}$ is

$$\mathsf{r}_{\mathsf{Input}} + 2\mathsf{r}_{\mathsf{Share}} + \mathsf{r}_{\mathsf{Coin}} + 2\mathsf{r}_{\mathsf{Mul}} + 3\mathsf{r}_{\mathsf{Open}},$$

namely 13 rounds with the instantiations we employ. If we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocols 9 and 10, then we have 15 rounds.

Our dVOPRF protocol Π^m_{dVOPRF} includes two sub-protocols.

The VIP protocol Π_{VIP}. By applying the FS transform as in Boyle et al. [11], or by moving $\mathcal{F}_{\mathsf{Coin}}$ in the iterations to the offline phase, the iterations in Π_{VIP} can be non-interactive, and hence all elements computed during the iterations can be shared among other servers in one shot. Π_{VIP} then calls $\mathcal{F}_{\mathsf{Coin}}$, and hence, it requires $\mathsf{r}_{\mathsf{Share}} + \mathsf{r}_{\mathsf{Coin}}$ rounds in total.

The batched VIP protocol $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$. Servers invoke n instances of Π_{VIP} in parallel, which can be done in $(\mathsf{r}_{\mathsf{Share}} + \mathsf{r}_{\mathsf{Coin}})$ rounds. $\mathcal{F}_{\mathsf{Coin}}$ to generate $\epsilon'_1, \ldots, \epsilon'_n$ can be simultaneously called with $\mathcal{F}_{\mathsf{Coin}}$ at the last step of the single VIP protocol Π_{VIP}, which shares the same randomness $\{\epsilon_k\}_{k\in[\lceil \log L\rceil]}$ across all the instances. Servers also call $n+1$ instances of $\mathcal{F}_{\mathsf{RSS.Mul}}$ in parallel, which can also be done in $\mathsf{r}_{\mathsf{Mul}}$ rounds. Finally, servers call $\mathcal{F}_{\mathsf{Coin}}$ to get ρ, requiring $\mathsf{r}_{\mathsf{Coin}}$ rounds. Therefore, the total number of rounds in $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ is $\mathsf{r}_{\mathsf{Share}} + 2\mathsf{r}_{\mathsf{Coin}} + \mathsf{r}_{\mathsf{Mul}}$. Note that we here assume that $\mathcal{F}_{\mathsf{Rand}}$ is called in the offline phase.

Our dVOPRF protocol Π^m_{dVOPRF}. For the online phase, it runs Π_{Input} at Step 2, which requires $\mathsf{r}_{\mathsf{Input}} - 1 = 2$ rounds here because its echo is absorbed into Step 6. Step 6 requires 1 round to exchange hash values, and carries that echo at no extra cost. Π^m_{dVOPRF} invokes $\Pi^{\mathsf{Prl}}_{\mathsf{VIP}}$ at Step 10, and calls $\mathcal{F}_{\mathsf{RSS.Open}}$ at Step 11. Overall, the number of rounds of the online phase of Π^m_{dVOPRF} is

$$(\mathsf{r}_{\mathsf{Input}} - 1) + \mathsf{r}_{\mathsf{Share}} + 2\mathsf{r}_{\mathsf{Coin}} + \mathsf{r}_{\mathsf{Mul}} + \mathsf{r}_{\mathsf{Open}} + 1,$$

namely 9 rounds with the instantiations we employ. If we instantiate $\mathcal{F}_{\mathsf{RSS.Share}}$ and $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocols 9 and 10, then we have 10 rounds.

G Optimizations and Extensions

We present two optimizations of Π^m_{dVOPRF}. The stratum counting and the detailed cost analysis behind them are given in Appendices G.3 and G.4.

G.1 Saving Terms by Applying Local Share Conversion

In our dVOPRF protocol Π^m_{dVOPRF}, L $(= N^2/n)$ terms of the desired product $c = ab = ([a]_1 + \cdots + [a]_N)([b]_1 + \cdots + [b]_N) = \sum_{u=1}^N \sum_{v=1}^N [a]_u \cdot [b]_v$ are assigned to each server. In this section, we present a technique to reduce the number of terms each server actually computes under the VIP protocol by employing the local share conversion. Specifically, each server S_i can use the local share conversion to non-interactively share N/n terms of c_i, which consists of L terms, among other servers. Therefore, it is sufficient for S_i to execute the VIP protocol Π_{VIP} with $L - N/n$ $(= N(N-1)/n)$ terms.

Stratified Balanced Assignments of Shares. First, we revisit the share-assignment algorithm proposed in §6.1 to appropriately utilize the above technique. Recall the set of eligible servers $\mathcal{E}_{u,v} \coloneqq$

$[n] \setminus (T_u \cup T_v)$ for term (u, v) for all $T_u, T_v \in \mathcal{T}$. In other words, $|\mathcal{E}_{u,v}|$ servers share the same term $[a]_u \cdot [b]_v$. Since we have $1 \leq |\mathcal{E}_{u,v}| \leq t+1$ in the case of $n = 2t+1$, the local share conversion can be applied to any term $[a]_u \cdot [b]_v$ such that the corresponding set $\mathcal{E}_{u,v}$ includes $t+1$ eligible servers.

For every $k \in [t+1]$, let $\mathcal{P}_k \coloneqq \{(u, v) \in [N]^2 \mid |\mathcal{E}_{u,v}| = k\}$ be a *k-th stratum*, which represents the set of terms $([a]_u, [b]_v)$ that exactly k eligible servers share. Clearly, $N^2 = \sum_{k=1}^{t+1} |\mathcal{P}_k|$ holds. Only terms in the top stratum $\mathcal{P}_{t+1}$, shared by $t+1$ eligible servers, admit local share conversion, and $|\mathcal{P}_{t+1}| = N$. Applying the balanced assignment of §6.1 to each stratum separately yields a *stratified balanced share assignment* that still gives every server L terms while letting each server share its N/n top-stratum terms locally (Appendix G.3).

The Most Effective Case: $n = 3$. Let us consider the smallest case, i.e., $t = 1$ and $n = 3$, with RSS. Suppose that for a secret γ, three random elements $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{F}_p$ are chosen under the constraint that $\gamma = \gamma_1 + \gamma_2 + \gamma_3$ (we here denote like this rather than $\gamma = [\gamma]_1 + [\gamma]_2 + [\gamma]_3$ for simple notation). S_1, S_2, and S_3 have their own shares $[\![\gamma]\!]_1 = (\gamma_2, \gamma_3)$, $[\![\gamma]\!]_2 = (\gamma_3, \gamma_1)$, and $[\![\gamma]\!]_3 = (\gamma_1, \gamma_2)$. For simplicity, we abuse the notation $[\![\gamma]\!]_i = (\gamma_{i+1}, \gamma_{i-1})$.[11]

Now, S_i wants to compute $c_i = a_{i+1}b_{i+1} + a_{i+1}b_{i-1} + a_{i-1}b_{i+1}$ and distribute it as RSS shares to S_{i-1} and S_{i+1}, as $ab = (a_1+a_2+a_3)(b_1+b_2+b_3) = (a_2b_2+a_3b_2+a_2b_3)+(a_3b_3+a_3b_1+a_1b_3)+(a_1b_1+a_2b_3+a_3b_2) = c_1+c_2+c_3$. A naive approach is to apply Protocol 5 or its Fiat–Shamir version (Protocol 20), which requires two iterations since the number of terms in c_i is three, i.e., $\lceil \log 3 \rceil = 2$.

The key observation here is that every pair of (S_{i-1}, S_{i+1}) has both a_i and b_i for $i = 1, 2, 3$. Therefore, we can employ the local share conversion technique for $a_i \cdot b_i$; namely, servers can locally compute $[\![a_i \cdot b_i]\!] = \{[a_i \cdot b_i]_{i-1}, [a_i \cdot b_i]_i, [a_i \cdot b_i]_{i+1}\} = \{(a_i \cdot b_i, 0), (0, 0), (0, a_i \cdot b_i)\}$. Thus, it suffices for S_i to compute and distribute $c_i = a_{i-1}b_{i+1} + a_{i+1}b_{i-1}$, thereby reducing the number of iterations from two to one and saving 50% of the communication cost.

General Case. In a similar way to the above, S_i's partial sum $c_i = \sum_{(u,v)\in\widehat{\Lambda}_i} [a]_u \cdot [b]_v$ can be divided into $\hat{c}_i = \sum_{(u,v)\in\widehat{\Lambda}_i \setminus \mathcal{P}_{t+1}} [a]_u \cdot [b]_v$ and $\tilde{c}_i = \sum_{(u,v)\in\mathcal{P}_{t+1}} [a]_u \cdot [b]_v$, where $\widehat{\Lambda}_i$ is a stratified balanced share assignment for S_i. Then, each S_i runs the VIP protocol Π_{VIP} with $\hat{c}_i$ and shares $\tilde{c}_i$ via using the local share conversion technique.[12] Servers then locally compute $[\![c_i]\!] \coloneqq [\![\hat{c}_i]\!] + [\![\tilde{c}_i]\!]$.

For any $t > 1$ and $n > 3$, unfortunately, the number of iterations remains the same, as $\lceil \log L \rceil = \lceil \log(L - N/n) \rceil$ holds. However, this technique still reduces the number of polynomials to be computed to $f_1, \ldots, f_{L-N/n}$, which contributes to the efficiency improvement.

G.2 Faster Online Protocol with Key-Dependent Offline Preprocessing

Protocol 21 gives an online-optimized variant of Protocol 8 by moving the expensive verifiable multiplication step to the preprocessing phase. In Protocol 8, the servers precompute only the input-independent randomness $[\![(\alpha^{(j)})^e]\!]$, and the verifiable multiplication $[\![x^{(j)} + k]\!] \cdot [\![(\alpha^{(j)})^e]\!]$ is performed after the client input $x^{(j)}$ is fixed. In contrast, Protocol 21 performs this multiplication in advance by replacing the unknown client input with a random mask $r^{(j)}$.

This optimization should be interpreted as *key-dependent* preprocessing. The preprocessed material depends on the shared PRF key $[\![k]\!]$, but remains independent of the client input. More precisely, once $[\![k]\!]$ has been fixed, the servers sample, for each future query $j \in [m]$, a random mask $[\![r^{(j)}]\!]$ and a random power $[\![(\alpha^{(j)})^e]\!]$, and then run the same verifiable multiplication as in Protocol 8 to obtain an RSS sharing $[\![c^{(j)}]\!]$ of $c^{(j)} \coloneqq (r^{(j)} + k)(\alpha^{(j)})^e$. During this preprocessing, the servers also prepare the aggregated verification values $[\![\Delta]\!]$ and $[\![\Sigma]\!]$. Thus, a preprocessing record consists of $([\![r^{(j)}]\!], [\![(\alpha^{(j)})^e]\!], [\![c^{(j)}]\!])$ together with the shared verification values. Each such record is intended to be consumed once.

Once the client input $x^{(j)}$ is later provided, the online phase requires no secure multiplication and no execution of the VIP protocol. The client first shares $x^{(j)}$ with the servers, who compute $[\![\delta^{(j)}]\!] \coloneqq [\![x^{(j)}]\!] - [\![r^{(j)}]\!]$ and open $\delta^{(j)}$ among themselves. Since $r^{(j)}$ is random and unknown before the opening, this masked value does not reveal the client input. The servers then locally correct the preprocessed value by computing $[\![v^{(j)}]\!] \coloneqq [\![c^{(j)}]\!] + \delta^{(j)} \cdot [\![(\alpha^{(j)})^e]\!]$. By linearity, the reconstructed value satisfies $v^{(j)} = (r^{(j)} + k)(\alpha^{(j)})^e + (x^{(j)} - r^{(j)})(\alpha^{(j)})^e = (x^{(j)} + k)(\alpha^{(j)})^e$. Therefore, the client can compute $(v^{(j)})^g = (x^{(j)} + k)^g$, as in the basic protocol.

This variant is particularly useful in deployments where the PRF key is long-lived, whereas client inputs arrive only later. For example, a threshold OPRF service may first agree on and secret-share a service key k, and then generate many single-use preprocessing records during idle time. These records can later be consumed by clients whose inputs were not known during preprocessing.

The trade-off is that the preprocessing records are tied to the specific shared key $[\![k]\!]$. If the key is refreshed or replaced, the corresponding records cannot be reused and must be generated again. This is the main distinction between Protocol 21 and the fully key-independent preprocessing used to generate only $[\![(\alpha^{(j)})^e]\!]$.

The online phase of Protocol 21 runs no secure multiplication and no VIP protocol: its communication consists of Π_{Input} and batched RSS openings, grows linearly in m, and takes 4 rounds under our instantiations, independently of m (Appendix G.4).

Takeaway. Protocol 21 trades key-independent preprocessing for a substantially lighter online phase. After the client inputs are fixed, the servers only open masked differences, perform local linear corrections, and open the final values together with two aggregated verification checks. Hence, when openings are batched, the number of online rounds is independent of m. This makes the protocol well-suited to latency-sensitive deployments with stable server-side keys, while being less attractive when keys are frequently refreshed or when preprocessing must remain reusable across different keys.

G.3 Counting the Strata

"$\mathcal{E}_{u,v}$ includes exactly k servers" means $|T_u \cup T_v| = n - k$, and therefore, we have $|T_u \cap T_v| = |T_u| + |T_v| - |T_u \cup T_v| = t + t - (n - k) = k - 1$. Hence, to derive $|\mathcal{P}_k|$ for every $k \in [t+1]$, it is sufficient to count the number of pairs $(T_u, T_v) \in \mathcal{T}^2$ such that $|T_u \cap T_v| = k - 1$. There are $\binom{n}{k-1}$ intersections $I \coloneqq T_u \cap T_v$; there are $\binom{n-(k-1)}{t-(k-1)}$ combinations of T_u for every intersection I; and there are

[11] Here, $\gamma_{i-1} = \gamma_3$ if $i = 1$, and $\gamma_{i+1} = \gamma_1$ if $i = 3$.

[12] To be precise, S_i shares every term $[a]_u \cdot [b]_v$ for $(u, v) \in \mathcal{P}_{t+1}$ and locally computes $[\![\tilde{c}_i]\!] \coloneqq \sum_{(u,v)\in\mathcal{P}_{t+1}} [\![[a]_u \cdot [b]_v]\!]$.

Protocol 21 Faster dOPRF Protocol in the $(\mathcal{F}_{\mathsf{RandExp}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{RSS.Open}})$-Hybrid Model

Players: Client C and Servers $S_1, \ldots, S_n$.
Input: Client C inputs $(x^{(1)}, \ldots, x^{(m)})$, and servers input $[\![k]\!]$.

1: Servers call $\mathcal{F}_{\mathsf{RandExp}}(e)$ m times and $\mathcal{F}_{\mathsf{Rand}}$ $2m$ times to obtain $([\![(\alpha^{(j)})^e]\!])_{j\in[m]}$, the preprocessing masks $([\![r^{(j)}]\!])_{j\in[m]}$, and the independent input masks $([\![\bar{r}^{(j)}]\!])_{j\in[m]}$ used by Π_{Input}.
2: For every $j \in [m]$, let $[\![a^{(j)}]\!] \leftarrow [\![r^{(j)} + k]\!]$ $(= [\![r^{(j)}]\!] + [\![k]\!])$ and $[\![b^{(j)}]\!] \leftarrow [\![(\alpha^{(j)})^e]\!]$.
3: For every $i \in [n]$ and $j \in [m]$, let $a_i^{(j,\ell)} \leftarrow [a^{(j)}]_{u_\ell}$ and $b_i^{(j,\ell)} \leftarrow [b^{(j)}]_{v_\ell}$ for every $(u_\ell, v_\ell) \in \Lambda_i$, where $[\![a^{(j)}]\!]_i = \{[a^{(j)}]_\ell \mid i \notin T_\ell \in \mathcal{T}\}$, $[\![b^{(j)}]\!]_i = \{[b^{(j)}]_\ell \mid i \notin T_\ell \in \mathcal{T}\}$, and $\{\Lambda_i\}_{i\in[n]}$ is a balanced assignment (defined in §6.1).
4: Servers locally obtain $[\![a_i^{(j,\ell)}]\!]$ and $[\![b_i^{(j,\ell)}]\!]$ via local share conversion for all $i \in [n]$, $j \in [m]$, and $\ell \in [L]$.
5: For every $j \in [m]$, servers invoke $\Pi_{\mathsf{VIP}}^{\mathsf{Prl}}$ with $\{[\![a_i^{(j,\ell)}]\!], [\![b_i^{(j,\ell)}]\!]\}_{\ell\in[L]}$ to obtain proof $([\![W^{(j)}]\!], [\![U^{(j)}]\!], [\![V^{(j)}]\!], [\![\Sigma^{(j)}]\!], [\![c^{(j)}]\!])$.
6: For every $j \in [m]$, servers invoke $\mathcal{F}_{\mathsf{RSS.Mul}}$ with $[\![U^{(j)}]\!]$ and $[\![V^{(j)}]\!]$ to obtain $[\![P^{(j)}]\!] = [\![U^{(j)} \cdot V^{(j)}]\!]$.
7: Servers call $\mathcal{F}_{\mathsf{Coin}}$ to obtain $\epsilon_1, \ldots, \epsilon_m \xleftarrow{\$} \mathbb{F}_p$ and locally compute $[\![\Sigma]\!] = \sum_{j=1}^m \epsilon_j [\![\Sigma^{(j)}]\!]$ and $[\![\Delta]\!] = \sum_{j=1}^m \epsilon_j ([\![W^{(j)}]\!] - [\![P^{(j)}]\!])$.
8: Client C and the servers run Π_{Input} (Protocol 7) on $(x^{(j)})_{j\in[m]}$ with the masks $([\![\bar{r}^{(j)}]\!])_{j\in[m]}$, attaching its echo to the opening of Step 7; the servers obtain $([\![x^{(j)}]\!])_{j\in[m]}$.
9: For every $j \in [m]$, servers locally compute $[\![\delta^{(j)}]\!] = [\![x^{(j)}]\!] - [\![r^{(j)}]\!]$, and then call $\mathcal{F}_{\mathsf{RSS.Open}}$ with $[\![\delta^{(j)}]\!]$ to obtain $\delta^{(j)}$. If at least one opening fails, the servers abort the protocol.
10: Servers locally compute $[\![v^{(j)}]\!] = [\![c^{(j)}]\!] + \delta^{(j)} \cdot [\![(\alpha^{(j)})^e]\!]$.
11: Client C calls $\mathcal{F}_{\mathsf{RSS.Open}}$ to robustly open $\{[\![v^{(j)}]\!]\}_{j\in[m]}$, $[\![\Delta]\!]$, and $[\![\Sigma]\!]$. If at least one opening fails, C aborts the protocol.
12: Client C aborts if $\Delta \neq 0$ or $\Sigma \neq 0$.

Output: Client C outputs $((v^{(1)})^g, \ldots, (v^{(m)})^g)$.

$\binom{t+1}{t-(k-1)}$ combinations of T_v such that $T_v \setminus I$ is disjoint from $T_u \setminus I$ since after choosing T_u, remaining elements outside $I \setminus (T \setminus I)$ are $n - t = t + 1$. Therefore, we have

$$|\mathcal{P}_k| = \binom{n}{k-1} \cdot \binom{2t-k}{t-k+1} \cdot \binom{t+1}{t-k+1}.$$

In particular, the number of terms to which we can apply the local share conversion is

$$|\mathcal{P}_{t+1}| = \binom{n}{t} \cdot \binom{t-1}{0} \cdot \binom{t+1}{0} = N$$

For every $k \in [t+1]$, $|\mathcal{P}_k|$ is divisible by n since it contains $\binom{n}{k-1}$ terms. Therefore, we can realize the *stratified balanced share-assignment algorithm*, which assigns terms $[a]_u \cdot [b]_v$ so that the same number of terms per stratum is assigned to servers while every server can compute the terms only from its own shares. The construction is essentially the same as our balanced share-assignment algorithm in §6.1; we just apply it to every $\mathcal{P}_k$, instead of $\mathcal{T}^2$. Consequently, $\sum_{k=1}^{t+1} \frac{|\mathcal{P}_k|}{n}$ $(= L)$ terms are assigned to each server.

G.4 Efficiency of the Faster Online Protocol

Efficiency Analysis. We analyze only the online cost of Protocol 21, namely the cost incurred after the client inputs $x^{(1)}, \ldots, x^{(m)}$ have been fixed. The key-dependent preprocessing cost is excluded from this analysis. We assume that, before the online phase begins, the servers have already generated $([\![r^{(j)}]\!], [\![(\alpha^{(j)})^e]\!], [\![c^{(j)}]\!])_{j\in[m]}$, where $[\![c^{(j)}]\!]$ is an RSS sharing of $c^{(j)} := (r^{(j)} + k)(\alpha^{(j)})^e$, and have also prepared the aggregated verification values $[\![\Delta]\!]$ and $[\![\Sigma]\!]$.

Communication costs. Let $N := \binom{n}{t}$, $N' := \binom{n-1}{t}$, and $N'' := \binom{n-1}{t-1}$. Let $\mathsf{c}_{\mathsf{Share}}$ and $\mathsf{r}_{\mathsf{Share}}$ denote the communication cost per server and the number of rounds for (t, n)-RSS sharing and also let $\mathsf{c}_{\mathsf{Open}}$ and $\mathsf{r}_{\mathsf{Open}}$ denote the communication cost per server and the number of rounds for (t, n)-RSS sharing.

Since all verifiable multiplication steps have already been completed during preprocessing, the online phase contains no invocation of $\Pi_{\mathsf{VIP}}^{\mathsf{Prl}}$ and no online call to secure multiplication. Its communication consists of three parts. First, the client and the servers run Π_{Input} to produce $([\![x^{(j)}]\!])_{j\in[m]}$, which costs $\mathsf{c}_{\mathsf{Input}}$ field elements per server. Second, the servers open the m masked differences $\delta^{(j)} = x^{(j)} - r^{(j)}$ among themselves, which costs $mn\mathsf{c}_{\mathsf{Open}}$ field elements per server. Third, the servers open the m output sharings $[\![v^{(1)}]\!], \ldots, [\![v^{(m)}]\!]$ together with $[\![\Delta]\!]$ and $[\![\Sigma]\!]$ to the client, which costs $(m+2)\mathsf{c}_{\mathsf{Open}}$ field elements per server. Therefore, the total online communication is $\mathsf{c}_{\mathsf{Input}} + (m(n+1)+2)\mathsf{c}_{\mathsf{Open}}$ field elements per server. When we instantiate $\mathcal{F}_{\mathsf{RSS.Open}}$ as stated at the beginning of Appendix F, it turns to $N'(m(n+1)+2) + m(N/n+1) + 1$ field elements. If we instantiate $\mathcal{F}_{\mathsf{RSS.Open}}$ by Protocol 10, then we have $(N'' + 2t)(m(n+1)+2) + m(N/n+1) + 1$ field elements.

The key improvement over Protocol 8 is that the online phase no longer communicates the batched VIP proof. All proof generation, proof aggregation, and secure multiplication are shifted to the key-dependent preprocessing phase. Thus, once the client inputs are known, the online communication is dominated only by input sharing and RSS openings, and grows linearly in m.

Round complexity. We again count only the rounds after the client inputs are fixed. The key-dependent preprocessing rounds are excluded. The online phase consists of the following sequential steps. First, the client and the servers run Π_{Input}; this costs $\mathsf{r}_{\mathsf{Input}} - 1 = 2$ rounds, since its echo is attached to the opening of the $\delta^{(j)}$ below. Second, after receiving the input sharings, the servers compute $[\![\delta^{(j)}]\!] := [\![x^{(j)}]\!] - [\![r^{(j)}]\!]$ for all $j \in [m]$, and open all $\delta^{(j)}$ values in one batched opening. This costs $\mathsf{r}_{\mathsf{Open}}$ rounds. Third, the servers locally compute $[\![v^{(j)}]\!] := [\![c^{(j)}]\!] + \delta^{(j)}[\![(\alpha^{(j)})^e]\!]$ for all $j \in [m]$, which adds no communication rounds. Finally, the servers open $[\![v^{(1)}]\!], \ldots, [\![v^{(m)}]\!]$ together with $[\![\Delta]\!]$ and $[\![\Sigma]\!]$ in one batched opening, costing another $\mathsf{r}_{\mathsf{Open}}$ rounds. Therefore, the online round complexity is $(\mathsf{r}_{\mathsf{Input}} - 1) + 2\mathsf{r}_{\mathsf{Open}}$. With either instantiation of $\mathcal{F}_{\mathsf{RSS.Open}}$ from the beginning of Appendix F, this is 4 rounds.