

Improved Multi-Party Distributed Point Functions from Oblivious Tensor Evaluation

Kelong Cong¹, Emmanuela Orsini², and Erik Pohle^{3*}

¹ Zama, Paris, France

`kelong.cong@zama.ai`

² Bocconi University, Milan, Italy

`emmanuela.orsini@unibocconi.it`

³ Aarhus University, Aarhus, Denmark

`erik.pohle@cs.au.dk`

Abstract. A multi-party distributed point function (DPF) compresses a point function instance $f_{\alpha,\beta}$, i.e., f evaluates to β at α and to 0 otherwise, into short keys that are distributed to N parties. Each key allows a party to locally derive an additive share of the point function evaluation. DPFs have become a widespread building block for cryptographic protocols and sometimes rely on a distributed setup. In this work, we present an N -party, full-threshold DPF for full-domain evaluation making black-box use of oblivious tensor evaluation. Let L denote the domain size of the point function and λ the security parameter. Instantiated with the non-interactive oblivious tensor evaluation primitive (NIOTE) by Abram et al. (STOC '25) based on learning with errors (LWE), we obtain N -party DPFs with (i) key-size $\mathcal{O}(N \cdot \text{poly}(\lambda) \cdot \log^2 L)$ with a $\log L$ -round distributed setup protocol, or (ii) similar key-size with almost $(\log \log L)$ -rounds (asymptotically $\Theta(\log L)$) but for relevant values of L in practice equal to $\log \log L$, or (iii) key-size $\mathcal{O}(N \cdot \text{poly}(\lambda) \cdot \log L \cdot \sqrt{L})$ with $(\log \log L)$ -round distributed setup (also asymptotically). Our construction improves over most of the state of the art either in terms of key-size or in the setup round complexity.

1 Introduction

A point function $f_{\alpha,\beta} : \mathcal{X} \rightarrow \mathcal{Y}$ is defined by a special position $\alpha \in \mathcal{X}$ and value $\beta \in \mathcal{Y}$, with $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(\alpha') = 0$ for all $\alpha' \neq \alpha$. An N -party t -private *distributed point function* (DPF) compresses this object into short keys so that N parties, holding keys $k_1, \dots, k_N$, can locally compute additive shares of $f_{\alpha,\beta}(x)$ for any public x , without further interaction. Security demands that any subset of up to $t - 1$ keys hides (α, β) , while efficiency requires *succinctness*, namely, keys sublinear in the domain size $|\mathcal{X}|$. Intuitively, a DPF is a compact encoding of a unit vector scaled by β , where the “1”-position is set to α .

DPFs are a fundamental cryptographic building block. They enable function secret sharing for comparison and mixed-mode MPC [BGI19, BCG⁺21, BDSS25], sublinear MPC in the preprocessing model [Cou19], distributed ORAM [Ds17, VH23], privacy-preserving ML [RTPB22, Wag22, JGB⁺24], and multi-server PIR [GI14], and they support private set operations [TSS⁺20, DIL⁺22, GRS22, GGM24, BC25]. They are also at the core of several pseudo-random correlation generators (PCGs) [BCGI18, SGRR19, BCG⁺19, BCG⁺20, AS22, BCG⁺22, BBC⁺24, LXYY25, COPZ25], which enable communication-efficient MPC in the correlated randomness model. In many deployments the dealer is distributed across the parties, giving rise to two efficiency measures: the per-party key-size, and the round and communication complexity of the distributed setup.

In the classical two-party setting, the best known DPFs achieve keys of size $\tilde{O}(\lambda \log L)$ with practical efficiency, where (from now on) $L = |\mathcal{X}|$ and λ is the security parameter [BGI15, BGI16].

* This work was partially done while the author was affiliated with COSIC, KU Leuven.

In contrast, full-threshold multi-party DPFs to date exhibit various trade-offs. Constructions for three parties [BKKO20, AS22] have good computational performance but $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ key-sizes. DPFs from succinct homomorphic secret sharing (HSS) either have $\mathcal{O}(\text{poly}(\lambda, L^\epsilon))$ key-size, for a constant $0 < \epsilon \leq 1$ [ARS24], or achieve polylogarithmic key-sizes [AMR25], and admit setup protocols with $\mathcal{O}(N)$ rounds.

Another line of work has explored non-interactive DPF setups, where parties each broadcast a single message and then locally compute the output. Early feasibility results relied on heavy tools like multi-key FHE: Each party’s key consists of a ciphertext encrypting bits of α and an additive share of the encryption key. The parties locally compute the DPF output, for instance, index-by-index using an appropriate Boolean circuit. To obtain additive shares at the end, parties use the AFS-spooky action when the FHE scheme has almost linear decryption [DHRW16, COS⁺22]. The AFS-spooky encryption scheme needs to be either at least $(\log \log L)$ -layered, or allow bootstrapping, and it further requires a circular security assumption. While FHE-based DPFs have short keys, i.e., $\mathcal{O}(\text{poly}(\lambda) \cdot \log L)$, computing the circuit is computationally very expensive. We are not aware of works that optimize the FHE-based approach for DPFs. Recent results [BDSS25] manage to avoid FHE by using assumptions such as DCR, QR or LWE. However, these DPFs either focus only on the two-party case or still suffer from large key-size, typically super-polylogarithmic in L . In summary, there remains a gap between two-party DPFs with polylogarithmic key-size under standard assumptions and full-threshold N -party DPFs, which either require much larger keys or stronger assumptions.

1.1 Our Contributions

We narrow this gap via a modular construction of full-threshold N -party DPFs that reduces the DPF setup to a sequence of secure tensor products. Our DPFs only support full-domain evaluation, which is sufficient for most applications requiring a distributed DPF setup. All our protocols make black-box use of two-party OTE (Oblivious Tensor Evaluation) and differ only in the public tensoring schedule. In this way, improvements in OTE protocols, in particular, more succinct non-interactive OTE (NIOTE) schemes, directly translate into improved DPF instantiations. Concretely, we instantiate the OTE with the succinct non-interactive scheme of Abram, Malavolta, and Roy [AMR25] based on learning with errors (LWE).

The N -party DPF construction of [ARS24] can also be instantiated by the two-party public-key HSS built from [AMR25], which obtains an asymptotically similar key-size of $\mathcal{O}(N \cdot \text{poly}(\lambda, \log L))$. Our construction avoids the detour in [AMR25] (NIOTE $\rightarrow$ reverse trapdoor hashing $\rightarrow$ succinct HSS $\rightarrow$ N -party DPF) by using NIOTE directly as the tensoring layer in the DPF setup computation. This yields a more direct instantiation and improves concrete efficiency by eliminating intermediate transformations. We view this direction as complementary to FHE-based non-interactive approaches, which are typically less efficient due to the circuit evaluation. Our contribution is two-fold.

N -Party NIOTE. Our starting point is the two-party succinct NIOTE [AMR25], which lets two parties obtain additive shares of a tensor product from a single simultaneous round of interaction. The original construction lacks hasher privacy, which has later been addressed by the same authors in [AMR26]. We revisit the primitive and add perfect correctness via a standard two-party non-interactive rounding protocol. Both changes, adding hasher privacy following [AMR26] and the rounding, keep the communication succinct (Section 3.1).

Then, we lift NIOTE to the N -party setting via pairwise composition. Each party broadcasts a single message consisting of a digest and an encoding, and these same objects are reused by every

other party. From its own local state together with the broadcasts of the others, each party can locally derive additive shares of all cross-terms. Combining these shares with the local term, each party obtains an output share $\mathbf{v}^i$ such that $\sum_i \mathbf{v}^i = (\sum_i \mathbf{x}^i) \otimes (\sum_i \mathbf{y}^i)$ (Section 3.2).

Black-Box N -Party DPF with Polylogarithmic Key-Size. We reduce the DPF (setup) computation to the evaluation of tensor products. Using the N -party NIOTE, we give a DPF construction parametrized by the integer $b \geq 1$ with the following trade-off: For $b = 1$, the DPF has key-size $\mathcal{O}(N \cdot \text{poly}(\lambda, \log L))$ and its distributed setup protocol has $\lceil \log L \rceil$ rounds. For $B = 2^b = \text{poly}(\lambda)$, the DPF has key-size $\mathcal{O}(N \cdot \text{poly}(\lambda, \log L))$ (but with a larger $\text{poly}(\lambda)$ factor) and $\Theta(\log \log B + \log_B L)$ rounds, which for practical values of L can be equal to $\log \log L$ with moderate choices for b . Finally, for $b = \log L$, the DPF keys are not polylogarithmic but still sublinear as $\mathcal{O}(N \cdot \log L \cdot \sqrt{L} \cdot \text{poly}(\lambda))$ with a $\mathcal{O}(\log \log L)$ -round setup protocol (Section 5).

Applications. Multi-party DPFs with polylogarithmic key-size improve communication cost in various applications; some examples are given below.

- *Distributed ORAM with low interaction.* Many DPF- and FSS-based distributed oblivious RAM (DORAM) constructions are organized so that, once a private point query has been generated, each server can process its share using only a local linear pass over the relevant memory structure, e.g., a bucket, with little additional interaction. In wide-area or cross-organization applications, the end-to-end cost is often dominated by the latency of synchronous communication rounds rather than by local computations. In this regime, reducing the address-to-point-query expansion from $\mathcal{O}(\log L)$ rounds to $\mathcal{O}(\log \log L)$ rounds directly lowers the latency of each ORAM access that internally requires one or more private point queries. In Section 6.1 and Appendix B, we show how our multi-party DPF can be used as the address-to-point-query subroutine used inside many DPF- and FSS-based distributed ORAM and RAM-MPC protocols.
- *Improved key-size for PCF for OLE and multiplication triples.* Hasler and Reiser [HR25] construct a two-party pseudo-random correlation function (PCF) for OLE and Beaver-style correlations by decomposing each correlated sample into a structured sum of sparse selector vectors and, internally, distributing many tensor-product terms of the form $(i, j) \mapsto e_{0,i}^{(\ell)} \cdot e_{1,j}^{(\ell')}$ (i.e., entries of $\mathbf{e}_0^{(\ell)} \otimes \mathbf{e}_1^{(\ell')}$). Their tensor-FSS instantiation requires an evaluation symmetry property, which prevents asymmetric optimizations, e.g., exploiting that each party knows the support of its own sparse vector, and leads to a dominant t^2 contribution in the key-size expression, where t is the sparsity parameter of the underlying sparse-LPN distribution used in [HR25]. We show in Section 6.2 that these tensor-sharing instances can instead be realized by a direct NIOTE-based *outer-product* primitive (NIOPE): the two parties hold asymmetric keys, and on input (i, j) they locally compute additive shares that sum to $e_{0,i}^{(\ell)} \cdot e_{1,j}^{(\ell')}$, without reconstructing the full $m_0 m_1$ tensor. This replacement in the Hasler and Reiser protocol preserves correctness and removes the symmetry constraint that produces the t^2 term, yielding a reduction in key-size under the same parameter choices. In addition, we note that replacing the two-party NIOTE call by our N -party NIOTE leads to an N -party outer-product-sharing. Integrating it into a full N -party PCF is left for future work.
- *N -party PCGs for authenticated correlations.* Degree-2 correlations (such as multiplication triples) need to be authenticated to be used in actively secure MPC [BDOZ11, DPSZ12], which turns them into degree-3 correlations. Existing N -party pseudo-random correlation generators (PCGs) for authenticated multiplication triples, e.g., [AS22], express the correlation as a sum of three-party products and require three-party full-threshold DPFs. With an N -party DPF, this

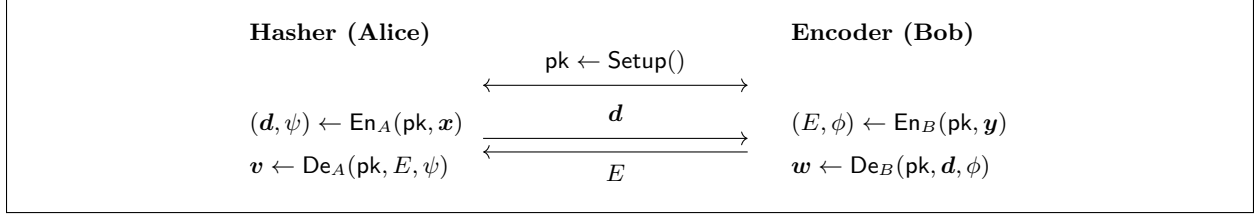


Figure 1: Syntax of succinct NIOTE from [AMR25].

workaround is no longer necessary and the framework of [BCG⁺20] can directly be generalized for N parties. Moreover, instantiated with our DPF, the seed size, or the communication cost of the setup protocol, is then polylogarithmic in the number of outputs of the PCG.

1.2 Technical Overview

We continue with a more detailed description of our techniques.

NIOTE. We start by revisiting the two-party *non-interactive oblivious tensor evaluation* (NIOTE) protocol by Abram, Malavolta and Roy (AMR25) [AMR25]. In the protocol, the hasher Alice compresses her input vector $\mathbf{x} \in \mathbb{Z}_p^m$ into a short digest $\mathbf{d}$. The encoder Bob encodes his input vector $\mathbf{y} \in \mathbb{Z}_p^\ell$ and produces an encoding E . The hasher and encoder exchange $\mathbf{d}$ and E in a single, simultaneous communication round. Then the hasher outputs $\mathbf{v} \in \mathbb{Z}_p^{m\ell}$, computed from the received encoding E and her secret state ψ . Likewise, the encoder outputs $\mathbf{w} \in \mathbb{Z}_p^{m\ell}$ that Bob obtained from the digest $\mathbf{d}$ and his secret state ϕ . The returned values are two-party secret shares $\mathbf{v}, \mathbf{w}$ of the tensor product, i.e., $\mathbf{v} + \mathbf{w} = \mathbf{x} \otimes \mathbf{y}$.

At the start, AMR25 provide an LWE-based instantiation for *half-succinct* NIOTE, where $\mathbf{d}$ comprises $\Theta(\lambda)$ bits and the size of E depends on m and ℓ . Further, they build a *succinct* NIOTE as follows: Fix a block size tn , where t is a compression factor and $n = n(\lambda)$ is an LWE parameter. Further let $m = t^r n$. The hasher arranges her long input vector $\mathbf{x}$ into blocks of length tn . These blocks form the leaves of a t -ary tree, and the hasher and encoder now want to obtain shares of $\mathbf{x}_i \otimes \mathbf{y}$, where $\mathbf{x}_i \in \mathbb{Z}_p^{tn}$. The encoder provides a single encoding, and the hasher computes t^{r-1} digests; however, these digests are not sent. Bob's decoding De_B is bilinear in $\mathbf{d}^i$ and ϕ , and in AMR25, it can be expressed as $\mathbf{P}(\mathbf{d}^i \otimes \phi \otimes \mathbf{g}_q)$ for a public matrix $\mathbf{P}$, where $\mathbf{g}_q = (1, 2, \dots, 2^{\log q})^\top$ is the gadget vector. Therefore, the parties now view $\mathbf{d}^i \otimes (\phi \otimes \mathbf{g}_q)$ as input to a shorter NIOTE instance, for which the hasher again provides t^{r-2} digests and the encoder provides a single encoding. This is repeated until only a single digest is left, which is then sent. Crucially, the parties obtain additive shares of $\mathbf{d}^i \otimes \phi \otimes \mathbf{g}_q$, at each inner level of the tree, and can therefore compute additive shares of De_B . Rearranging and summing gives additive shares of the proper $\mathbf{x}_i \otimes \mathbf{y}$ and thus of $\mathbf{x} \otimes \mathbf{y}$. In total, the encoder provides one encoding for each level of the tree. This way, the digest size is still $\Theta(\lambda)$ and each encoding size depends on tn and ℓ instead of m and ℓ . A summary of the succinct NIOTE primitive is depicted in Figure 1.

Hasher-Private and Perfectly Correct NIOTE. Our use of NIOTE requires hasher privacy and a negligible correctness error.

To obtain hasher privacy, note that only the digest of the half-succinct NIOTE at the root is revealed to the encoder. Therefore, we can replace this half-succinct call with another NIOTE that is hasher-private but not succinct, such as the one given in [ARS24, AMR26]. This maintains the

communication efficiency because input parameters to the root NIOTE are $\Theta(\lambda)$ for both parties, i.e., independent of m or ℓ .

To obtain a negligible correctness error, each party first locally rounds the output of the half-succinct NIOTE at each level in the tree to control the error [DHRW16]. We then lift the shares of each party, which have been rounded to the plaintext modulus p , into the ciphertext modulus q using the LWE-based lifting from Boyle et al. [BKS19]. Since correctness depends on the magnitude of the lifted message, our bootstrapping uses the tensor product of the bit-decomposed digest $\hat{\mathbf{d}}$ and encoder secret $\hat{\phi}$. This unfortunately requires larger parameters⁴ compared to AMR25 due to lifting and bit decomposition but brings necessary properties for our usage. The modular structure of our multi-party DPF construction compensates for this: any advances in efficiency of the NIOTE primitive will be reflected in our design. We discuss more details in Section 3 and give the complete protocol including proofs in Appendix A.

Multi-Party NIOTE. The goal, in the multi-party NIOTE, is that each party outputs $\mathbf{v}^i$, s.t., $\sum \mathbf{v}^i = (\sum \mathbf{x}^i) \otimes (\sum \mathbf{y}^i)$, where $\mathbf{x}^i \in \mathbb{Z}_p^m$ and $\mathbf{y}^i \in \mathbb{Z}_p^\ell$ are the inputs of $\mathcal{P}_i$. Note that the tensor product can be written as pairwise products between all $(\mathcal{P}_i, \mathcal{P}_j)$ as

$$(\sum \mathbf{x}^i) \otimes (\sum \mathbf{y}^j) = \sum_{i,j} \mathbf{x}^i \otimes \mathbf{y}^j.$$

Since the digest computation does not depend on Bob’s input $\mathbf{y}$, and the encoding does not depend on Alice’s input $\mathbf{x}$, each party only creates one digest $\mathbf{d}^i$ and one encoding E^j which every other party $\mathcal{P}_j$ can use to derive their share of $\mathbf{x}^i \otimes \mathbf{y}^j$ and of $\mathbf{x}^j \otimes \mathbf{y}^i$, respectively. The resulting N -party primitive is non-interactive, where each party sends a single message that is succinct in m (the size of $\mathbf{x}^i$).

Multi-Party DPF. At the core of our DPF lies the following observation. A unit vector of length L can be expressed as an iterated tensor product of small “selector” vectors corresponding to the special position α . Concretely, if α is available as shared bits $\{\alpha_t\}_{t \in [\ell]}$, as is typical inside RAM-style MPC where addresses are already represented bitwise, or after an independent bit-decomposition step, then each bit α_t induces a length-2 selector $\mathbf{s}_t := (1 - \alpha_t, \alpha_t)$ whose entries indicate whether the bit is 0 or 1. The full one-hot vector is obtained by tensoring these length-2 selectors across all ℓ bits, which multiplies dimensions from 2 to $2^\ell = L$, i.e.,

$$\mathbf{u}_{\alpha,1} = \mathbf{s}_{\ell-1} \otimes \mathbf{s}_{\ell-2} \otimes \cdots \otimes \mathbf{s}_0. \quad (1)$$

Combined with the multi-party NIOTE to evaluate the tensor products, we derive the following evaluation strategies and corresponding trade-offs.

- *A succinct N -party DPF with $\log L$ -round setup.* Evaluating Equation (1) sequentially, i.e., $(\cdots((\mathbf{s}_0 \otimes \mathbf{s}_1) \otimes \mathbf{s}_2) \otimes \cdots) \otimes \mathbf{s}_{\ell-1}$ uses NIOTE instances with $m = 2^j$ and $\ell = 1$ (for the tensor $((\cdots) \otimes \mathbf{s}_j) \otimes \cdots$). Since the multi-party NIOTE is succinct in m , this construction has per-party DPF keys of $\mathcal{O}(\text{poly}(\lambda, \log L))$. Its sequential structure means the corresponding distributed setup protocol has $\mathcal{O}(\log L)$ rounds and the same communication cost as the key-size. We present an alternative protocol with the same number of rounds but better concrete communication cost in Figure 10 (Section 5.4).

⁴ Our parameters are similar to the parameters of [BKS19].

- *A succinct N -party DPF with $\log \log L$ -round setup.* We can reduce the number of communication rounds in the setup protocol to $\mathcal{O}(\log \log L)$ by evaluating Equation (1) in a balanced binary tree. Thus, in the first round, every two length-2 selectors are tensored, yielding a length-4 vector that encodes two bits. In the second round, every two length-4 vectors are tensored, yielding length-16 vectors that encode four bits, and so on. Since this doubles the number of encoded bits per round, the parties have combined all ℓ bits after $\lceil \log \log L \rceil$ rounds where tensor products on the same level are independent and can be executed in parallel. Note the following caveat: The right input, i.e., ℓ , of each NIOTE invocation doubles per level in the balanced tree. As such, implementing the OTE with the NIOTE of AMR25, which is only succinct on the left input size, this no longer leads to polylogarithmic DPF keys but the keys are still *sublinear* with size $\mathcal{O}(N \cdot \log L \cdot \sqrt{L} \cdot \text{poly}(\lambda))$.
- *A succinct N -party DPF with almost $\log \log L$ -round setup.* We can get the best of both: polylogarithmic keys while obtaining a setup protocol with the same number of rounds as $(\log \log L)$ for values of L used in practice. We fix a block size $b \geq 1$ with $B = 2^b$ and split the ℓ bits of the special position α into $K = \lceil \ell/b \rceil$ blocks. The length-2 selectors within each block are tensored using the balanced tensor tree approach and form K length- B vectors, $\mathbf{s}^{(j)} \in \mathbb{Z}_p^{2^{b_j}}$, with $b_j \leq b$. This step takes exactly $\lceil \log_2 b \rceil$ broadcast rounds and never invokes NIOTE with right-input length exceeding $2^{\lceil b/2 \rceil} \approx \sqrt{B}$. We then combine the K length- B vectors sequentially, i.e., computing $(\mathbf{s}^{(K-1)} \otimes \mathbf{s}^{(K-2)}) \otimes \mathbf{s}^{(K-3)} \otimes \dots \otimes \mathbf{s}^{(0)}$. This requires $K - 1 = \lceil \ell/b \rceil - 1$ rounds, and crucially each NIOTE call in this second phase has right-input length at most B , which is a public constant chosen as $\text{poly}(\lambda)$, avoiding the right-dimension blow-up. Overall, the round complexity is $R(b) = \lceil \log_2 b \rceil + \lceil \ell/b \rceil - 1$. Asymptotically, keeping B independent of L implies $R = \Theta(\log L)$, so we do not achieve the $\Theta(\log \log L)$ round bound of the previous DPF. However, for practically relevant values of L , the total number of rounds $R(b)$ changes only when b crosses certain thresholds, since both terms are integers obtained via ceiling. Consequently, for many values of L used in practice there exists a moderate block size b such that $R(b)$ equals the balanced-tree bound $\lceil \log \log L \rceil$, while the sequential chain phase keeps the right-input length bounded by the public constant $B = 2^b$. For instance, for $\ell = 20$, choosing $b = 7$ (hence $B = 128$) yields $R = 5 = \lceil \log_2 20 \rceil$, i.e., the same number of rounds as in the balanced tensor tree. Notably, since the length of the right-input of any NIOTE is $\text{poly}(\lambda)$, this DPF has polylogarithmic key-size while its corresponding setup protocol has the same number of rounds as the balanced tree approach for practical values of L .

In all constructions, the special value β can be integrated in different ways. Depending on the application, β can be injected early, by scaling the appropriate leaf selector coordinate so that β is carried through the tensoring, or applied at the end by scaling the final one-hot sharing, with no effect on round complexity. In both variants, correctness follows from the fact that exactly the unique address consistent with α survives the structured cancellations induced by the selectors and correction information, while all other coordinates remain shares of zero. Privacy follows from the security of the N -party NIOTE sub-protocol. In Section 5, we give the general construction that first computes length- B blocks via the balanced tensor tree followed by sequential chaining.

1.3 Related Work

Distributed point functions (DPFs), introduced by Gilboa and Ishai [GI14], have since been studied in a range of settings. The literature typically considers two closely related models. In the dealer

Table 1: Overview of $(N - 1)$ -private, N -party DPFs in the literature where N is the number of parties, λ is the security parameter and L is the domain size of the DPF. MRP+DlogVOH stands for MinRank-Pair-Rerandomization and subspace DLog vector oracle heuristic.

Construction	Assumption	Key-Size
[CBM15, KPM24]	DDH	$\mathcal{O}(N \cdot \lambda \cdot \sqrt{L})$
[BGI15]	OWF	$\mathcal{O}(2^{N/2} \cdot \lambda \cdot \sqrt{L})$
[GWW25]	OWF	$\mathcal{O}(N^3 \cdot \lambda \cdot \sqrt{L})$
[ARS24]	DCR $\vee$ DDH over class groups $\vee$ LWE	$\mathcal{O}(N \cdot \text{poly}(\lambda, \sqrt{L}))$
[KPR25]	Factoring $\wedge$ MRP+DlogVOH	$\mathcal{O}(\text{poly}(N, \lambda, \log L))$
[ARS24] using HSS from [AMR25]	LWE	$\mathcal{O}(N \cdot \text{poly}(\lambda, \log L))$
Ours	LWE	$\mathcal{O}(N \cdot \text{poly}(\lambda) \cdot \log^2 L)$

model, a trusted algorithm runs $\text{Gen}(1^\lambda, \alpha, \beta)$ and distributes one key to each party. In this case, the main efficiency metric is the per-party key-size. However, when a trusted dealer is unavailable, the computing parties run a distributed setup that emulates Gen or directly produces shares of the full-domain evaluation. In this case, the key metrics are both communication and round complexity. Accordingly, we review prior work along both axes: DPF evaluation (Table 1) and DPF setup protocols (Table 2).

We start with DPF constructions, focusing on assumptions and key-size. In the original construction of Gilboa and Ishai, the full-domain evaluation of the point function is arranged into a square matrix and each row-vector is compressed using a PRG, resulting in $\approx \sqrt{L}$ λ -bit seeds. A correction word and recursion are used to also compress the “special row” that contains the special value β .

Two-Party DPFs. In this setting, the smallest DPF key-sizes are $\mathcal{O}(\lambda \log L)$ with small constants, stemming from the line of work [BGI15, BGI16], where both keys have the same size. These works instead arrange the full-domain evaluation as leaves of a binary tree and make use of a weak homomorphism of two-party additively shared seeds to a (non-homomorphic) PRG. The “special path” through the tree to the special leaf containing β is encoded with correction words. In an asymmetric setting, one of the keys can even have constant size $\mathcal{O}(\lambda)$, with the other key-size still being polylogarithmic $\mathcal{O}(\text{poly}(\lambda, \log L))$, which is coined as programmability in [BGIK22].

Multi-Party DPFs. In the multi-party setting, the work of [BGI15] constructs $(N - 1)$ -private, N -party DPFs with $\mathcal{O}(2^N \text{poly}(\lambda, \sqrt{L}))$ key-size. The exponential factor in N was improved by [GWW25] to $\mathcal{O}(N^3 \text{poly}(\lambda, \sqrt{L}))$. The works of [CBM15, KPM24] extend the matrix template into the multi-party setting using seed homomorphic PRGs based on DDH and thus achieve $\mathcal{O}(N \lambda \sqrt{L})$ key-size. With different techniques, [ARS24] show DPF constructions from succinct HSS built from DCR, DDH over class groups or LWE with super-polynomial modulus-to-noise ratio, and obtain $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ key-sizes. One can also instantiate the DPF construction from [ARS24] using the succinct HSS built from NIOTE [AMR25], which yields $\mathcal{O}(N \cdot \text{poly}(\lambda, \log L))$ key-size. Compared to our construction, this approach relies on multiple intermediate transformations to turn NIOTE into a public-key HSS, which impacts efficiency. Our DPF is a direct construction from NIOTE which provides a complementary trade-off to [ARS24] for the DPF setup protocol. In unpublished work, [KPR25] propose a DPF with polylogarithmic key-size $\mathcal{O}(\text{poly}(\lambda, \log L))$ assuming factoring and new ad-hoc

assumptions called subspace DLog vector oracle heuristic and MinRank-Pair-Rerandomization. Our construction achieves polylogarithmic dependence on L under LWE (see Table 1), whereas many prior full-threshold multi-party DPFs have key-size $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ under a range of assumptions. Unlike [KPR25], which achieves the same sizes asymptotically, we base our construction on more commonly studied lattice assumptions, which also make our DPF plausibly post-quantum secure. Finally, [CKY25] build a 2^m -party DPF with key-size $\mathcal{O}(\lambda^3)$ for the domain $L = \lambda^d$ (for any constant d) based on LPN and MQ. Their nesting approach targets 2^m parties, whereas our construction applies to any number of parties $N \geq 2$. For completeness, we also note partial-threshold and small-party constructions. In particular, the work [ZYP24] introduces 1-private, three-party DPFs based on two-party DPFs with key-size $\mathcal{O}(\lambda \log L)$ but their technique does not generalize to $N > 3$. A 2-private, three-party DPF was proposed by [BKKO20] and [AS22] with key-sizes $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ for binary extension fields and prime fields, respectively. These two works generalize the matrix approach used in the two-party case.

We summarize relevant full-threshold multi-party constructions in Table 1.

DPF Setup Protocols. In many applications, including sublinear communication for preprocessing in MPC using PCGs [AS22] or distributed oblivious RAM [Ds17, BKKO20, ZYP24, VH23, SVG24], the DPF dealer is distributed among all parties, necessitating setup protocols.

The setup for two-party DPFs based on the tree approach is efficient in $\log L$ rounds [Ds17] in the semi-honest setting, yielding DPF protocols with $\mathcal{O}(\text{poly}(\lambda, \log L))$ communication. Staying in the two-party setting, the setup for [BGIK22] has $\mathcal{O}(1)$ rounds and the (non-interactive) DPFs of [BDSS25, CKY25] only 1. All of them share the increased communication cost of $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$.

For three-party matrix-based DPFs, [BKKO20] and [AS22] provide setup protocols with $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ communication in the semi-honest and malicious setting with $\mathcal{O}(\log \log L)$ rounds, respectively. The setup protocol implied by the multi-party DPF based on succinct homomorphic secret sharing [ARS24] requires $\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$ communication in $\mathcal{O}(N)$ rounds. Our DPF construction is based on NIOTE invocations structured in a balanced or unbalanced binary tree. Each NIOTE invocation requires exactly one communication round where parallel invocations can be sent in the same round. Consequently, depending on the structure of the tree, our setup protocol requires between $\log L$ and $\log \log L$ rounds with polylogarithmic or sublinear communication. We summarize the literature for DPF setup protocols in Table 2.

Other Related Works. Beyond DPFs, which can be viewed as a restricted function secret sharing (FSS) specialized to point functions, general function sharing can be obtained via the additive-function-sharing (AFS) *spooky encryption* of Dodis et al. [DHRW16]. The LWE-based instantiation requires common public randomness and, for negligible correctness error, super-polynomial modulus-to-noise regimes. Chillotti et al. [COS⁺22] propose Scooby, an N -party HSS for general circuits with computation complexity $\mathcal{O}(|F| + \log N)$ and per-party key-size of $\mathcal{O}(\text{poly}(\lambda, \log N))$. For a circuit that evaluates the DPF at a single point of the domain with size L , evaluating the equality circuit costs $\mathcal{O}(\log L)$, followed by the decryption-to-shares procedure that additionally traverses $\mathcal{O}(\log N)$ levels of an FHE-key tree at a per-level cost that is described as being comparable to an FHE bootstrapping operation. Scooby is more expressive but requires a trusted setup, or MPC realization of the setup whose round and communication costs are not analyzed. Our construction is more specialized, only supports full-domain evaluation with explicit per-instance key-size, communication, and round bounds for the distributed generation and avoids evaluating a point-function circuit with

Table 2: DPF Setup Protocols in the literature. We denote with N the number of parties, λ the security parameter and L the domain size of the DPF.

	Parties	Communication	Rounds
[Ds17]	2	$\mathcal{O}(\text{poly}(\lambda, \log L))$	$\log L$
[BGIK22]	2	$\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$	$\mathcal{O}(1)$
[BDSS25, CKY25]	2	$\mathcal{O}(\text{poly}(\lambda, \sqrt{L}))$	1
[BKKO20, AS22]	3	$\mathcal{O}(\sqrt{L} \cdot \text{poly}(\lambda))$	$\mathcal{O}(\log \log L)$
[CKY25]	2^m	$\mathcal{O}(\lambda^3)$ (for $L = \lambda^d$)	1
[ARS24]	N	$\mathcal{O}(N \cdot \text{poly}(\lambda, \sqrt{L}))$	$\mathcal{O}(N)$
[ARS24] using HSS from [AMR25]	N	$N \cdot \text{poly}(\lambda, \log L)$	$\mathcal{O}(N)$
Ours[$b = 1$] (Figure 9)	N	$(N - 1) \cdot (\log^2 L) \cdot \text{poly}(\lambda)$	$\log L$
Ours[$b = \log L$] (Figure 9)	N	$(N - 1) \cdot \mathcal{O}(\log L \cdot \sqrt{L}) \cdot \text{poly}(\lambda)$	$\mathcal{O}(\log \log L)$
Ours[$2^b = \text{poly}(\lambda)$] (Figure 9)	N	$(N - 1) \cdot (\log^2 L) \cdot \text{poly}(\lambda)$	$\log_2 b + \log_{2^b} L$

FHE. The two approaches are complementary, offering different trade-offs between generality, setup assumptions, evaluation functionality, and concrete efficiency.

2 Preliminaries and Notation

We start this section with our notation. We write $N \in \mathbb{N}$ for the number of parties, $L = 2^\ell$ for a DPF domain size, and $\lambda \in \mathbb{N}$ for the security parameter. In the context of LWE, we use a power-of-two ciphertext modulus $q = 2^{\text{poly}(\lambda)}$, a plaintext modulus p with $p \mid q$ and let $\Delta = q/p \geq 2 \cdot \text{poly}(\lambda)$. Let $k = \Theta(\lambda)$ denote the LWE secret dimension, let $\ell_q := \lceil \log_2 q \rceil$, and set $n := k \cdot \ell_q$. We use $m, \ell \in \mathbb{N}$ for (arbitrary) message lengths. The column vector $\mathbf{g}_q = (1, 2, \dots, 2^{\log q})^\top$ is the gadget vector and $\chi(1^\lambda)$ denotes a low-norm distribution over $\mathbb{Z}$ which is $B(\lambda)$ -bounded.

We write column vectors in bold, lowercase letters, e.g., $\mathbf{v} \in \mathbb{Z}_p^m$, and matrices as bold, uppercase, e.g., $\mathbf{M} \in \mathbb{Z}_p^{m \times \ell}$. We denote with $\otimes$ the Kronecker product

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{\ell 1}\mathbf{B} & \dots & a_{\ell m}\mathbf{B} \end{pmatrix} \quad (\text{for } \mathbf{A} \in \mathbb{Z}_p^{\ell \times m}).$$

For $\mathbf{x} \in \mathbb{Z}_q^m$, $\text{Bits}(\mathbf{x}) \in \mathbb{Z}_b^{m \log_2 q}$ denotes the concatenated bit expansion with lowest-order bit first. For any matrix $\mathbf{B} \in \mathbb{Z}_q^{(t \cdot \ell) \times m}$ and vectors $\mathbf{x} \in \mathbb{Z}_q^m$ and $\mathbf{s} \in \mathbb{Z}_q^\ell$, we write the matrix linearization $\text{Lin}(\mathbf{B}) \in \mathbb{Z}_q^{t \times (m \cdot \ell)}$, s.t. $\text{Lin}(\mathbf{B})(\mathbf{x} \otimes \mathbf{s}) = (\mathbf{I}_t \otimes \mathbf{s}^\top) \mathbf{B} \mathbf{x}$ and $\|\text{Lin}(\mathbf{B})\|_\infty = \|\mathbf{B}\|_\infty$ (see [AMR25, Lemma 2.1]).

We use $[x]$ for the set $[x] = \{0, \dots, x - 1\}$, for $x \in \mathbb{N}$. A unit vector with special position α and special value β is denoted as $\mathbf{u}_{\alpha, \beta}$. In the context of secret sharing, $\langle x \rangle$ denotes a full-threshold additive share of x .

We let $\lfloor x \rfloor$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$ as $\lfloor v \rfloor_p = \lfloor \frac{p}{q} \cdot v \rfloor$. Throughout this paper, we represent $x \in \mathbb{Z}_p$ as $x \in \{0, \dots, p - 1\}$. Also define $\text{center}_p(x)$ as the centered representative map

on $\mathbb{Z}_p$, mapping into $(-p/2, \dots, p/2]$ and $\text{uncenter}_p(x)$ mapping back into $\{0, \dots, p-1\}$, as

$$\text{center}_p(x) := \begin{cases} x - p & \text{if } x \geq p/2, \\ x & \text{else,} \end{cases} \quad \text{uncenter}_p(y) := \begin{cases} y & \text{if } y \geq 0, \\ y + p & \text{else,} \end{cases}$$

where y is in centered representation $y \in (-p/2, \dots, p/2]$. For two moduli p, q with $p < q$, a PRF $F_K(\text{cnt}) : \{0, 1\}^\lambda \rightarrow \mathbb{Z}_p$ with key K , the value $x \in \mathbb{Z}_p$ and bit $b \in \{0, 1\}$, we define the following operation $\text{Lift}_K^b(x)$:

$$\text{Lift}_K^b(x) = \text{uncenter}_q(\text{center}_p(\text{center}_p(x) + (2b - 1) \cdot \text{center}_p(F_K(\text{cnt})))) \mod q.$$

Thus, $\text{Lift}_K^b(x)$ lifts a value in $\mathbb{Z}_p$ to $\mathbb{Z}_q$ via the centered representation. This also works for secret-shared x with high probability (see [BKS19] and Corollary 2)⁵. When writing $\text{Lift}_K^b(x)$ it is implicitly understood that the PRF input cnt is unique for each time that a party uses lifting. For example, the party holds a counter and increments it after each use. For a vector $\mathbf{x} \in \mathbb{Z}_p^n$, we extend $\text{Lift}_K^b(\mathbf{x}) \in \mathbb{Z}_q^n$ to component-wise lifting with a fresh PRF input.

2.1 Distributed Point Function

A *point function* $f_{\alpha, \beta} : \mathcal{X} \rightarrow \mathcal{Y}$ parametrized by $\alpha \in \mathcal{X}$ and $\beta \in \mathcal{Y}$ is defined as

$$f_{\alpha, \beta}(x) = \begin{cases} \beta & \text{if } x = \alpha, \\ 0 & \text{otherwise} \end{cases}.$$

We often consider the point function evaluated at every input, resulting in a vector $(f_{\alpha, \beta}(0), f_{\alpha, \beta}(1), \dots)$. We refer to this full-domain evaluation as the unit vector $\mathbf{u}_{\alpha, \beta}$ parametrized by α, β , i.e., $\mathbf{u}_{\alpha, \beta} = (\underbrace{0, \dots, 0}_\alpha, \beta, 0, \dots)$.

Definition 1 (Distributed Point Function [GI14, BGI16]). Let $\mathcal{X}$ be a finite set of size $|\mathcal{X}| = L$ and let $\mathbb{G}$ be a finite (abelian) group written additively. An N -party DPF is a pair of PPT algorithms $(\text{Gen}, \text{FullEval})$ with:

- $\text{Gen}(1^\lambda, \alpha, \beta)$, where $\alpha \in \mathcal{X}$ and $\beta \in \mathbb{G}$, outputs keys $(k_1, \dots, k_N)$.
- $\text{FullEval}(k_i)$ outputs a share $\langle \mathbf{u}_{\alpha, \beta} \rangle_i \in \mathbb{G}^L$.

It must satisfy the following correctness and security requirements:

Correctness: For all $\alpha \in \mathcal{X}$, $\beta \in \mathbb{G}$,

$$\Pr \left[\sum_{i \in [N]} \text{FullEval}(k_i) = \mathbf{u}_{\alpha, \beta} \mid (k_1, \dots, k_N) \leftarrow \text{Gen}(1^\lambda, \alpha, \beta) \right] = 1,$$

where the probability is taken over the internal randomness of Gen .

Security: For every strict subset $T \subsetneq [N]$ there exists a PPT simulator Sim such that, for all $\alpha \in \mathcal{X}$ and $\beta \in \mathbb{G}$, $\{k_j\}_{j \in T} \stackrel{\epsilon}{\approx} \text{Sim}(1^\lambda, \mathcal{X}, \mathbb{G}, T)$, where $(k_1, \dots, k_N) \leftarrow \text{Gen}(1^\lambda, \alpha, \beta)$.

⁵ Note that the PRF is only used for correctness to rerandomize the two-party non-interactive lifting of a secret-shared value.

2.2 Non-Interactive Oblivious Tensor Evaluation

We recall the non-interactive oblivious tensor evaluation (NIOTE) primitive, defined by [AMR25]. Two parties, Alice (the hasher) and Bob (the encoder), exchange a single simultaneous message to obtain a two-party additive share of $\mathbf{x} \otimes \mathbf{y}$, where $\mathbf{x} \in \mathbb{Z}_p^m$ and $\mathbf{y} \in \mathbb{Z}_p^\ell$ are Alice's and Bob's input, respectively. We write En_A for Alice's hashing and En_B for Bob's encoding operations, and De_A, De_B for their respective decoding operations.

Definition 2 (NIOTE [AMR25]). Let $m = m(\lambda)$, $\ell = \ell(\lambda)$, p, q be positive integers. Then a non-interactive oblivious tensor evaluation scheme NIOTE for $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^\ell$ is a tuple of five PPT algorithms $(\text{Setup}, \text{En}_A, \text{En}_B, \text{De}_A, \text{De}_B)$ with the following syntax:

- $\text{Setup}(1^\lambda)$ is a probabilistic algorithm which takes the security parameter 1^λ as input and outputs a public key pk .
- $\text{En}_A(\text{pk}, \mathbf{x})$ is a probabilistic algorithm which takes a vector $\mathbf{x} \in \mathbb{Z}_p^m$ as input and outputs a digest $\mathbf{d}$ and secret information ψ .
- $\text{En}_B(\text{pk}, \mathbf{y})$ is a probabilistic algorithm which takes a vector $\mathbf{y} \in \mathbb{Z}_p^\ell$ as input and outputs an encoding E and secret information ϕ .
- $\text{De}_A(\text{pk}, E, \psi)$ is a deterministic algorithm which takes the encoding E and A 's secret information ψ as input, and returns $\mathbf{v} \in \mathbb{Z}_p^{m\ell}$.
- $\text{De}_B(\text{pk}, \mathbf{d}, \phi)$ is a deterministic algorithm which takes the digest $\mathbf{d}$ and B 's secret information ϕ as input, and returns $\mathbf{w} \in \mathbb{Z}_p^{m\ell}$.

We say a NIOTE is perfectly or fully correct, if there exists a negligible function $\text{negl}(\lambda)$ such that, for all sufficiently large $\lambda \in \mathbb{N}$, and for every $\mathbf{x} \in \mathbb{Z}_p^m$ and every $\mathbf{y} \in \mathbb{Z}_p^\ell$, we have

$$\Pr \left[\left\| \text{De}_A(\text{pk}, E, \psi) + \text{De}_B(\text{pk}, \mathbf{d}, \phi) - \mathbf{x} \otimes \mathbf{y} \right\|_\infty > 0 \mid \begin{array}{l} \text{pk} \leftarrow \text{Setup}(1^\lambda) ; \\ (\mathbf{d}, \psi) \leftarrow \text{En}_A(\text{pk}, \mathbf{x}) ; \\ (E, \phi) \leftarrow \text{En}_B(\text{pk}, \mathbf{y}) \end{array} \right] \leq \text{negl}(\lambda),$$

where the probability is taken over the randomness in the evaluation of Setup , En_A and En_B .

Bilinear Encoder Evaluation. We say that a NIOTE has bilinear encoder evaluation if (i) $\mathbf{d} \in \mathbb{Z}_q^n$, (ii) $\phi \in \mathbb{Z}_q^k$, and (iii) there exists a public matrix $\mathbf{P} \in \mathbb{Z}_q^{(m\ell) \times (nk\ell_q)}$, derivable from pk , such that:

$$\text{De}_B(\text{pk}, \mathbf{d}, \phi) = \mathbf{P} \cdot (\mathbf{d} \otimes \phi \otimes \mathbf{g}_q),$$

where the computation is performed over $\mathbb{Z}_q$ (ciphertext space) and reduced to $\mathbb{Z}_p$ in the output.

Succinctness. Following [AMR25], we call the NIOTE *half-succinct* if the size of $\mathbf{d}$ is sublinear in m (the size of E may be linear), and *fully succinct* if both $|\mathbf{d}|$ and $|E|$ are sublinear in m .

Security. We formalize two security notions: hasher privacy and encoder privacy. These notions guarantee that the digest and the encoding are simulatable, respectively, without knowing the input vector.

Definition 3 (Hasher Privacy). Let $\mathcal{G}_{\mathcal{A}, \text{Sim}}^{\text{Hpriv}}$ be the security game as defined in Figure 2 (left). We say that a NIOTE scheme $(\text{Setup}, \text{En}_A, \text{En}_B, \text{De}_A, \text{De}_B)$ is hasher-private, if there exists a PPT simulator Sim , such that for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, we have

$$\left| \frac{1}{2} - \Pr \left[\mathcal{G}_{\mathcal{A}, \text{Sim}}^{\text{Hpriv}}(1^\lambda) = 1 \right] \right| \leq \text{negl}(\lambda),$$

$G_{\mathcal{A}, \text{Sim}}^{\text{Hpriv}}$	$G_{\mathcal{A}, \text{Sim}}^{\text{Epriv}}$
1 : $\text{pk} \leftarrow \text{Setup}(1^\lambda)$	1 : $\text{pk} \leftarrow \text{Setup}(1^\lambda)$
2 : $(\mathbf{x}, \text{aux}) \leftarrow \mathcal{A}_0(1^\lambda, \text{pk})$	2 : $(\mathbf{y}, \text{aux}) \leftarrow \mathcal{A}_0(1^\lambda, \text{pk})$
3 : $b \leftarrow \mathbb{S}\{0, 1\}$	3 : $b \leftarrow \mathbb{S}\{0, 1\}$
4 : if $b = 0$: $\mathbf{d}_0 \leftarrow \text{En}_A(\text{pk}, \mathbf{x})$	4 : if $b = 0$: $E_0 \leftarrow \text{En}_B(\text{pk}, \mathbf{y})$
5 : else : $\mathbf{d}_1 \leftarrow \text{Sim}(1^\lambda, \text{pk})$	5 : else : $E_1 \leftarrow \text{Sim}(1^\lambda, \text{pk})$
6 : $b' \leftarrow \mathcal{A}_1(\mathbf{d}_b, \text{aux})$	6 : $b' \leftarrow \mathcal{A}_1(E_b, \text{aux})$
7 : return 1 iff. $b = b'$	7 : return 1 iff. $b = b'$

Figure 2: Security games for hasher privacy $G_{\mathcal{A}, \text{Sim}}^{\text{Hpriv}}$ (left) and encoder privacy $G_{\mathcal{A}, \text{Sim}}^{\text{Epriv}}$ (right).

for every $\lambda \in \mathbb{N}$, where the probability is taken over the randomness in Setup , En_A and the random coins of Sim .

Definition 4 (Encoder Privacy [AMR25]). Let $G_{\mathcal{A}, \text{Sim}}^{\text{Epriv}}$ be the security game as defined in Figure 2 (right). We say that a NIOTE scheme $(\text{Setup}, \text{En}_A, \text{En}_B, \text{De}_A, \text{De}_B)$ is encoder-private if there exists a PPT simulator Sim , such that for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, we have

$$\left| \frac{1}{2} - \Pr \left[G_{\mathcal{A}, \text{Sim}}^{\text{Epriv}}(1^\lambda) = 1 \right] \right| \leq \text{negl}(\lambda),$$

for every $\lambda \in \mathbb{N}$, where the probability is taken over the randomness in Setup , En_B and the random coins of Sim .

2.3 Learning with Errors

We use the same notation used in [AMR25]. In particular, we say that a distribution $\chi(1^\lambda)$ over $\mathbb{Z}$ is $B(\lambda)$ -bounded, if

$$\Pr[|e| \leq B(\lambda) \mid e \leftarrow \chi(1^\lambda)] = 1.$$

When we write $\mathbf{v} \leftarrow \chi(1^\lambda)$ and $\mathbf{v}$ is a vector, we mean that each entry of $\mathbf{v}$ is sampled from $\chi(1^\lambda)$ independently of all the others. The security of our protocols is based on the learning with error assumption [Reg05].

Definition 5 (Learning with Errors [Reg05]). Let $k := k(\lambda)$, $m := m(\lambda)$ and $q := q(\lambda)$ be positive integers. Let $\chi(1^\lambda)$ be a low-norm distribution over $\mathbb{Z}$. We say that the (χ, k, m, q) -LWE problem is hard if, for every PPT adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\lambda)$ such that, for every $\lambda \in \mathbb{N}$, we have

$$\left| \Pr \left[\mathcal{A}(1^\lambda, \mathbf{M}, \mathbf{u}) = 1 \mid \begin{array}{l} \mathbf{M} \leftarrow \mathbb{Z}_q^{m \times k}, \mathbf{s} \leftarrow \mathbb{Z}_q^k \\ \mathbf{e} \leftarrow \chi^m(1^\lambda) \\ \mathbf{u} \leftarrow \mathbf{M} \cdot \mathbf{s} + \mathbf{e} \end{array} \right] - \Pr \left[\mathcal{A}(1^\lambda, \mathbf{M}, \mathbf{u}) = 1 \mid \begin{array}{l} \mathbf{M} \leftarrow \mathbb{Z}_q^{m \times k} \\ \mathbf{u} \leftarrow \mathbb{Z}_q^m \end{array} \right] \right| \leq \text{negl}(\lambda).$$

Suppose that $\chi(1^\lambda)$ is $B(\lambda)$ -bounded. We call the quantity $\alpha := q/B$ the *modulus-to-noise ratio*.

We also use the non-uniform LWE (NLWE) assumption, where the matrix $\mathbf{M}$ is no longer uniformly random over $\mathbb{Z}_q^{m \times k}$ but over $\mathbb{Z}_2^{m \times k}$. It is shown in [BLMR13] that non-uniform LWE is at least as hard as LWE, with a slightly increased parameter size.

Theorem 1 (Non-uniform LWE [BLMR13]). Assume the hardness of (χ, k, m, q) -LWE. Then, $(\chi, k \cdot \log_2 q, m, q)$ -NLWE is hard.

3 Multi-Party NIOTE

In this section, we recall how to augment the two-party fully succinct NIOTE of Abram et al. [AMR25] with hasher privacy and perfect correctness. Fully succinct refers to the sizes of digest and encoding(s), which are succinct in Alice’s input $\mathbb{Z}_p^m$. Once we have hasher privacy and correctness, we show a simple extension of the two-party construction into a multi-party fully succinct NIOTE. We use the latter as a building block for our multi-party DPF setup protocols.

3.1 A Fully Correct, Hasher-Private, Succinct NIOTE for 2 Parties

The original construction by [AMR25] already provides encoder privacy, i.e., the encodings do not reveal information about Bob’s input $\mathbf{y}$. However, the digest $\mathbf{d} = \mathbf{A}\hat{\mathbf{x}}$ reveals information about Alice’s input $\mathbf{x}$, where $\hat{\mathbf{x}}$ is the bit decomposition of $\mathbf{x}$.

The Bootstrapping Construction. The construction of [AMR25] bootstraps a half-succinct NIOTE, where the digest is succinct but the encodings are not, into a fully succinct scheme. Recall that in a NIOTE instance the encoder outputs $(E, \phi) \leftarrow \text{En}_B(\cdot)$, where ϕ is the encoder’s secret state that is later used by De_B .

The key observation is that the encoder’s decoding $\text{De}_B(\mathbf{d}, \phi)$ can be evaluated *without revealing the digest $\mathbf{d}$* , as long as the parties hold additive shares of $\mathbf{d} \otimes \phi$. The authors call this the bilinear encoder evaluation property. Since shares of $\mathbf{d} \otimes \phi$ can themselves be generated as the output of another NIOTE instance with smaller inputs, this idea can be applied recursively until only a single digest needs to be sent.

More concretely, let r be the number of levels in a t -ary tree. The hasher splits its long input $\mathbf{x}$ into chunks of constant size, say n , which form the leaves of the tree. At the leaf level, the hasher groups the leaves into blocks of t chunks and hashes each such block into a digest $\mathbf{d}_{r-1}^i$. In parallel, the encoder computes $(E_{r-1}, \phi_{r-1}) \leftarrow \text{En}_B(\mathbf{y})$ and keeps ϕ_{r-1} private.

Next, each internal node of the tree is computed similarly: every t digests from level $j + 1$ (e.g., $\mathbf{d}_{j+1}^{(i-1)t+1}, \dots, \mathbf{d}_{j+1}^{it}$) together with the encoder’s secret ϕ_{j+1} serve as input to a NIOTE for $(\mathbf{d}_{j+1}^{(i-1)t+1}, \dots, \mathbf{d}_{j+1}^{it}) \otimes \phi_{j+1}$. The hasher obtains the parent digest $\mathbf{d}_j^i$, while the encoder produces a single encoding and obtains ϕ_j , i.e., $(E_j, \phi_j) \leftarrow \text{En}_B(\phi_{j+1})$. This process repeats level-by-level until a single digest $\mathbf{d}_0$ and a single encoder secret ϕ_0 remain. Finally, the digest $\mathbf{d}_0$ is sent to the encoder, and all encodings $E_{r-1}, \dots, E_0$ are sent to the hasher.

Decoding starts at the root, where the parties obtain additive shares of $\mathbf{d}_0 \otimes \phi_0$. Using bilinear encoder evaluation, the parties can combine their shares of $\mathbf{d}_j \otimes \phi_j$ with the encodings to derive shares of $\mathbf{d}_{j+1} \otimes \phi_{j+1}$ at the next level. At the leaf level, the parties use their shares of $\mathbf{d}_{r-1} \otimes \phi_{r-1}$ to obtain shares of $\mathbf{x}^i \otimes \mathbf{y}$, which can be reshaped into the desired tensor evaluation. Figure 3 visualizes the bootstrapping for $t = 2$.

Achieving Hasher Privacy. We follow the observation by [AMR26]: Only the root digest $\mathbf{d}_0$ is sent to the encoder, and the input size at the root is $tn \times n$, i.e., independent of m or ℓ . Therefore, one can replace the half-succinct, not hasher-private NIOTE with a hasher-private, not necessarily succinct NIOTE protocol, e.g., [ARS24], only for the final digest computation at the root. Since the input sizes at the root are independent of m , this change preserves succinctness in m .

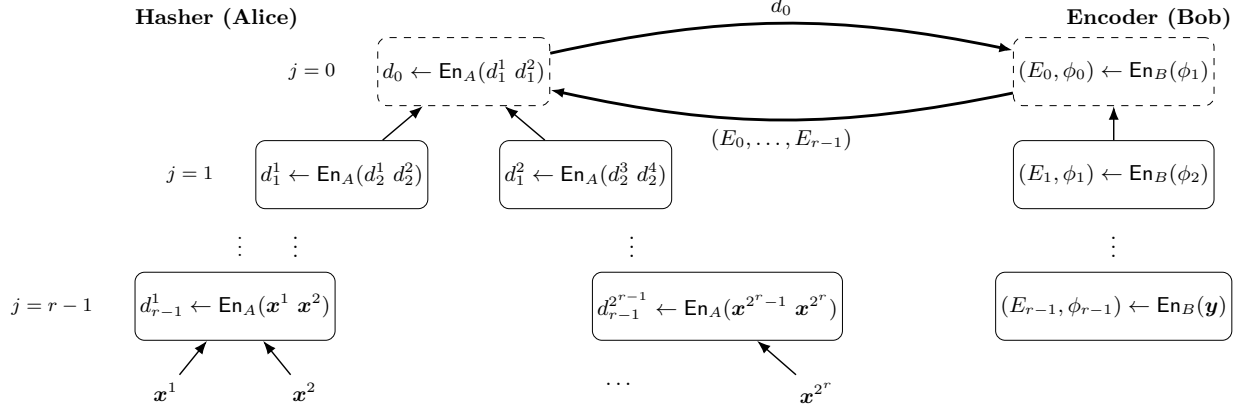


Figure 3: Bootstrapping of [AMR25] for $t = 2$. The parties only exchange d and $(E_0, \dots, E_{r-1})$. To achieve hasher privacy, we replace the root NIOTE (dashed box) by a non-succinct, hasher-private construction.

Achieving a Negligible Correctness Error. Let $\mathbf{v}_j$ and $\mathbf{w}_j$ be the intermediate outputs of both parties at level j . Correctness of the half-succinct NIOTE scheme guarantees that

$$\mathbf{v}_j + \mathbf{w}_j \mod q = \Delta \cdot (\mathbf{d}_{j-1} \otimes \phi_{j-1}) + \mathbf{e} \mod q,$$

where the error term $\mathbf{e}$ is bounded and $\Delta = q/p$ is the scaling term. Since using noisy $\mathbf{v}_j$ and $\mathbf{w}_j$ as input to the bilinear encoder evaluation to derive shares of the next tensor $\mathbf{d}_j \otimes \phi_j$ leads to noise growth, we use the standard trick [DHRW16] and locally round to obtain noise-free shares *over the plaintext modulus*. However, the bilinear encoder evaluation requires additive shares over the ciphertext modulus q . Following [BKS19], we therefore (non-interactively) lift the shares back to the ciphertext modulus, which requires $\|\mathbf{d}_{j-1} \otimes \phi_{j-1}\|_\infty \ll p$ to work with negligible correctness error. We adapt the procedure such that both parties use *bit-decomposed* inputs, i.e., $\hat{\mathbf{d}}_{j-1} := \text{Bits}(\mathbf{d}_{j-1})$ and $\hat{\phi}_{j-1} := \text{Bits}(\phi_{j-1})$, during hashing/encoding, and also adapt the bilinear encoder evaluation to work over shares of $\hat{\mathbf{d}}_{j-1} \otimes \hat{\phi}_{j-1}$. With appropriate choices for p and q , the failure probabilities for rounding and lifting can be made negligible in λ . Thus we have,

$$\begin{array}{ll} \text{(Hasher)} & \text{(Encoder)} \\ \tilde{\mathbf{v}}_j := \lceil \mathbf{v}_j \rceil_p, & \tilde{\mathbf{w}}_j := \lceil \mathbf{w}_j \rceil_p, \\ \hat{\mathbf{v}}_j := \text{Lift}_K^0(\tilde{\mathbf{v}}_j), & \hat{\mathbf{w}}_j := \text{Lift}_K^1(\tilde{\mathbf{w}}_j), \end{array}$$

where $(\tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ form additive shares mod p , and $(\hat{\mathbf{v}}, \hat{\mathbf{w}})$ are shares mod q of the same value. Since the lifting correctness requires re-randomized shares, the parties use a PRF with key K , which is also stored in the public key pk of the succinct NIOTE.

For completeness, we reproduce the half-succinct NIOTE from [AMR25] in Figure 12 and the bootstrapping with hasher privacy in Figures 13 and 14 in Appendix A. The proposition below follows from the correctness and security proof in [AMR25] and the correctness and security of the hasher-private OTE⁶ at the root of the tree. A complete proof of correctness can be found in Appendix A.

⁶ We detail the bootstrapping with any hasher-private, non-succinct OTE at the root while [AMR26] give a concrete protocol based on [ARS24].

Multi-Party NIOTE

PARAMETERS: Number of parties N , plaintext modulus p , succinct NIOTE = (Setup, En_A, En_B, De_A, De_B) for $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^\ell$.

PARTY INPUT: $\mathbf{x}^i \in \mathbb{Z}_p^m$ and $\mathbf{y}^i \in \mathbb{Z}_p^\ell$.

OUTPUT: $\mathbf{v}^i$, where $\mathbf{v}^1 + \dots + \mathbf{v}^N = (\mathbf{x}^1 + \dots + \mathbf{x}^N) \otimes (\mathbf{y}^1 + \dots + \mathbf{y}^N)$.

The construction comprises three procedures (Setup, En, De), defined as follows:

- Setup(1^λ). Output $\mathbf{pk} \leftarrow \text{NIOTE.Setup}(1^\lambda)$.
- En($\mathbf{pk}, \mathbf{x}^i, \mathbf{y}^i$). Compute $(\mathbf{d}^i, \psi^i) \leftarrow \text{NIOTE.En}_A(\mathbf{pk}, \mathbf{x}^i)$ and $(E^i, \phi^i) \leftarrow \text{NIOTE.En}_B(\mathbf{pk}, \mathbf{y}^i)$. Output message $M^i := (\mathbf{d}^i, E^i)$ and secret state $\mathbf{v}^i := (\mathbf{x}^i, \mathbf{y}^i, \psi^i, \phi^i)$.
- De($\mathbf{pk}, \{\mathbf{d}^j, E^j\}_{j \neq i}, \mathbf{v}^i$). Parse $\mathbf{v}^i$ as $\mathbf{x}^i, \mathbf{y}^i, \psi^i, \phi^i$, then compute and output

$$\mathbf{v}^i := \mathbf{x}^i \otimes \mathbf{y}^i + \sum_{j \neq i} (\text{NIOTE.De}_A(E^j, \psi^i) + \text{NIOTE.De}_B(\mathbf{d}^j, \phi^i)) \in \mathbb{Z}_p^{m\ell}.$$

Figure 4: Multi-Party NIOTE.

Proposition 1. *The fully succinct protocol described in Figures 13 and 14 is hasher-private, encoder-private, fully correct and succinct in m .*

On the Efficiency of NIOTE. Bootstrapping a half-succinct NIOTE over a t -ary tree of depth $r = \lceil \log_t(m/n) \rceil$ yields a scheme succinct in the left length m and linear in ℓ . Concretely, in the AMR25 bootstrapping Alice sends a single digest $\mathbf{d} \in \mathbb{Z}_q^n$, while Bob sends r encodings. The online transcript is dominated by Bob and scales as $\tilde{O}(r \cdot t^2 \cdot n^3)$ $\mathbb{Z}_q$ -elements for a single right input, and becomes $\tilde{O}(\ell \cdot r \cdot t^2 \cdot n^2)$ $\mathbb{Z}_q$ -elements after splitting $\mathbf{y}$ into n -sized blocks. The main constant is the digest dimension n , which in LWE/GSW-style instantiations grows linearly with the gadget length, e.g., $n = k \cdot \lceil \log_b q \rceil$ for a base- b gadget. Hence moving to a larger-base gadget can reduce communication by a polynomial factor in $\log b$, at the cost of larger noise terms that must be considered in the q/p slack. Orthogonally, a ring/module-LWE instantiation can pack S plaintext slots per ring element, effectively replacing m by m/S in the tree depth r , which reduces the number of bootstrapping levels and, in the perfectly-correct variant from Appendix A, also reduces the number of lifted coordinates. Overall, these optimizations suggest that the transcript size can drop by multiple orders of magnitude compared to an unpacked, binary-gadget baseline. A concrete end-to-end ring implementation is left to future work.

3.2 Multi-Party NIOTE

Consider a multi-party version of NIOTE for N parties, $\mathcal{P}_1, \dots, \mathcal{P}_N$, where each party $\mathcal{P}_i$ holds vectors $\mathbf{x}^i \in \mathbb{Z}_p^m$ and $\mathbf{y}^i \in \mathbb{Z}_p^\ell$ as input (or alternatively $\mathcal{P}_i$ holds additive shares of $\mathbf{x}$ and $\mathbf{y}$). Then, we want the parties to output $\mathbf{v}^i \in \mathbb{Z}_p^{m\ell}$ after a single, simultaneous broadcast message to all other parties, where

$$\mathbf{v}^1 + \dots + \mathbf{v}^N = \left(\sum_{i=1}^N \mathbf{x}^i \right) \otimes \left(\sum_{i=1}^N \mathbf{y}^i \right).$$

Due to the linearity of the tensor product and the two-party NIOTE, this can be achieved by running the two-party NIOTE protocol pairwise among the parties. Intuitively, the inputs of each party are kept private due to hasher and encoder privacy of the two-party scheme. We formalize

the multi-party construction in Figure 4. Note that the parties now have symmetric roles, thus En computes the message of a party to be broadcast and De decodes all received messages to output $\mathbf{v}^i$. The construction is succinct if the underlying two-party NIOTE is succinct.

Correctness. Observe that we can express

$$\left(\sum_{i=1}^N \mathbf{x}^i\right) \otimes \left(\sum_{i=1}^N \mathbf{y}^i\right) = \sum_{i=1}^N \left(\mathbf{x}^i \otimes \mathbf{y}^i + \sum_{j=1; j \neq i}^N \mathbf{x}^i \otimes \mathbf{y}^j\right),$$

by the linearity of the tensor product. Each pairwise tensor product $\mathbf{x}^i \otimes \mathbf{y}^j$ may be computed by a NIOTE between $\mathcal{P}_i$ and $\mathcal{P}_j$. Each $\mathcal{P}_i$ therefore computes $\mathbf{d}^i, \psi^i := \text{En}_A(\mathbf{x}^i)$ and $E^i, \phi^i := \text{En}_B(\mathbf{y}^i)$, and broadcasts $(\mathbf{d}^i, E^i)$. Then, $\mathcal{P}_i$ needs to compute $\mathbf{v}^i := \mathbf{x}^i \otimes \mathbf{y}^i + \sum_{j=1; j \neq i}^N (\text{De}_A(E^j, \psi^i) + \text{De}_B(\mathbf{d}^j, \phi^i))$, where $\text{De}_A(E^j, \psi^i)$ yields a share of $\mathbf{x}^i \otimes \mathbf{y}^j$ and $\text{De}_B(\mathbf{d}^j, \phi^i)$ yields a share of $\mathbf{x}^j \otimes \mathbf{y}^i$.

Remark 1. In our instantiation of NIOTE, correctness of the local rounding step in De_A and De_B is unfortunately limited to the two-party case, see, e.g., [DHRW16], which limits the multi-party construction to make pairwise use of the two-party NIOTE.

Our multi-party NIOTE therefore realizes the OTE functionality $\mathcal{F}_{\text{OTE}}$ in Figure 5 for N parties.

Proposition 2. *The N -party NIOTE protocol in Figure 4 securely realizes the N -party OTE functionality $\mathcal{F}_{\text{OTE}}$ (Figure 5).*

Proof. We construct a simulator $\mathcal{S}$ that given inputs of the corrupted parties $\mathbf{x}^j, \mathbf{y}^j$, for $j \in \mathcal{C}$, proceeds as follows. Let pk denote the public key, then the simulator computes $(M^j, \nu^j) \leftarrow \text{NIOTE.En}(\text{pk}, \mathbf{x}^j, \mathbf{y}^j)$. Further, let $\text{Sim}_{\text{HPriv}}$ denote the hasher-privacy simulator, and $\text{Sim}_{\text{EPriv}}$ denote the encoder-privacy simulator. $\mathcal{S}$ computes simulated messages M^i of the honest parties $i \notin \mathcal{C}$ as $M^i = (\mathbf{d}^i, E^i) \leftarrow (\text{Sim}_{\text{HPriv}}(\text{pk}), \text{Sim}_{\text{EPriv}}(\text{pk}))$ and then computes the expected output of the corrupted parties $\mathbf{v}^j := \text{NIOTE.De}(\text{pk}, \{M^k\}_{k \neq j}, \nu^j)$ using honestly computed M^j for corrupted parties, and simulated M^i for honest parties. Finally, $\mathcal{S}$ sends $(\mathbf{x}^j, \mathbf{y}^j, \mathbf{v}^j)$, for $j \in \mathcal{C}$ to $\mathcal{F}_{\text{OTE}}$.

Correctness follows from the construction and $\mathcal{F}_{\text{OTE}}$. For indistinguishability to the real protocol, consider the following sequence of distributions. Let c denote the number of corrupted parties. The first distribution is the output of the simulator $\mathcal{S}$. For each $i \in [N - c]$, in the i -th distribution, we replace the simulated $\mathbf{d}^i$ with a $\mathbf{d}^i$ obtained from $\text{En}_A(\mathbf{x}^i)$, i.e., correctly computed from honest party $\mathcal{P}_i$'s input. Similarly, for each $i \in [N - c]$, in the $(N - c + i)$ -th distribution, we replace the simulated encoding E^i with a correctly computed one. Now, the final distribution is identical to the real protocol execution. Each step between a distribution only differs in the changed digest or encoding, which is indistinguishable by hasher and encoder privacy. $\square$

4 Our OTE-Based Distributed Point Function

In this section, we present our OTE-based N -party DPF framework.

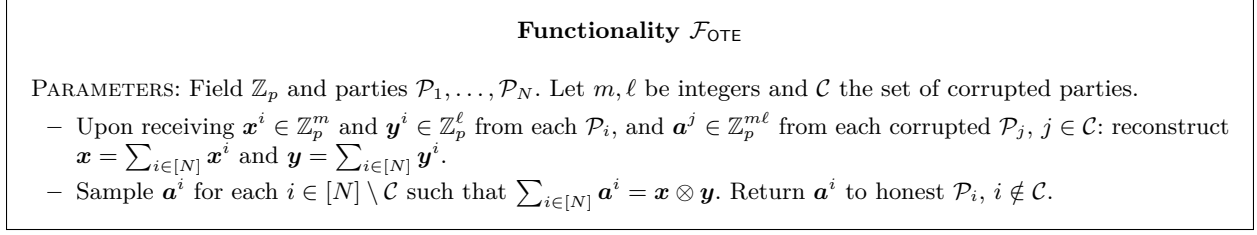


Figure 5: Ideal functionality for OTE.

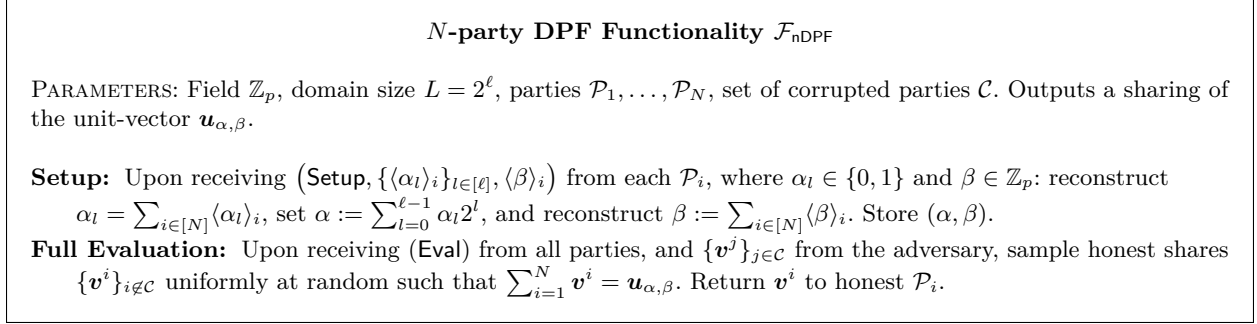


Figure 6: Ideal functionality for an N -party DPF with explicit setup and evaluation steps.

Overview. At a high level, we first design an N -party MPC protocol Π_{nDPF} that realizes the ideal functionality $\mathcal{F}_{\text{nDPF}}$ (Figure 6) in the $\mathcal{F}_{\text{OTE}}$ -hybrid model, and then compile Π_{nDPF} into a distributed point function as in Definition 1. The compilation is straightforward and the security of the resulting scheme is directly derived from the simulation-based security of Π_{nDPF} .

Later (in Section 5), we will instantiate Π_{nDPF} with concrete protocols based on the OTE primitive. These instantiations differ in how they arrange the necessary OTE invocations, and therefore exhibit different trade-offs between the number of setup rounds and the total communication. In particular, let $L = 2^\ell$ be the domain size of the DPF, then implementing the OTE primitive with a succinct NIOTE, we obtain one instantiation with $\mathcal{O}(\log L)$ setup rounds and total communication $\mathcal{O}(N \text{poly}(\lambda) \log^2 L)$, as well as alternative instantiations with fewer setup rounds and corresponding changes in communication overhead.

From Π_{nDPF} to N -party DPF. Given an N -party MPC protocol Π_{nDPF} that securely realizes $\mathcal{F}_{\text{nDPF}}$ against a semi-honest adversary corrupting up to $N - 1$ parties, the compilation to an N -party DPF is described in Figure 7. At a high level, the setup algorithm emulates an execution of Π_{nDPF} on additive shares of (α, β) and stores, for each party, its local shares, its private randomness, and the messages it would receive during the emulation. This recorded local view is the party’s DPF key. To evaluate, the party locally replays Π_{nDPF} using its key and outputs its final vector share.

Security follows directly from simulation-based security of Π_{nDPF} . Indeed, for any coalition $\mathcal{C}$ of size at most $N - 1$, the joint distribution of keys $\{\mathbf{k}_i\}_{i \in \mathcal{C}}$ is exactly the joint view of the corrupted parties in the emulated execution of Π_{nDPF} . Thus, distinguishing between keys for two choices (α_0, β_0) and (α_1, β_1) implies distinguishing a real joint view from a simulated one, contradicting that Π_{nDPF} securely realizes $\mathcal{F}_{\text{nDPF}}$.

Proposition 3. *Let Π_{nDPF} be an N -party protocol that securely realizes $\mathcal{F}_{\text{nDPF}}$ against a semi-honest adversary corrupting up to $N - 1$ parties. Let $\text{DPF}[\Pi]$ be the N -party DPF resulting from the*

DPF[Π]: $\Pi_{\text{nDPF}} \Rightarrow N\text{-party DPF}$

Let Π be an N -party protocol that realizes $\mathcal{F}_{\text{nDPF}}$ and whose communication is a public transcript tr . Define $\text{DPF}[\Pi] = (\text{Gen}, \text{FullEval})$:

Gen($1^\lambda, \alpha, \beta$):

1. Let $(\alpha_0, \dots, \alpha_{\ell-1}) = \text{Bits}(\alpha)$ and create additive shares $\{\langle \alpha_l \rangle_i\}_{i,l}$ and $\{\langle \beta \rangle_i\}_i$ over $\mathbb{Z}_p$.
2. Sample a global seed $\mathbf{s} \leftarrow \{0, 1\}^\lambda$ and per-party seeds $\mathbf{r}^1, \dots, \mathbf{r}^N \leftarrow \{0, 1\}^\lambda$.
3. Emulate an execution of Π where party $\mathcal{P}_i$ uses input $(\{\langle \alpha_l \rangle_i\}_{l \in [\ell]}, \langle \beta \rangle_i)$ and randomness derived from $\mathbf{r}^i$; all public coins are derived from $\mathbf{s}$. Let tr denote the resulting transcript.
4. Output keys $\mathbf{k}_i = (\mathbf{s}, \mathbf{r}^i, \{\langle \alpha_l \rangle_i\}_{l \in [\ell]}, \langle \beta \rangle_i, \text{tr})$ for all i .

FullEval($\mathbf{k}_i$): Given $\mathbf{k}_i$, locally re-run party $\mathcal{P}_i$'s algorithm of Π using public coins from $\mathbf{s}$, private coins from $\mathbf{r}^i$, and using tr as the sequence of received messages. Output the resulting $\mathbf{v}^i \in \mathbb{Z}_p^L$.

Figure 7: Compiler producing an N -party DPF from any protocol realizing $\mathcal{F}_{\text{nDPF}}$.

compiler in Figure 7. Then $\text{DPF}[\Pi]$ satisfies the standard $(N - 1)$ -privacy game for N -party DPFs: for any coalition $\mathcal{C} \subseteq [N]$ with $|\mathcal{C}| \leq N - 1$, the joint distribution of keys $\{\mathbf{k}_i\}_{i \in \mathcal{C}}$ is computationally indistinguishable for any two choices of (α, β) .

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ and a coalition $\mathcal{C}$ with $|\mathcal{C}| \leq N - 1$ that wins the N -party DPF privacy game for $\text{DPF}[\Pi]$ with non-negligible advantage. We build a distinguisher $\mathcal{D}$ against the security of Π . On input either (i) a real joint view of the corrupted parties in an execution of Π (real world) or (ii) a simulated joint view produced by the simulator for Π (ideal world), the distinguisher $\mathcal{D}$ constructs keys $\{\mathbf{k}_i\}_{i \in \mathcal{C}}$ exactly as prescribed by the compilation to $\text{DPF}[\Pi]$ (i.e., by packaging the corrupted parties' shares, randomness, and received messages) and forwards these keys to $\mathcal{A}$. Finally, $\mathcal{D}$ outputs whatever bit $\mathcal{A}$ outputs.

By construction, when $\mathcal{D}$ is given a real view, the resulting distribution of $\{\mathbf{k}_i\}_{i \in \mathcal{C}}$ is exactly the real key distribution in the DPF privacy game; when $\mathcal{D}$ is given a simulated view, it matches the simulated key distribution for the same coalition. Therefore, any non-negligible distinguishing advantage of $\mathcal{A}$ between real and simulated keys would imply a non-negligible distinguishing advantage of $\mathcal{D}$ between the real and ideal executions of Π , contradicting that Π securely realizes $\mathcal{F}_{\text{nDPF}}$. Hence, $\text{DPF}[\Pi]$ is $(N - 1)$ -private in the standard DPF privacy game. $\square$

5 Instantiating Π_{nDPF}

We now instantiate Π_{nDPF} , i.e., we give concrete N -party MPC protocols that realize the DPF functionality $\mathcal{F}_{\text{nDPF}}$ of Figure 6 in the $\mathcal{F}_{\text{OTE}}$ -hybrid model. As before, we fix $L = 2^\ell$ and assume that each party inputs additive shares over $\mathbb{Z}_p$ of the address bits $\alpha_0, \dots, \alpha_{\ell-1}$ and of $\beta \in \mathbb{Z}_p$, such that $\alpha = \sum_{l=0}^{\ell-1} \alpha_l 2^l$.

We give several instantiations of Π_{nDPF} . We first present our main protocol family Π_{DPF}^B , parameterized by a public block size B . Once we instantiate the functionality $\mathcal{F}_{\text{OTE}}$ (Figure 5) using the multi-party NIOTE primitive described in the previous section, this parameter B trades setup rounds for communication by ensuring that each NIOTE invocation has right-input length bounded by B . Later, in Section 5.4, we give an alternative construction to Π_{DPF}^B , called $\Pi_{\text{DPF}}^{\text{shuf}}$, which requires $\Theta(\log L)$ rounds and polylogarithmic communication. $\Pi_{\text{DPF}}^{\text{shuf}}$ has at least a factor-two lower communication cost than the main protocol family when setting $\log B = 1$ with the same number of setup rounds, and can be instantiated with any *non-interactive scalar-vector multiplication* scheme.

5.1 Unit Vector Tensor Representation

The core idea of our protocol is a tensor-tree technique that allows us to reduce the number of synchronous rounds and that is based on the following lemma and its block generalization.

Lemma 1. Fix a prime field $\mathbb{Z}_p$ and let $L = 2^\ell$. Let $\alpha \in [L]$ with binary decomposition $\alpha = \sum_{t=0}^{\ell-1} \alpha_t 2^t$, where each $\alpha_t \in \{0, 1\} \subseteq \mathbb{Z}_p$, and let $\beta \in \mathbb{Z}_p$. For each $t \in [\ell]$, define the length-2 selector vector $\mathbf{s}_t = (1 - \alpha_t, \alpha_t) \in \mathbb{Z}_p^2$. Define the ℓ -fold tensor product

$$\mathbf{u} = \mathbf{s}_{\ell-1} \otimes \mathbf{s}_{\ell-2} \otimes \cdots \otimes \mathbf{s}_0 \in \mathbb{Z}_p^{2^\ell}. \quad (2)$$

Then $\mathbf{u}$ is exactly the unit vector $\mathbf{u}_{\alpha,1}$, i.e., $\mathbf{u}[\alpha] = 1$ and $\mathbf{u}[j] = 0$ for all $j \neq \alpha$. Consequently, $\mathbf{u}_{\alpha,\beta} = \beta \cdot \mathbf{u} = \beta \cdot (\mathbf{s}_{\ell-1} \otimes \cdots \otimes \mathbf{s}_0)$.

Proof. We start by formalizing our indexing notation. For any $j \in [L] = \{0, \dots, 2^\ell - 1\}$, we can write its binary representation (LSB-first) as $j = \sum_{t=0}^{\ell-1} j_t 2^t$, where $j_t \in \{0, 1\}$. Equivalently, $j = 2j' + j_0$ where $j' \in [2^{\ell-1}]$ has binary expansion $j' = \sum_{t=1}^{\ell-1} j_t 2^{t-1}$.

Let $\mathbf{a} \in \mathbb{Z}_p^m$ and $\mathbf{b} \in \mathbb{Z}_p^n$. By definition, $\mathbf{a} \otimes \mathbf{b} \in \mathbb{Z}_p^{mn}$ satisfies, for all $i \in [m]$ and $r \in [n]$, $(\mathbf{a} \otimes \mathbf{b})[i \cdot n + r] = \mathbf{a}[i] \cdot \mathbf{b}[r]$. Equivalently, for any $j \in [mn]$, writing $i = \lfloor j/n \rfloor$ and $r = j \bmod n$, we have $(\mathbf{a} \otimes \mathbf{b})[j] = \mathbf{a}[i] \cdot \mathbf{b}[r]$.

Define $\mathbf{u} := \mathbf{s}_{\ell-1} \otimes \mathbf{s}_{\ell-2} \otimes \cdots \otimes \mathbf{s}_0 \in \mathbb{Z}_p^{2^\ell}$. We claim that for every $j \in [L]$ with bit representation $(j_0, \dots, j_{\ell-1})$,

$$\mathbf{u}[j] = \prod_{t=0}^{\ell-1} \mathbf{s}_t[j_t]. \quad (3)$$

To see this, write $\mathbf{u} = \mathbf{u}' \otimes \mathbf{s}_0$ where $\mathbf{u}' := \mathbf{s}_{\ell-1} \otimes \mathbf{s}_{\ell-2} \otimes \cdots \otimes \mathbf{s}_1 \in \mathbb{Z}_p^{2^{\ell-1}}$. For any $j \in [2^\ell]$, write $j = 2j' + j_0$. Since the right tensor factor has length $n = 2$, the tensor indexing rule gives

$$\mathbf{u}[j] = (\mathbf{u}' \otimes \mathbf{s}_0)[2j' + j_0] = \mathbf{u}'[j'] \cdot \mathbf{s}_0[j_0].$$

Applying the same argument recursively to $\mathbf{u}'$ (equivalently, by induction on ℓ) yields (3).

Now evaluate each factor. Recall $\mathbf{s}_t = (1 - \alpha_t, \alpha_t)$ with $\alpha_t \in \{0, 1\}$. Hence for any $j_t \in \{0, 1\}$,

$$\mathbf{s}_t[j_t] = \begin{cases} 1 & \text{if } j_t = \alpha_t, \\ 0 & \text{if } j_t \neq \alpha_t. \end{cases} \quad (4)$$

Fix any $j \in [L]$. Combining (3) and (4), we get that $\mathbf{u}[j] = 1$ iff $j_t = \alpha_t$ for all t , i.e., iff $j = \alpha$. Otherwise at least one factor is 0 and thus $\mathbf{u}[j] = 0$. Therefore $\mathbf{u} = \mathbf{u}_{\alpha,1}$. Finally, $\mathbf{u}_{\alpha,\beta} := \beta \cdot \mathbf{u}_{\alpha,1}$, so $\beta \mathbf{u} = \mathbf{u}_{\alpha,\beta}$. $\square$

Remark 2. We stress here that under the standard tensor indexing $(\mathbf{a} \otimes \mathbf{b})[i \cdot n + r] = \mathbf{a}[i] \cdot \mathbf{b}[r]$, the rightmost tensor factor corresponds to the least-significant element in base n . Thus, with LSB-first binary expansions $j = \sum_{t=0}^{\ell-1} j_t 2^t$, the unit-vector identity is naturally written as $\mathbf{u} = \mathbf{s}_{\ell-1} \otimes \cdots \otimes \mathbf{s}_0$. If instead we define $\tilde{\mathbf{u}} := \mathbf{s}_0 \otimes \cdots \otimes \mathbf{s}_{\ell-1}$, then $\tilde{\mathbf{u}}$ is the unit vector at the bit-reversal of α .

Example 1. Let $\ell = 3$, so $L = 8$, p a prime and $\alpha = 5$ with binary representation $(\alpha_0, \alpha_1, \alpha_2) = (1, 0, 1)$. We can define the selector vector $\mathbf{s}_0 = (0, 1)$, $\mathbf{s}_1 = (1, 0)$ and $\mathbf{s}_2 = (0, 1)$. Lemma 1 defines

$$\mathbf{u} = \mathbf{s}_2 \otimes \mathbf{s}_1 \otimes \mathbf{s}_0 = (0, 0, 1, 0) \otimes (0, 1) = (0, 0, 0, 0, 0, 1, 0, 0).$$

We can generalize the previous lemma to block selector vectors. As before, we assume $L = 2^\ell$. We then choose a block size $b \geq 1$ and define the corresponding $B = 2^b$. We partition the ℓ address bits into consecutive blocks of size b , except possibly the last block, which may be shorter. In this way, we can write $\ell = kb + b_{\text{last}}$, with $k := \lfloor \ell/b \rfloor$ and $b_{\text{last}} \in \{0, \dots, b-1\}$. We also define the number of blocks as

$$K := \begin{cases} k & \text{if } b_{\text{last}} = 0, \\ k + 1 & \text{if } b_{\text{last}} > 0, \end{cases}$$

and their block bit-lengths by setting $b_j := b$ for $j \in \{0, \dots, k-1\}$ and $b_{K-1} := b_{\text{last}}$ if $b_{\text{last}} > 0$. Let $\alpha \in [L]$ have binary expansion $\alpha = \sum_{t=0}^{\ell-1} \alpha_t 2^t$. For each block $j \in \{0, \dots, K-1\}$, we define $\alpha^{(j)} \in [2^{b_j}]$ by grouping together the bits in that block, i.e., $\alpha^{(j)} := \sum_{h=0}^{b_j-1} \alpha_{jb+h} 2^h$, $j = 0, \dots, K-1$. Equivalently, α admits the representation $\alpha = \sum_{j=0}^{K-1} \alpha^{(j)} \cdot 2^{jb} = \sum_{j=0}^{K-1} \alpha^{(j)} \cdot B^j$, where the last block may be shorter. For each block j , we define the length- 2^{b_j} selector vector $\mathbf{s}^{(j)} \in \mathbb{Z}_p^{2^{b_j}}$ as the standard basis vector at position $\alpha^{(j)}$:

$$\mathbf{s}^{(j)}[\mathbf{u}] := \mathbf{1}[\mathbf{u} = \alpha^{(j)}] \quad \text{for } \mathbf{u} \in [2^{b_j}],$$

where $\mathbf{1}[\cdot] \in \{0, 1\} \subseteq \mathbb{Z}_p$.

We can now prove the following lemma.

Lemma 2. *Let $L = 2^\ell$ and let $\alpha \in [L]$ with block representation $\alpha^{(0)}, \dots, \alpha^{(K-1)}$ as above. We define the K -fold tensor product*

$$\mathbf{u} = \mathbf{s}^{(K-1)} \otimes \mathbf{s}^{(K-2)} \otimes \dots \otimes \mathbf{s}^{(0)} \in \mathbb{Z}_p^{\prod_{j=0}^{K-1} 2^{b_j}} = \mathbb{Z}_p^{2^\ell} = \mathbb{Z}_p^L.$$

Then $\mathbf{u}$ is exactly the unit vector $\mathbf{u}_{\alpha,1}$. Consequently, for any $\beta \in \mathbb{Z}_p$ we have $\mathbf{u}_{\alpha,\beta} = \beta \cdot \mathbf{u}$.

Proof. We use the standard tensor indexing rule: for $a \in \mathbb{Z}_p^m$ and $b \in \mathbb{Z}_p^n$,

$$(a \otimes b)[i \cdot n + r] = a[i] \cdot b[r], \quad (i \in [m], r \in [n]).$$

Let $x \in [L]$ and write its mixed-radix expansion $x = \sum_{j=0}^{K-1} x^{(j)} \cdot 2^{jb}$ where each $x^{(j)} \in [2^{b_j}]$ is the block induced by the bits $x_{jb}, \dots, x_{jb+b_j-1}$. By iterating the tensor indexing rule, the coordinate of the K -fold tensor product satisfies

$$u[x] = \prod_{j=0}^{K-1} s^{(j)}[x^{(j)}].$$

By definition of $s^{(j)}$ as a unit vector at $\alpha^{(j)}$, each factor is 1 iff $x^{(j)} = \alpha^{(j)}$ and 0 otherwise. Thus $u[x] = 1$ iff $x^{(j)} = \alpha^{(j)}$ for all blocks j , i.e., iff $x = \alpha$; otherwise $u[x] = 0$. Finally, scaling by β gives $\mathbf{u}_{\alpha,\beta} = \beta \cdot \mathbf{u}$. $\square$

5.2 Our Multi-Party DPF Protocol

Lemma 1 and its block version, Lemma 2, reduce the setup of our N -party DPF to evaluating an iterated tensor product of selector vectors. To securely tensor secret-shared vectors, the protocol makes black-box use of OTE (modeled as $\mathcal{F}_{\text{OTE}}$, Figure 5), which on input additive shares of $(\mathbf{x}, \mathbf{y})$

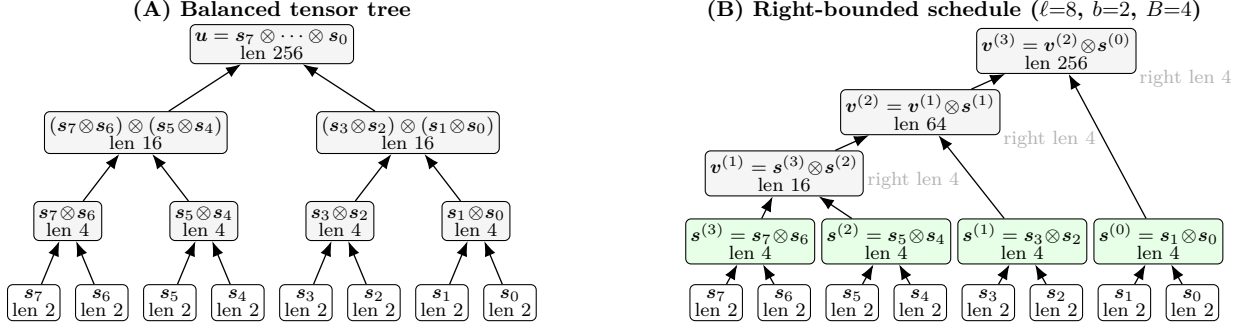


Figure 8: Small example for $\ell = 8$. On the left we have a balanced tensor tree and on the right a right-bounded schedule with $b = 2$ and $B = 4$.

returns additive shares of $\mathbf{x} \otimes \mathbf{y}$. Concretely, throughout this section we write each call to OTE by one execution of our N -party NIOTE = (Setup, En, De) from the previous section: each party runs En on its local shares, broadcasts once, and then runs De to obtain its output share. In this way, one tensoring layer, that is, a set of independent tensorings executed in parallel, costs exactly one synchronous broadcast round.

The core idea is that since $\otimes$ is associative, we are free to choose how to organize the ℓ -fold tensor product computation. In particular, this choice will determine (i) the number of synchronous rounds, and (ii) the input lengths in the underlying NIOTE calls.

Right-Bounded Tensoring and Polylogarithmic Key-Size. The protocol Π_{DPF}^B is fully described in Figure 9. First, it fixes a public block size $b \geq 1$ and let $B = 2^b$. Hence, it partitions the ℓ address bits into $K = \lceil \ell/b \rceil$ blocks and defines the corresponding block selectors $\mathbf{s}^{(j)} \in \mathbb{Z}_p^{2^{b_j}}$, as in Lemma 2. The protocol then proceeds in two phases: (i) it computes all block selectors $\mathbf{s}^{(j)}$ using balanced tensor trees of depth $R_{\text{block}} = \lceil \log_2 b \rceil$, where every NIOTE call tensors vectors of length at most B ; and (ii) it chains the blocks from most significant to least significant by computing $\mathbf{v}^{(r)} = \mathbf{v}^{(r-1)} \otimes \mathbf{s}^{(K-1-r)}$ for $r = 1, \dots, K-1$. The key property is that in every chaining step, the *right* input is a block selector of length at most B , while the left input may grow up to L . This parametrization contains two special corner cases. On the one hand, we have a fully balanced tensor tree by setting $b = \ell$. In fact, this implies $K = 1$, so Phase (ii) vanishes and Phase (i) is a single balanced tensor tree of depth $\lceil \log_2 \ell \rceil = \mathcal{O}(\log \log L)$ that directly outputs $\mathbf{u}_{\alpha,1} \in \mathbb{Z}_p^L$. At the other extreme, setting $b = 1$ gives $K = \ell$ and yields an $\mathcal{O}(\log L)$ -round pure chaining schedule. This behavior is illustrated in Figure 8, which compares the fully balanced tensor tree which achieves $\log \log L$ rounds with the right-bounded schedule that we use to obtain polylogarithmic key-size.

Theorem 2. *Let p be a prime and $L = 2^\ell$ the domain size. The protocol Π_{DPF}^B (Figure 9) securely realizes $\mathcal{F}_{\text{nDPF}}$ against a semi-honest adversary corrupting up to $N-1$ parties in the $\mathcal{F}_{\text{OTE}}$ -hybrid model.*

Proof. We begin with correctness in two steps: first for the balanced tensor tree sub-procedure, then for the full evaluation.

Correctness of BalancedTensorTree. First, by correctness of the N -party NIOTE, we have

$$\sum_i \mathbf{v}_{h+1,j}^i = \left(\sum_i \mathbf{v}_{h,2j}^i \right) \otimes \left(\sum_i \mathbf{v}_{h,2j+1}^i \right).$$

Let each binary leaf selector $\sum_i \mathbf{v}_{0,j}^i = (1 - \alpha_j, \alpha_j)$, for $\alpha_j \in \{0, 1\}$, then the protocol computes $\mathbf{v}_{0,T-1} \otimes \cdots \otimes \mathbf{v}_{0,0}$ in a balanced binary tree, by always tensoring nodes $2j$ and $2j + 1$. By Lemma 1,

$$\mathbf{v}_{0,T-1} \otimes \cdots \otimes \mathbf{v}_{0,0} = \mathbf{u}_{\sum_{j=0}^{T-1} \alpha_j 2^j, 1},$$

as desired.

Correctness of FullEval. By correctness of BalancedTensorTree, the shares of the block selector $\mathbf{s}^{(j)}$, for $j \in \{0, \dots, K-1\}$, are shares of the unit vector encoding $\alpha^{(j)} \in [2^{b_j}]$,

$$\sum_i \mathbf{s}^{(j)} = \mathbf{u}_{\alpha^{(j)}, 1}.$$

Note further, that for all $r \in \{1, 2, \dots, K-1\}$, by the correctness property of the N -party NIOTE,

$$\sum_i \mathbf{v}^{(r),i} = \left(\sum_i \mathbf{v}^{(r-1),i} \right) \otimes \left(\sum_i \mathbf{s}^{(K-1-r),i} \right).$$

Thus,

$$\sum_i \mathbf{v}^{(K-1),i} = \left(\left(\mathbf{s}^{(K-1)} \otimes \mathbf{s}^{(K-2)} \right) \otimes \mathbf{s}^{(K-3)} \right) \otimes \cdots \otimes \mathbf{s}^{(0)},$$

and by Lemma 2, $\sum_i \mathbf{v}^{(K-1),i} = \mathbf{u}_{\alpha, 1}$, where $\alpha = \sum_{j=0}^{K-1} \alpha^{(j)} 2^{b_j}$. Finally, the scaling by β indeed outputs $\sum_i \mathbf{u}^i = \mathbf{u}_{\alpha, \beta}$, again due to the correctness of NIOTE.

Security First, we can replace all calls to NIOTE with calls to $\mathcal{F}_{\text{OTE}}$ without changing the output of the procedure. Concretely,

- BalancedTensorTree: Each $\mathcal{P}_i$ sends $\mathbf{v}_{h,2j}^i$ and $\mathbf{v}_{h,2j+1}^i$ to $\mathcal{F}_{\text{OTE}}$, and receives a share of the tensor product, $\mathbf{v}_{h+1,j}^i$.
- FullEval: Each $\mathcal{P}_i$ sends $\mathbf{v}^{(r-1),i}$ and $\mathbf{s}^{(K-1-r),i}$ to $\mathcal{F}_{\text{OTE}}$ and receives a share of the tensor evaluation $\mathbf{v}^{(r),i}$. Further, each $\mathcal{P}_i$ sends δ^i and $\langle \beta \rangle_i$ to $\mathcal{F}_{\text{OTE}}$ and receives a share of the tensor evaluation $\mathbf{u}^i$.

Since now, all parties only interact with $\mathcal{F}_{\text{OTE}}$, security is immediate. $\square$

Remark 3 (Incorporating the value β with no extra costs). For simplicity, Figure 9 produces shares of $\mathbf{u}_{\alpha, 1}$ and then performs one final scalar-vector multiplication by β , implemented as a single NIOTE call with right input dimension 1. Alternatively, one can avoid this explicit final scaling step by injecting β into the tensoring computation itself: we can treat β as a length-1 vector $(\beta) \in \mathbb{Z}_p^1$ and compute a β -weighted block selector $\tilde{\mathbf{s}}^{(j)} := (\beta) \otimes \mathbf{s}^{(j)}$ for a fixed block j during Phase (i), and then chain using $\tilde{\mathbf{s}}^{(j)}$ in place of $\mathbf{s}^{(j)}$. By bilinearity of $\otimes$, the final output is $\mathbf{u}_{\alpha, \beta}$ and the explicit final scaling step can be removed.

Batched Broadcasts within a Tensoring Layer. At any tensoring layer h , the parties may need to evaluate q_h independent NIOTE instances in parallel. Rather than broadcasting q_h separate messages, party $\mathcal{P}_i$ runs En independently for each instance and concatenates the resulting messages into a single broadcast string

$$M_h^i := M_{h,0}^i \| M_{h,1}^i \| \cdots \| M_{h,q_h-1}^i,$$

while storing the corresponding local secrets $\nu_h^i := (\nu_{h,0}^i, \dots, \nu_{h,q_h-1}^i)$. Upon receiving M_h^k from other parties, $\mathcal{P}_i$ parses each M_h^k into its q_h components and applies De component-wise. This realizes exactly the same distribution as sending q_h separate broadcasts, but reduces the number of network messages to one broadcast per party per layer.

Table 3: Required rounds of communication and number of NIOTE calls for representative values of L and b . Highlighted values have the same number of rounds as the fully balanced tree (5).

L	NIOTE	$b = 5$	$b = 6$	$b = 7$	$b = 8$
2^{17}	16	6	5	5	5
2^{18}	17	6	5	5	5
2^{19}	18	6	6	5	5
2^{20}	19	6	6	5	5

L	NIOTE	$b = 5$	$b = 6$	$b = 7$	$b = 8$
2^{21}	20	7	6	5	5
2^{22}	21	7	6	6	5
2^{23}	22	7	6	6	5
2^{24}	23	7	6	6	5

5.3 Efficiency

Here we analyze the communication costs of our protocol. Phase (i) requires $R_{\text{block}} = \lceil \log_2 b \rceil$ synchronous rounds, independent of L , since all K block-trees are executed in parallel level-by-level. Phase (ii) requires $K - 1 = \lceil \ell/b \rceil - 1 = \log_B L$ synchronous rounds due to the sequential chaining. Including the single (optional) scalar-vector multiplication for β adds one further synchronous round. Therefore the total depth is $R_{\text{total}} = \lceil \log_2 b \rceil + (\lceil \ell/b \rceil - 1) + 1 = \Theta(\log b + \log_B L)$. As mentioned before, this interpolates between (i) the $\Theta(\log \log L)$ depth of the fully balanced tensor tree, obtained by taking $b = \ell$, and (ii) a pure left-fold chain, obtained by taking $b = 1$, which keeps the right dimension constant but offers no tree parallelism. Increasing b therefore reduces the number of setup rounds while increasing the communication cost exponentially.

In the right-bounded protocol, every Phase (ii) NIOTE invocation has right input length $m_1 \leq B$. Consequently, the communication per chaining step is bounded by $\mathcal{O}(B \log |\mathbf{v}^{(r-1)}|) \leq \mathcal{O}(B \log L)$, since $|\mathbf{v}^{(r-1)}| \leq L$ throughout. There are exactly $K - 1 = \mathcal{O}(\log_B L)$ such steps, hence the total Phase (ii) communication is

$$\mathcal{O}(B \log L \cdot \log_B L) = \mathcal{O}(B \log L \cdot \log L / \log B).$$

Finally, choosing $B = \text{poly}(\lambda)$ makes B and $\log B$ depend only on the security parameter, so the dependence on L is polylogarithmic. Phase (i) contributes an additional $\text{poly}(\lambda) \cdot \text{polylog}(L)$ term, since every NIOTE call in the block trees has both inputs of length at most B and there are only $\lceil \log_2 b \rceil$ synchronous levels. Table 3 lists the number of rounds and black-box NIOTE calls for relevant practical parameters. Our N -party NIOTE based on pairwise two-party NIOTE has an additional $\mathcal{O}(N)$ factor overhead (per party) for the communication cost.

5.4 An Alternative DPF Protocol with $\log L$ Rounds

We describe a DPF construction with asymptotic cost similar to Π_{nDPF}^B (Figure 9) with $b = 1$ but improved concrete efficiency. We will use the NIOTE as a non-interactive scalar-vector multiplication primitive by setting $\ell = 1$. The resulting primitive over $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^1$ has communication cost that is succinct in m .

To sketch the DPF construction, we write the special position $\alpha = \sum_{l=0}^{\ell-1} \alpha_l 2^l$ in binary with bits α_l . Our protocol outputs a secret-shared vector $\mathbf{v} := (\underbrace{0, \dots, 0}_{\alpha}, \beta, 0, \dots) \in \mathbb{Z}_p^{2^\ell}$, which corresponds to the full-domain evaluation of the DPF. Consider the j -th entry of $\mathbf{v}$, denoted $\mathbf{v}[j]$. It can be expressed as

$$\mathbf{v}[j] = \mathbf{u}_{0,\beta}[j \oplus 2^0 \alpha_0 \oplus \dots \oplus 2^{\ell-1} \alpha_{\ell-1}],$$

Multi-Party DPF Protocol Π_{DPF}^B

PARAMETERS: Prime field $\mathbb{Z}_p$, domain size $L = 2^\ell$, public block size $b \geq 1$ with $B = 2^b$, and a fully correct N -party NIOTE = (Setup, En, De) over $\mathbb{Z}_p$.

PARTY INPUT: Shares $\{\langle \alpha_t \rangle_i\}_{t \in [\ell]}$ of bits $\alpha_t \in \{0, 1\}$ and a share $\langle \beta \rangle_i \in \mathbb{Z}_p$.

OUTPUT: A vector $\mathbf{u}^i \in \mathbb{Z}_p^L$ such that $\sum_{i=1}^N \mathbf{u}^i = \mathbf{u}_{\alpha, \beta}$.

Setup: Parties run $\text{pk} \leftarrow \text{Setup}(1^\lambda)$ and get pk .

BalancedTensorTree: This is a sub-procedure of **FullEval**. On input T shares of binary leaf selectors $\mathbf{v}_{0,0}^i, \dots, \mathbf{v}_{0,T-1}^i$, where $\mathbf{v}_{0,j}^i \in \mathbb{Z}_p^2$.

1. Initialize the level-0 list: $V_0^i := (\mathbf{v}_{0,T-1}^i, \mathbf{v}_{0,T-2}^i, \dots, \mathbf{v}_{0,0}^i)$.
2. For levels $h = 0, 1, 2, \dots$, repeat until $|V_h^i| = 1$:
 - (a) Let $t_h := |V_h^i|$ (this value is the same for all parties).
 - (b) For each $j \in \{0, \dots, \lfloor t_h/2 \rfloor - 1\}$, in parallel:
 - i. Each $\mathcal{P}_i$ computes $(M_{h,j}^i, \nu_{h,j}^i) \leftarrow \text{En}(\text{pk}, \mathbf{v}_{h,2j}^i, \mathbf{v}_{h,2j+1}^i)$ and broadcasts $M_{h,j}^i$.
 - ii. Each $\mathcal{P}_i$ computes $\mathbf{v}_{h+1,j}^i := \text{De}(\text{pk}, \{M_{h,j}^k\}_{k \neq i}, \nu_{h,j}^i) \in \mathbb{Z}_p^{m_{h,2j} \cdot m_{h,2j+1}}$, where $m_{h,j}$ denotes the (public) length of $\mathbf{v}_{h,j}^i$.
 - (c) If t_h is odd, append the last vector unchanged: $\mathbf{v}_{h+1, \lfloor t_h/2 \rfloor}^i := \mathbf{v}_{h, t_h-1}^i$.
 - (d) Set $V_{h+1}^i := (\mathbf{v}_{h+1,0}^i, \dots, \mathbf{v}_{h+1, \lfloor t_h/2 \rfloor - 1}^i)$.
3. Let $\mathbf{w}^i$ denote the unique remaining vector in V_h^i , so $\mathbf{w}^i \in \mathbb{Z}_p^{2^T}$. Output $\mathbf{w}^i$.

FullEval: Parties proceed as follows.

1. PHASE (I): decompose into blocks and compute block selectors ($R_{\text{block}} = \lceil \log_2 b \rceil$ rounds).
 - (a) Let $K = \lceil \ell/b \rceil$ and block lengths b_j . For each block $j \in \{0, \dots, K-1\}$, let $\alpha^{(j)} \in [2^{b_j}]$ denote the j -th block digit and let $\mathbf{s}^{(j)} \in \mathbb{Z}_p^{2^{b_j}}$ be the corresponding selector vector.
 - (b) For each block j , the parties locally form additive shares of the b_j binary leaf selectors $\mathbf{s}_{jb+h} = (1 - \alpha_{jb+h}, \alpha_{jb+h}) \in \mathbb{Z}_p^2$, for $h = 0, \dots, b_j - 1$, and compute additive shares of the block selector $\mathbf{s}^{(j)} = \mathbf{s}_{jb+b_j-1} \otimes \dots \otimes \mathbf{s}_{jb} \in \mathbb{Z}_p^{2^{b_j}}$ using the **BalancedTensorTree** procedure restricted to the b_j leaves of block j . All blocks are processed in parallel level-by-level.
2. PHASE (II): chain blocks: Initialize $\mathbf{v}^{(0)} := \mathbf{s}^{(K-1)} \in \mathbb{Z}_p^{2^{b_{K-1}}}$. For $r = 1, 2, \dots, K-1$ do:
 - (a) Each party $\mathcal{P}_i$ computes $(M_r^i, \nu_r^i) \leftarrow \text{En}(\text{pk}, \mathbf{v}^{(r-1),i}, \mathbf{s}^{(K-1-r),i})$ and broadcasts M_r^i .
 - (b) Each party $\mathcal{P}_i$ computes $\mathbf{v}^{(r),i} := \text{De}(\text{pk}, \{M_r^k\}_{k \neq i}, \nu_r^i)$ where $\mathbf{v}^{(r),i} \in \mathbb{Z}_p^{|\mathbf{v}^{(r-1),i}| \cdot |\mathbf{s}^{(K-1-r),i}|}$. Let $\delta^i := \mathbf{v}^{(K-1),i} \in \mathbb{Z}_p^L$.
3. SCALE BY β . Each party $\mathcal{P}_i$ computes $(M_\beta^i, \nu_\beta^i) \leftarrow \text{En}(\text{pk}, \delta^i, (\langle \beta \rangle_i))$, broadcasts M_β^i , and outputs $\mathbf{u}^i := \text{De}(\text{pk}, \{M_\beta^k\}_{k \neq i}, \nu_\beta^i) \in \mathbb{Z}_p^L$.

Figure 9: The protocol Π_{DPF}^B with bounded right input length of every NIOTE call by $B = 2^b$.

where $\mathbf{u}_{0,\beta} = (\beta, 0, \dots) \in \mathbb{Z}_p^{2^\ell}$. If $j = \alpha$, then $\mathbf{v}[\alpha] = \mathbf{u}_{0,\beta}[0] = \beta$, so the special value is correctly positioned. If $j \neq \alpha$, then by the same argument $\mathbf{v}[j] = \mathbf{u}_{0,\beta}[j']$ for some $j' \neq 0$, and therefore all other entries are zero. Our DPF protocol starts with a secret share of $\mathbf{v}_0 := \mathbf{u}_{0,\beta}$ and iteratively applies a permutation to the vector. In iteration $l = 0, \dots, \ell - 1$, we obviously shift each element by 2^l positions, conditioned on the l -th bit α_l , i.e.,

$$\mathbf{v}_{l+1} := (1 - \alpha_l) \cdot \mathbf{v}_l + \alpha_l \cdot \underbrace{\text{SHUF}(\mathbf{v}_l, 2^l)}_{\mathbf{v}'_l[j] \leftarrow \mathbf{v}_l[j \oplus 2^l]} = \mathbf{v}_l + \text{SHUF}(\alpha_l \cdot \mathbf{v}_l, 2^l) - \alpha_l \cdot \mathbf{v}_l,$$

where SHUF denotes the permutation. Since SHUF only rearranges elements, $\alpha_l \cdot \text{SHUF}(\mathbf{v}_l, \cdot) = \text{SHUF}(\alpha_l \cdot \mathbf{v}_l, \cdot)$. Note that if $\alpha_l = 0$, no permutation is applied; if $\alpha_l = 1$, the shift is applied. We compute $\alpha_l \cdot \mathbf{v}_l$ using the non-interactive scalar-vector multiplication with *sublinear* communication cost. More precisely, since our NIOTE-based instantiation of the non-interactive scalar-vector multiplication produces digests/encodings of size $\mathcal{O}(\lambda \log L)$, the DPF protocol's communication cost is $\mathcal{O}(\ell \cdot \lambda \cdot \log L)$.

Note that the setup protocol described above requires $\log L$ rounds, since the vector input to the non-interactive scalar-vector multiplication, $\mathbf{v}_{l+1}$, depends on the output of the previous round, $\alpha_l \cdot \mathbf{v}_l$. Figure 10 details the construction.

Remark 4. The scheme resulting from the compilation of $\Pi_{\text{nDPF}}^{\text{shuf}}$ can be instantiated with any *non-interactive scalar-vector multiplication* scheme, defined as a variant of the non-interactive (matrix) multiplication from [BDSS25]. Our instantiation, the non-interactive tensor product has the same functionality but achieves logarithmic encoding sizes compared to [BDSS25].

Correctness. We write the index $\alpha = \sum_{l=0}^{\ell-1} \alpha_l 2^l$, where each bit, α_l , is additively-shared among the parties. We prove via induction over $l \in \{0, 1, \dots, \ell\}$ the statement $\sum_{i=1}^N \mathbf{v}_l^i = \mathbf{u}_{\hat{\alpha}_l, \beta}$, where $\hat{\alpha}_l = \sum_{j=0}^{l-1} \alpha_j 2^j$, i.e., $\hat{\alpha}_l$ is the index comprising the l least significant bits.

The base case $l = 0$ is evident, as $\sum_{i=1}^N \mathbf{v}_0^i = \sum_{i=1}^N \mathbf{u}_{0, \langle \beta \rangle_i} = \mathbf{u}_{0, \beta}$, (clearly $\hat{\alpha}_0 = 0$). We now prove the induction step from l to $l + 1$. First, note that $\sum_{i=1}^N \mathbf{w}_l^i = \alpha_l \cdot \mathbf{v}_l$ by correctness of NIOTE where $\alpha_l = \sum_{i=1}^N \langle \alpha_l \rangle_i$ and $\mathbf{v}_l = \sum_{i=1}^N \mathbf{v}_l^i$. Then,

$$\begin{aligned} \sum_{i=1}^N \mathbf{v}_{l+1}^i &= \sum_{i=1}^N \left(\mathbf{v}_l^i + \text{SHUF}(\mathbf{w}_l^i, 2^l) - \mathbf{w}_l^i \right) \\ &= \mathbf{v}_l + \text{SHUF} \left(\sum_{i=1}^N \mathbf{w}_l^i, 2^l \right) - \sum_{i=1}^N \mathbf{w}_l^i && (\text{SHUF is linear in } \mathbf{w}) \\ &= \mathbf{v}_l + \text{SHUF}(\alpha_l \cdot \mathbf{v}_l, 2^l) - \alpha_l \cdot \mathbf{v}_l \\ &= \mathbf{u}_{\hat{\alpha}_l, \beta} + \alpha_l \cdot \left(\text{SHUF}(\mathbf{u}_{\hat{\alpha}_l, \beta}, 2^l) - \mathbf{u}_{\hat{\alpha}_l, \beta} \right) && (\text{ind. hyp}) \\ &= \mathbf{u}_{\hat{\alpha}_l, \beta}[k] + \alpha_l \cdot \left(\mathbf{u}_{\hat{\alpha}_l, \beta}[k \oplus 2^l] - \mathbf{u}_{\hat{\alpha}_l, \beta}[k] \right) && \forall k \in [2^\ell] \\ &= \mathbf{u}_{\hat{\alpha}_l, \beta}[k \oplus \alpha_l 2^l] && \forall k \in [2^\ell] \\ &= \mathbf{u}_{0, \beta}[k \oplus \sum_{j=0}^{l-1} \alpha_j 2^j + \alpha_l 2^l] = \mathbf{u}_{\hat{\alpha}_{l+1}, \beta}, \end{aligned}$$

Multi-Party DPF Protocol $\Pi_{\text{nDPF}}^{\text{shuf}}$

PARAMETERS: Number of parties N , plaintext modulus p , output length $L = 2^\ell$, point function $f_{\alpha,\beta} : [L] \rightarrow \mathbb{Z}_p$, N -party NIOTE scheme $\text{NIOTE} = (\text{Setup}, \text{En}, \text{De})$ for $\mathbb{Z}_p^L \otimes \mathbb{Z}_p^1$. Let $\text{SHUF}(\mathbf{x}, 2^j) := (x[0 \oplus 2^j], x[1 \oplus 2^j], \dots, x[L \oplus 2^j])$ be a permutation over $\mathbf{x} \in \mathbb{Z}_p^L$.
 PARTY INPUT: Shares $\{\langle \alpha_t \rangle_i\}_{t \in [\ell]}$ of bits $\alpha_t \in \{0, 1\}$ and a share $\langle \beta \rangle_i \in \mathbb{Z}_p$.
 OUTPUT: A vector $\mathbf{u}^i \in \mathbb{Z}_p^L$ such that $\sum_{i=1}^N \mathbf{u}^i = \mathbf{u}_{\alpha,\beta}$.

Setup: Parties run $\text{pk} \leftarrow \text{Setup}(1^\lambda)$.

FullEval: Each party $\mathcal{P}_i$ holds $\{\langle \alpha_l \rangle_i\}_{l \in [\ell]}$ and $\langle \beta \rangle_i$.

1. Set $\mathbf{v}_0 := (\langle \beta \rangle_i, 0, \dots, 0)^\top \in \mathbb{Z}_p^L$.
2. For $l \in [\ell]$:
 - (a) Compute $(M_l^i, \mathbf{v}_l^i) \leftarrow \text{En}(\text{pk}, \mathbf{v}_l, \langle \alpha_l \rangle_i)$ and broadcast M_l^i .
 - (b) Compute $\mathbf{w}_l := \text{De}(\text{pk}, \{M_l^j\}_{j \neq i}, \mathbf{v}_l^i) \in \mathbb{Z}_p^L$.
 - (c) Set $\mathbf{v}_{l+1} := \mathbf{v}_l + \text{SHUF}(\mathbf{w}_l, 2^l) - \mathbf{w}_l$.
3. Output $\mathbf{v}_\ell$.

Figure 10: N -party Distributed Point Function with $\log L$ rounds using SHUF.

where the last two transformations used the fact that $\mathbf{u}_{x,\beta}[k] = \mathbf{u}_{0,\beta}[k \oplus x], \forall k \in [2^\ell]$. Since $\mathbf{u}_{x,\beta}[x] = \beta = \mathbf{u}_{0,\beta}[0] = \mathbf{u}_{0,\beta}[x \oplus x]$, and $\mathbf{u}_{x,\beta}[x'] = 0$ for any $x' \neq x$ otherwise. We conclude the correctness proof by noting that $\hat{\alpha}_\ell = \alpha$ and therefore $\sum_{i=1}^N \mathbf{v}_\ell^i = \mathbf{u}_{\alpha,\beta}$, as desired.

Security.

Theorem 3. *Let p be a prime and $L = 2^\ell$ the domain size. The protocol $\Pi_{\text{nDPF}}^{\text{shuf}}$ (Figure 10) securely realizes $\mathcal{F}_{\text{nDPF}}$ against a semi-honest adversary corrupting up to $N - 1$ parties in the $\mathcal{F}_{\text{OTE}}$ -hybrid model.*

Proof. Security follows from the security of the N -party NIOTE (or the interaction with $\mathcal{F}_{\text{OTE}}$, respectively). The parties only exchange messages M_l^i (which can be simulated from the N -party NIOTE security) and otherwise compute local operations on their shares. $\square$

6 Applications

We discuss two applications in more detail.

6.1 Distributed ORAM and RAM-MPC: Private Point Queries from Secret-Shared Addresses

Oblivious RAM (ORAM) hides memory access patterns by ensuring that transcripts corresponding to any two logical access sequences of the same length are computationally indistinguishable [GO96, LO13]. A common design pattern in distributed ORAM (DORAM) and in MPC with RAM-style memory is to reduce each logical read and write to (i) generating a *private point query* for an address $\alpha \in [L]$, and (ii) applying this point query to the distributed memory representation. In this subsection, we focus only on this address-to-point-query subroutine and we do not propose a complete ORAM protocol. Many constructions implement point queries using DPF-style techniques [Ds17, BKKO20, ZYP24, VH23, SVG24]. Our multi-party DPF is well-suited for such settings: it supports privacy against up to $N - 1$ semi-honest corruptions among N servers, and provides succinct keys. In the RAM-MPC regime where the address itself is secret-shared, our Π_{DPF}^B protocol reduces the number of synchronous interaction needed to generate the point query.

Point queries and block updates. We write $[L] = \{0, \dots, L-1\}$. Let $\mathbf{x} \in (\mathbb{Z}_p^w)^L$ be an array of L blocks, where each block $\mathbf{x}[j] \in \mathbb{Z}_p^w$ consists of w field elements. For $\alpha \in [L]$, let $\mathbf{u}_{\alpha,1} \in \mathbb{Z}_p^L$ denote the unit vector with $\mathbf{u}_{\alpha,1}[\alpha] = 1$ and 0 elsewhere. It chooses which block to read or update. For a value $\beta \in \mathbb{Z}_p^w$, we define the corresponding sparse block update vector $\mathbf{U}_{\alpha,\beta} \in (\mathbb{Z}_p^w)^L$ by $\mathbf{U}_{\alpha,\beta}[j] = \mathbf{u}_{\alpha,1}[j] \cdot \beta$, i.e., by component-wise scalar-by-block multiplication in $\mathbb{Z}_p^w$. Equivalently, if we flatten $\mathbf{x}$ into $\mathbb{Z}_p^{wL}$, then $\mathbf{U}_{\alpha,\beta}$ is the tensor product $\mathbf{u}_{\alpha,1} \otimes \beta$.

A point *read* returns $\mathbf{x}[\alpha] \in \mathbb{Z}_p^w$ and can be written as the block-wise inner product

$$\mathbf{x}[\alpha] = \sum_{j=0}^{L-1} \mathbf{u}_{\alpha,1}[j] \cdot \mathbf{x}[j] \in \mathbb{Z}_p^w.$$

A point *write* which adds β at address α is the sparse update $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{U}_{\alpha,\beta}$, i.e., $\mathbf{x}[\alpha] \leftarrow \mathbf{x}[\alpha] + \beta$. Both operations can be combined into one, oblivious operation if we introduce a write flag as a selector to select the result of read or write. For simplicity of exposition, we keep the operations separate.

Let $L = 2^\ell$. We consider the case where the address α is secret-shared with bit shares $\{\langle \alpha_b \rangle_i\}_{b \in [\ell], i \in [N]}$, where $\alpha_b \in \{0,1\} \subseteq \mathbb{Z}_p$. We let the memory be either public $\mathbf{x} \in (\mathbb{Z}_p^w)^L$ or additively shared as $\{\mathbf{x}^i\}_{i \in [N]}$ with $\sum_i \mathbf{x}^i = \mathbf{x}$.

Proposition 4 (Read & write from a secret-shared address). *Assume a correct N -party NIOTE primitive NIOTE over $\mathbb{Z}_p$ and a protocol Π_{DPF}^B that, on input additive shares of the address bits $\{\langle \alpha_b \rangle_i\}_{b \in [\ell], i \in [N]}$ and additive shares of a scalar payload $\{\langle \beta \rangle_i\}_{i \in [N]} \subseteq \mathbb{Z}_p$, outputs additive shares $\{\mathbf{u}_{\alpha,\beta}^i\}_{i \in [N]} \subseteq \mathbb{Z}_p^L$ such that $\sum_i \mathbf{u}_{\alpha,\beta}^i = \mathbf{u}_{\alpha,\beta}$. In particular, running Π_{DPF}^B with $\beta = 1$ yields additive shares of the selector $\mathbf{u}_{\alpha,1} \in \mathbb{Z}_p^L$.*

Then there exist N -party MPC protocols that, on input additive shares of α (and optionally of $\beta \in \mathbb{Z}_p^w$), implement:

1. *READ:* output additive shares of $\mathbf{x}[\alpha] \in \mathbb{Z}_p^w$;
2. *WRITE:* update shared memory from $\{\mathbf{x}^i\}_i$ to additive shares of $\mathbf{x} + \mathbf{U}_{\alpha,\beta}$.

For block updates, one can obtain additive shares of $\mathbf{U}_{\alpha,\beta}$ by executing Π_{DPF}^B independently on each coordinate of β . The round complexity is unchanged.

Moreover, the address-to-selector step costs $(\log_2 b + \log_B L)$ rounds, where $1 \leq b \leq \ell$ and $B = 2^b$. All remaining work is local linear algebra plus, for shared-memory reads, a single batched multiplication-layer.

Proof. See Appendix B.1.

Many distributed ORAM and RAM-MPC designs invoke several private point queries per single read/write operation, e.g., across tree levels, within buckets, and for auxiliary state such as position maps and stashes, [Ds17, JW18, VHG23, SVG24]. Because our tensor-tree schedule is fixed and all NIOTE instances within a level are independent, these point-query generations can be batched.

Remark 5 (Cost of Read in Shared Memory). If $\mathbf{x}$ is additively shared, then READ requires computing the products $\mathbf{u}_{\alpha,1}[j] \cdot \mathbf{x}[j]$ for all $j \in [L]$, i.e., L block multiplications. These can be performed in a single batched multiplication layer using standard preprocessing (e.g., Beaver triples), but the total number multiplication is linear in the size of the addressed array. In ORAM applications, this subroutine is typically applied to small structures such as buckets, so the linear factor is in the bucket size rather than in the full database size.

Corollary 1 (Batched Point-Query Generation). *Let $\alpha^{(1)}, \dots, \alpha^{(T)} \in [L]$ be T secret-shared addresses, and optionally let $\beta^{(1)}, \dots, \beta^{(T)} \in \mathbb{Z}_p^w$ be secret-shared values. There is a batched variant of Π_{DPF}^B that produces additive shares of $\mathbf{u}_{\alpha^{(\iota)}, 1}$ and of $\mathbf{U}_{\alpha^{(\iota)}, \beta^{(\iota)}}$ for all $\iota \in [T]$ using the same $\Theta(\log_2 b + \log_B L)$ synchronous rounds as a single instance, by executing all NIOTE calls for all ι in parallel within each tensor-tree level and concatenating the per-level broadcasts.*

Structure-Aware Batching. In several ORAM protocols, the public control flow fixes a *pattern* of subqueries, for example “read a full bucket” or “touch one bucket per level on a root-to-leaf path”. In these cases, one can exploit the public prefix-sharing structure while keeping the transcript independent of the secret addresses beyond the public pattern. We formalize an ORAM-compatible leakage requirement and give two concrete structure-aware batch protocols in Appendix B.

6.2 PCF for OLE and Beaver-Style Correlations

Hasler–Reisert [HR25] construct a two-party pseudo-random correlation function (PCF) for OLE and Beaver-style correlations. At a high level, a PCF compresses a large correlated object (e.g., a matrix of correlations) into two keys so that later, given a public query point, each party can *locally* compute an additive share of the correlated value at that point. In the HR25 construction, each correlated sample is expressed as a structured sum of outer products of two sparse “selector” vectors $\mathbf{e}_0^{(\ell)} \in \mathbb{F}^{m_0}$ and $\mathbf{e}_1^{(\ell')} \in \mathbb{F}^{m_1}$. A single outer-product term contributes the following value to coordinate $(i, j) \in [m_0] \times [m_1]$: $f_{\ell, \ell'}(i, j) := \mathbf{e}_0^{(\ell)}[i] \cdot \mathbf{e}_1^{(\ell')}[j]$. The overall correlation at (i, j) is obtained by summing $f_{\ell, \ell'}(i, j)$ over the relevant pairs (ℓ, ℓ') , together with additional public structure and scalings.

To support *local* evaluation at a queried coordinate $(\hat{i}, \hat{j})$, HR25 generates, for each required pair (ℓ, ℓ') , an FSS key pair for the bivariate function $f_{\ell, \ell'}(\cdot, \cdot)$. Each party evaluates its FSS key at $(\hat{i}, \hat{j})$ to obtain an additive share of $f_{\ell, \ell'}(\hat{i}, \hat{j})$, and summing across all pairs reconstructs the desired correlation entry.

The dominant cost comes from realizing the per-pair sharings $f_{\ell, \ell'}(i, j) = \mathbf{e}_0^{(\ell)}[i] \cdot \mathbf{e}_1^{(\ell')}[j]$ when the selectors are t -sparse, where t is their sparse-LPN sparsity parameter. HR25 implement these sharings using a tensor-product FSS that requires an *evaluation-symmetry* property: the evaluation algorithm must be independent of the party index. This symmetry requirement prevents asymmetric optimizations and leads to a key-size expression with a dominant t^2 contribution.

Our Solution. We replace the tensor-FSS layer by an NIOTE-based asymmetric outer-product primitive NIOPE (Non-Interactive Outer Product Evaluation). Instead of producing a *symmetric* FSS key pair for $f_{\ell, \ell'}$, NIOPE produces two party-specific keys whose evaluations may differ but still add up to $(i, j) \mapsto \mathbf{e}_0^{(\ell)}[i] \cdot \mathbf{e}_1^{(\ell')}[j]$, i.e., to entries of $\mathbf{e}_0^{(\ell)} \otimes \mathbf{e}_1^{(\ell')}$. This preserves the correctness of the overall PCF evaluation while removing the evaluation-symmetry constraint that leads to the t^2 in the key-size.

NIOPE Syntax and Security. Let $\mathbb{F}$ be a finite field and let $m_0, m_1 \geq 1$. Define the outer-product function family $\mathbb{F}_{m_0, m_1}^\otimes = \{f_{\mathbf{x}, \mathbf{y}} : [m_0] \times [m_1] \rightarrow \mathbb{F} \mid f_{\mathbf{x}, \mathbf{y}}(i, j) = x_i \cdot y_j\}$.

Definition 6 (Non-Interactive Outer-Product Evaluation (NIOPE)). *A two-party NIOPE for $\mathbb{F}_{m_0, m_1}^\otimes$ is a pair of PPT algorithms $\text{NIOPE} = (\text{KeyGen}, \text{Eval})$ with:*

- $\text{KeyGen}(1^\lambda, \mathbf{x}, \mathbf{y})$, where $\mathbf{x} \in \mathbb{F}^{m_0}$ and $\mathbf{y} \in \mathbb{F}^{m_1}$, outputs keys (k_0, k_1) .
- $\text{Eval}(\sigma, k_\sigma, i, j)$ outputs $y_\sigma(i, j) \in \mathbb{F}$, for $\sigma \in \{0, 1\}$ and $(i, j) \in [m_0] \times [m_1]$.

We require correctness and one-sided input privacy.

CORRECTNESS: The NIOPE is correct, if for all $\mathbf{x}, \mathbf{y}$ and all (i, j) , and given $(k_0, k_1) \leftarrow \text{KeyGen}(1^\lambda, \mathbf{x}, \mathbf{y})$, we have

$$\Pr [\text{Eval}(0, k_0, i, j) + \text{Eval}(1, k_1, i, j) = x_i \cdot y_j] \geq 1 - \text{negl}(\lambda).$$

It is perfectly correct if above holds with probability 1.

ONE-SIDED INPUT PRIVACY:

- Right-privacy: there exists a PPT simulator $\mathcal{S}_0$ such that for all $\mathbf{x}, \mathbf{y}$, the distribution of k_0 in $\text{KeyGen}(1^\lambda, \mathbf{x}, \mathbf{y})$ is computationally indistinguishable from $\mathcal{S}_0(1^\lambda, \mathbf{x})$.
- Left-privacy: there exists a PPT simulator $\mathcal{S}_1$ such that for all $\mathbf{x}, \mathbf{y}$, the distribution of k_1 in $\text{KeyGen}(1^\lambda, \mathbf{x}, \mathbf{y})$ is computationally indistinguishable from $\mathcal{S}_1(1^\lambda, \mathbf{y})$.

In the HR25 construction, the left and right sparse selector vectors are generated from the parties' local seeds and are therefore already known to their respective owners. Hence, for our substitution it suffices that each party's key may depend on its own selector but hides the other selector: otherwise, a party could recover the other side's sparse selector, or its support, and thus compute the entire outer-product term on its own. Our NIOPE keys inherit exactly this property from NIOTE via encoder privacy, hiding the right input in k_0 , and via hasher privacy, hiding the left input in k_1 .

Constructing NIOPE from NIOTE. We now show that any two-party NIOTE for $\mathbb{F}^{m_0} \otimes \mathbb{F}^{m_1}$ yields a NIOPE for $\mathbb{F}_{m_0, m_1}^\otimes$. Let $\text{NIOTE} = (\text{Setup}, \text{En}_A, \text{En}_B, \text{De}_A, \text{De}_B)$ be a two-party NIOTE in $\mathbb{F}^{m_0} \otimes \mathbb{F}^{m_1}$. We additionally assume *coordinate-decodability*: For each $(i, j) \in [m_0] \times [m_1]$, there are deterministic polynomial-time algorithms $\text{De}_A^{(i, j)}$ and $\text{De}_B^{(i, j)}$ such that

$$\text{De}_A^{(i, j)}(\text{pk}, E, \psi) = \text{De}_A(\text{pk}, E, \psi)[i, j], \quad \text{De}_B^{(i, j)}(\text{pk}, \mathbf{d}, \phi) = \text{De}_B(\text{pk}, \mathbf{d}, \phi)[i, j],$$

and $\text{De}_A^{(i, j)}, \text{De}_B^{(i, j)}$ can be implemented without constructing the full length- $m_0 m_1$ output vector. In the NIOTE, the encoder-side decoding is an explicit public linear map of the form $\text{De}_B(\text{pk}, \mathbf{d}, \phi) = \lceil \mathbf{P}_{\text{pk}} \cdot (\mathbf{d} \otimes \phi \otimes \mathbf{g}_q) \rceil_p$, for a public matrix $\mathbf{P}_{\text{pk}}$ derived from pk (and a standard gadget vector $\mathbf{g}_q$). Hence for each (i, j) we may define $P_{\text{pk}}^{(i, j)}$ as the $(m_1 \cdot i + j)$ -th row of $\mathbf{P}_{\text{pk}}$ and set

$$\text{De}_B^{(i, j)}(\text{pk}, \mathbf{d}, \phi) := \lceil \langle P_{\text{pk}}^{(i, j)}, \mathbf{d} \otimes \phi \otimes \mathbf{g}_q \rangle \rceil_p.$$

Importantly, $P_{\text{pk}}^{(i, j)}$ is never explicitly described and is instead generated on the fly from pk and (i, j) . Our construction is given in Figure 11.

We show correctness of NIOPE directly follows from NIOTE correctness and coordinate-decodability.

Lemma 3 (Correctness of NIOPE). *If NIOTE is perfectly correct, then for all $\lambda, m_0, m_1 \geq 1$, all $\mathbf{x} \in \mathbb{F}^{m_0}$, $\mathbf{y} \in \mathbb{F}^{m_1}$, and all $(i, j) \in [m_0] \times [m_1]$, the outputs of NIOPE_{m_0, m_1} satisfy $y_0(i, j) + y_1(i, j) = x_i \cdot y_j$.*

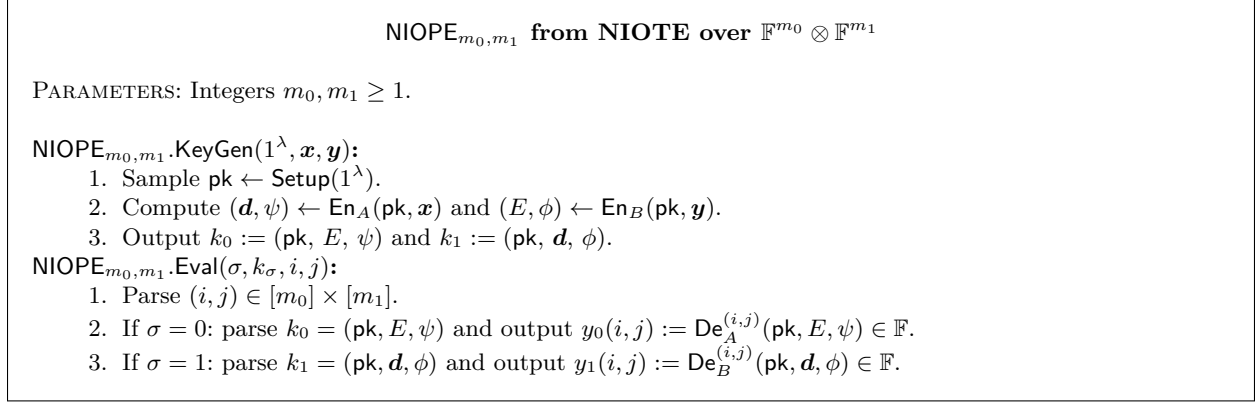


Figure 11: Our NIOPE built from NIOTE.

Proof. By NIOTE perfect correctness, we have $\text{De}_A(\mathbf{pk}, E, \psi) + \text{De}_B(\mathbf{pk}, \mathbf{d}, \phi) = \mathbf{x} \otimes \mathbf{y}$. Projecting to coordinate (i, j) and using coordinate-decodability gives $\text{De}_A^{(i,j)}(\mathbf{pk}, E, \psi) + \text{De}_B^{(i,j)}(\mathbf{pk}, \mathbf{d}, \phi) = (\mathbf{x} \otimes \mathbf{y})[i, j] = x_i y_j$. $\square$

Lemma 4. *If NIOTE is encoder-private, then NIOPE is right-private. If NIOTE is hasher-private, then NIOPE is left-private.*

Proof. Right-privacy: to simulate $k_0 = (\mathbf{pk}, E, \psi)$ given $\mathbf{x}$, the simulator samples $\mathbf{pk} \leftarrow \text{Setup}(1^\lambda)$, runs $(\cdot, \psi) \leftarrow \text{En}_A(\mathbf{pk}, \mathbf{x})$, and simulates E using the encoder-privacy simulator for NIOTE. The resulting distribution matches k_0 . Left-privacy is analogous, using the hasher-privacy simulator to simulate $\mathbf{d}$ and running En_B to obtain ϕ given $\mathbf{y}$. $\square$

The NIOPE in Figure 11 is already a two-party “PCF-like” primitive for the outer-product correlation $\mathbf{x} \otimes \mathbf{y}$. On query (i, j) , each party outputs a local share of $x_i y_j$ and the shares add up correctly. Therefore, in the HR25 construction one can replace each tensor-FSS instance for $f_{\ell, \ell'}(i, j) = \mathbf{e}_0^{(\ell)}[i] \cdot \mathbf{e}_1^{(\ell')}[j]$ by one call to NIOPE instantiated with $(\mathbf{x}, \mathbf{y}) = (\mathbf{e}_0^{(\ell)}, \mathbf{e}_1^{(\ell')})$, while leaving the rest of their PCF unchanged.

Acknowledgments. The authors would like to thank Lawrence Roy for helpful discussions on the NIOTE primitive. This work was partly supported by the junior postdoctoral fellowship 12A1F26N from the Research Foundation Flanders (FWO).

References

- AMR25. Damiano Abram, Giulio Malavolta, and Lawrence Roy. Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 1875–1886. ACM Press, June 2025.
- AMR26. Damiano Abram, Giulio Malavolta, and Lawrence Roy. Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits, 2026.
- ARS24. Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024.
- AS22. Damiano Abram and Peter Scholl. Low-communication multiparty triple generation for SPDZ from ring-LPN. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 221–251. Springer, Cham, March 2022.

- BBC⁺24. Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$ OLE-based multi-party computation for boolean circuits. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 69–101. Springer, Singapore, December 2024.
- BC25. Dung Bui and Kelong Cong. Efficient fuzzy labeled PSI from vector ring-OLE. In Yongdae Kim, Atsuko Miyaji, and Mehdi Tibouchi, editors, *CANS 25*, volume 16351 of *LNCS*, pages 173–198. Springer, Singapore, November 2025.
- BCG⁺19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.
- BCG⁺20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020.
- BCG⁺21. Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 871–900. Springer, Cham, October 2021.
- BCG⁺22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022.
- BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.
- BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011.
- BDSS25. Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. Non-interactive distributed point functions. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part I*, volume 15674 of *LNCS*, pages 3–35. Springer, Cham, May 2025.
- BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, April 2015.
- BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.
- BGI19. Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 341–371. Springer, Cham, December 2019.
- BGIK22. Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 121–151. Springer, Cham, August 2022.
- BKKO20. Paul Bunn, Jonathan Katz, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient 3-party distributed ORAM. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 215–232. Springer, Cham, September 2020.
- BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.
- Ble89. Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, November 1989.
- Ble90. Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.
- BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Berlin, Heidelberg, August 2013.

- CBM15. Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.
- CKY25. Geoffroy Couteau, Naman Kumar, and Xiayi Ye. Multiparty homomorphic secret sharing and more from LPN and MQ. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 498–530. Springer, Cham, December 2025.
- COPZ25. Kelong Cong, Emmanuela Orsini, Erik Pohle, and Oliver Zajonc. Row reduction techniques for n-party garbling. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 522–555. Springer, Cham, August 2025.
- COS⁺22. Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel P. Smart, and Barry van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 540–563. Springer, Cham, September 2022.
- Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503. Springer, Cham, May 2019.
- DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Berlin, Heidelberg, August 2016.
- DIL⁺22. Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Streaming and unbalanced PSI from function secret sharing. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 564–587. Springer, Cham, September 2022.
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.
- Ds17. Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.
- GGM24. Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 309–345. Springer, Cham, August 2024.
- GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.
- GO96. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
- GRS22. Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 323–352. Springer, Cham, August 2022.
- GW25. Aarushi Goel, Mingyuan Wang, and Zhiheng Wang. Multiparty distributed point functions. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part IV*, volume 16003 of *LNCS*, pages 140–173. Springer, Cham, August 2025.
- HR25. Sebastian Hasler and Pascal Reiser. Pseudorandom correlation functions for multiparty beaver triples from sparse LPN. Cryptology ePrint Archive, Report 2025/2002, 2025.
- JGB⁺24. Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based secure training and inference with GPUs. In *2024 IEEE Symposium on Security and Privacy*, pages 597–616. IEEE Computer Society Press, May 2024.
- JW18. Stanislaw Jarecki and Boyang Wei. 3PC ORAM with low latency, low bandwidth, and fast batch retrieval. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 2018*, volume 10892 of *LNCS*, pages 360–378. Springer, Cham, July 2018.
- KPM24. Chandan Kumar, Sikhar Patranabis, and Debdeep Mukhopadhyay. Compact key function secret sharing with non-linear decoder. *CiC*, 1(2):8, 2024.
- KPR25. Toomas Krips and Pille Pullonen-Raudvere. Multi-party distributed point functions with polylogarithmic key size from invariants of matrices. Cryptology ePrint Archive, Report 2025/978, 2025.
- LO13. Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 377–396. Springer, Berlin, Heidelberg, March 2013.

- LXYY25. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part V*, volume 15605 of *LNCS*, pages 145–175. Springer, Cham, May 2025.
- Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.
- RTPB22. Théo Ryffel, Pierre Tholoniati, David Pointcheval, and Francis R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *PoPETs*, 2022(1):291–316, January 2022.
- SGRR19. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.
- SVG24. Sajin Sasy, Adithya Vadapalli, and Ian Goldberg. PRAC: Round-efficient 3-party MPC for dynamic data structures. *PoPETs*, 2024(3):692–714, July 2024.
- TSS⁺20. Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 43(2):95–107, 2020.
- VHG23. Adithya Vadapalli, Ryan Henry, and Ian Goldberg. Duoram: A bandwidth-efficient distributed ORAM for 2- and 3-party computation. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3907–3924. USENIX Association, August 2023.
- Wag22. Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *PoPETs*, 2022(4):351–377, October 2022.
- ZYP24. Guy Zyskind, Avishay Yanai, and Alex ‘Sandy’ Pentland. High-throughput three-party DPFs with applications to ORAM and digital currencies. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4152–4166. ACM Press, October 2024.

Additional Supplementary Material

A Details on the Succinct NIOTE

We will first give details on the half-succinct NIOTE with perfect correctness.

A.1 Half-Succinct NIOTE with Perfect Correctness

The protocol is depicted in Figure 12 in adapted form from [AMR25, Construction 3.4] with minor changes: We use the standard technique to scale the message by $\Delta = q/p$ and use the two-party, local rounding property to achieve perfect correctness. We also sample $\mathbf{B}$ uniformly in $\mathbb{Z}_q$ (instead of in $\mathbb{Z}_2$ as done in [AMR25]).

Correctness. In the following, we recall perfect correctness of the construction for appropriately chosen p and q , where $p/q \in \lambda^{-\omega(\lambda)}$.

We first show that $\text{De}_B^{\text{raw}}(\text{pk}, \mathbf{d}, \phi = \mathbf{s}) = \text{De}_{B, \text{raw}}^{\text{Bits}}(\text{pk}, \hat{\mathbf{d}}, \phi = \hat{\mathbf{s}})$, if $\hat{\mathbf{d}} = \text{Bits}(\mathbf{d})$ and $\hat{\mathbf{s}} = \text{Bits}(\mathbf{s})$. Note that for any vector $\mathbf{x} \in \mathbb{Z}_q^a$, the bit-decomposition is $\hat{\mathbf{x}} := \text{Bits}(\mathbf{x}) \in \mathbb{Z}_q^{a\ell_q}$. Similarly, composing the bits is $\mathbf{x} := \mathbf{G}_a \hat{\mathbf{x}}$, where $\mathbf{G}_a = \mathbf{I}_a \otimes \mathbf{g}_q^\top$. Then it is easy to see that

$$\text{De}_{B, \text{raw}}^{\text{Bits}}(\text{pk}, \hat{\mathbf{d}}, \phi = \hat{\mathbf{s}}) = - \left(\mathbf{I}_{\bar{m}\ell} \otimes (\mathbf{G}_k \hat{\mathbf{s}})^\top \right) \mathbf{B}^\top (\mathbf{G}_n \hat{\mathbf{d}}) = - \left(\mathbf{I}_{\bar{m}\ell} \otimes \mathbf{s}^\top \right) \mathbf{B}^\top \mathbf{d},$$

as desired.

Lemma 5 (Local-Rounding Correctness Modulo p). *Let $\mathbf{x} \in \mathbb{Z}_p^{\bar{m}}$ and $\mathbf{y} \in \mathbb{Z}_p^\ell$, and let $\mathbf{v}, \mathbf{w} \in \mathbb{Z}_q^{\bar{m}\ell}$ be produced by Figure 12. Then for all coins of the protocol,*

$$\mathbf{v} + \mathbf{w} = \Delta \cdot (\mathbf{x} \otimes \mathbf{y}) + \mathbf{e},$$

where $m_{\text{hs}} = \bar{m} \cdot \ell_q$ and $\|\mathbf{e}\|_\infty \leq (n+1) m_{\text{hs}} B_\chi$. Consequently, with local rounding and summation modulo p ,

$$\lceil \mathbf{v} \rceil_p + \lceil \mathbf{w} \rceil_p \equiv \mathbf{x} \otimes \mathbf{y} \pmod{p}$$

with probability at least $1 - (\bar{m}\ell) \cdot (\|\mathbf{e}\|_\infty + 1) \cdot \frac{p}{q}$.

Proof. From the algorithms in Figure 12, we get

$$\mathbf{v} + \mathbf{w} = \Delta \cdot (\mathbf{x} \otimes \mathbf{y}) + \mathbf{E}'^\top (\mathbf{A} \hat{\mathbf{x}}) + \tilde{\mathbf{E}}^\top \hat{\mathbf{x}}.$$

Each coordinate is bounded as $\|\mathbf{E}'^\top (\mathbf{A} \hat{\mathbf{x}})\|_\infty \leq n m_{\text{hs}} B_\chi$, $\|\tilde{\mathbf{E}}^\top \hat{\mathbf{x}}\|_\infty \leq m_{\text{hs}} B_\chi$, so $\|\mathbf{e}\|_\infty \leq (n+1) m_{\text{hs}} B_\chi$. Applying the standard two-share rounding bound and a union bound over the $\bar{m}\ell$ coordinates yields the stated overall success probability. $\square$

Fully Correct Half-Succinct NIOTE

PARAMETERS. Let $q = 2^{\text{poly}(\lambda)}$ (power of two), $p \mid q$, $\Delta := q/p$, $\ell_q := \log_2 q$, $k = \Theta(\lambda)$, $n := k \cdot \ell_q$. Choose $\bar{m}, \ell \in \mathbb{N}$ and set $m := \bar{m} \cdot \ell_q$. Let $\mathbf{g}_q = (1, 2, \dots, 2^{\ell_q-1})^\top \in \mathbb{Z}_q^{\ell_q}$ and $\mathbf{G}_a = (\mathbf{I}_a \otimes \mathbf{g}_q^\top) \in \mathbb{Z}_q^{a \times \ell_q}$. Noise distribution χ is $B(\lambda)$ -bounded.

- **Setup**(1^λ): Sample $\mathbf{A} \in \mathbb{Z}_2^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{n \times (\bar{m}\ell \cdot k)}$. Publish $\text{pk} := (\mathbf{A}, \mathbf{B})$.
- **En_A**($\text{pk}, \mathbf{x} \in \mathbb{Z}_p^{\bar{m}}$): $\hat{\mathbf{x}} := \text{Bits}(\mathbf{x}) \in \mathbb{Z}_2^m$; Set $\mathbf{d} := \mathbf{A}\hat{\mathbf{x}} \in \mathbb{Z}_q^n$. Output $(\mathbf{d}, \psi := (\hat{\mathbf{x}}))$.
- **En_B**($\text{pk}, \mathbf{y} \in \mathbb{Z}_p^\ell$): Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^k$, $\mathbf{E}' \leftarrow \chi^{n \times \bar{m}\ell}$ and $\tilde{\mathbf{E}} \leftarrow \chi^{m \times \bar{m}\ell}$. Let $\mathbf{T} := \mathbf{B} \cdot (\mathbf{I}_{\bar{m}\ell} \otimes \mathbf{s}) + \mathbf{E}' \in \mathbb{Z}_q^{n \times \bar{m}\ell}$ and compute the encoding as

$$\mathbf{C} := \mathbf{A}^\top \mathbf{T} + \tilde{\mathbf{E}} + (\mathbf{I}_{\bar{m}} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{\bar{m}} \otimes \Delta \mathbf{y}^\top) \in \mathbb{Z}_q^{m \times \bar{m}\ell}.$$

Send $E := \mathbf{C}$ and store $\phi := \mathbf{s}$.

- **De_A^{raw}**($\text{pk}, E = \mathbf{C}, \psi = \hat{\mathbf{x}}$): Compute $\mathbf{v} := \mathbf{C}^\top \hat{\mathbf{x}} \in \mathbb{Z}_q^{\bar{m}\ell}$ and return $\mathbf{v}$.
- **De_B^{raw}**($\text{pk}, \mathbf{d}, \phi = \mathbf{s}$): Set $\mathbf{w} := -(\mathbf{I}_{\bar{m}\ell} \otimes \mathbf{s}^\top) \mathbf{B}^\top \mathbf{d} \in \mathbb{Z}_q^{\bar{m}\ell}$ and return $\mathbf{w}$.
- **De_{B,raw}^{Bits}**($\text{pk}, \hat{\mathbf{d}}, \hat{\mathbf{s}}$): Set $\mathbf{w} := -(\mathbf{I}_{\bar{m}\ell} \otimes \hat{\mathbf{s}}^\top \mathbf{G}_k^\top) \mathbf{B}^\top \mathbf{G}_n \hat{\mathbf{d}} \in \mathbb{Z}_q^{\bar{m}\ell}$ and return $\mathbf{w}$.
- $\text{De}_A := \lceil \text{De}_A^{\text{raw}} \rceil_p$
- $\text{De}_B := \lceil \text{De}_B^{\text{raw}} \rceil_p$
- $\text{De}_B^{\text{Bits}} := \lceil \text{De}_{B,\text{raw}}^{\text{Bits}} \rceil_p$.

Figure 12: Half-succinct NIOTE (adapted from [AMR25]) with full correctness.

Security. We show encoder privacy.

Lemma 6 (Encoder Privacy for Figure 12). *Let m, ℓ, q be positive integers, $k = k(\lambda)$, $n = k \log_2 q$, then assuming the hardness of $(\chi, k, n \cdot \bar{m}\ell, q)$ -LWE and $(\chi, k, m+k, q)$ -LWE, the construction in Figure 12 is encoder-private.*

Proof. The proof uses the same strategy as [AMR25], adapted to our construction. Consider the following hybrid distributions.

- (Hybrid 0) Given pk from **Setup**, the encoding $\mathbf{C}$ is computed as in the construction.
- (Hybrid 1) where $\mathbf{U} \leftarrow \mathbb{Z}_q^{n \times \bar{m}\ell}$ is sampled uniformly at random and we compute the encoding as

$$\mathbf{C} := \mathbf{A}^\top \mathbf{U} + \tilde{\mathbf{E}} + (\mathbf{I}_{\bar{m}} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{\bar{m}} \otimes \Delta \mathbf{y}^\top).$$

We can rewrite the result of $\mathbf{B} \cdot (\mathbf{I}_{\bar{m}\ell} \otimes \mathbf{s})$ as $\mathbf{T}_{i,j} = \sum_{\iota=1}^k \mathbf{B}_{i,k(j-1)+\iota} \cdot \mathbf{s}_\iota$, for all $1 \leq i \leq n$, $1 \leq j \leq \bar{m}\ell$. So by chopping $\mathbf{B}$ into $\bar{m}\ell$ matrices of size $n \times k$ and stacking them vertically, we obtain the public LWE matrix $\mathbf{B}' \in \mathbb{Z}_q^{n\bar{m}\ell \times k}$. Note that in hybrid 0, the encoding is computed using the LWE sample $\mathbf{U} = \mathbf{B}'\mathbf{s} + \mathbf{E}'$ whereas in hybrid 1, $\mathbf{U}$ is uniformly random. Assuming $(\chi, k, n \cdot \bar{m}\ell, q)$ -LWE is hard, this change is indistinguishable.

- (Hybrid 2) Instead of computing the encoding, we sample and return $\mathbf{C} \leftarrow \mathbb{Z}_q^{m \times \bar{m}\ell}$.

Note that in hybrid 1, we obtain $\mathbf{C} = \mathbf{A}^\top \mathbf{T} + \tilde{\mathbf{E}} + (\mathbf{I}_{\bar{m}} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{\bar{m}} \otimes \Delta \mathbf{y}^\top)$, i.e., $\mathbf{C}$ is formed by $\bar{m}\ell$ independent (χ, n, m, q) -NLWE instances, which are as hard to distinguish as (χ, k, m, q) -LWE (see Theorem 1). We conclude by noting that the simulator for $\mathcal{G}_{\mathcal{A}, \text{Sim}}^{\text{Epriv}}$ returns $\mathbf{C}$ sampled uniformly at random (like in Hybrid 2). $\square$

A.2 Succinct NIOTE

We adapt the bootstrapping procedure from [AMR25, AMR26] that constructs a succinct, hasher-private and encoder-private NIOTE from a half-succinct NIOTE scheme to obtain perfect correctness as follows. We bootstrap tensor products of the *bit-decomposition* of $\mathbf{d} \otimes \phi$ which allows us to round to control the error and then lift the shares into $\mathbb{Z}_q$ again at appropriate times, similar to [BKS19].

For p, q integers, our succinct NIOTE computes over $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^\ell$ with communication $\mathcal{O}(\ell \text{poly}(\lambda) \log m)$ where $m = t^r n$. Let $\ell_q = \log q$. Let $n = k \ell_q$ be the size of the digest of the half-succinct NIOTE (hOTE) instance and k be the size of the encoders secret. We use hOTE for two parameter regimes: for $\mathbb{Z}_p^{tn} \otimes \mathbb{Z}_p^\ell$, denoted hOTE $_\ell$, and for $\mathbb{Z}_p^{tn\ell_q} \otimes \mathbb{Z}_p^n$, denoted hOTE $_n$. Further, the OTE instance (needed for hasher privacy at the root) will compute over $\mathbb{Z}_p^{tn\ell_q} \otimes \mathbb{Z}_p^n$. The protocol is given in Figures 13 and 14.

Protocol Description. The protocol in Figures 13 and 14 implements a NIOTE for $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^\ell$ that is succinct in m by bootstrapping a half-succinct NIOTE along a t -ary tree. We set $m = t^r n$ and interpret the m coordinates of $\mathbf{x}$ as being arranged into t^{r-1} leaf blocks of length tn . At the leaf level, party A encodes each block $\mathbf{x}_i \in \mathbb{Z}_p^{tn}$ using hOTE $_\ell$, obtaining a short digest $\mathbf{h}_{r-1}^i \in \mathbb{Z}_q^n$ and local decoding information ψ_{r-1}^i . All leaf digests are concatenated and then bit-decomposed to a binary vector $\hat{\mathbf{d}}_{r-1} := \text{Bits}(\mathbf{h}_{r-1}^1 \| \dots \| \mathbf{h}_{r-1}^{t^{r-1}})$.

This bit string is then compressed level-by-level towards the root. For each internal level $j = r-2, \dots, 1$, party A groups $\hat{\mathbf{d}}_{j+1}$ into t^j blocks, each block representing the bit-decomposition of t child digests, and encodes each block using hOTE $_n$. This produces new digests $\mathbf{h}_j^i \in \mathbb{Z}_q^n$ (one per parent node) and corresponding local states ψ_j^i . Concatenating and bit-decomposing these digests yields the next-level string $\hat{\mathbf{d}}_j := \text{Bits}(\mathbf{h}_j^1 \| \dots \| \mathbf{h}_j^{t^j})$. After $r-1$ such compressions, party A reaches $\hat{\mathbf{d}}_1$ at the top internal level and finally runs the root protocol OTE.En $_A$ on $\hat{\mathbf{d}}_1$, producing the public digest $\mathbf{d} \in \mathbb{Z}_q^n$ and root state ψ_0 . Overall, En $_A$ is thus a bottom-up tree reduction: the work within each level is parallel across nodes, but levels depend sequentially on the digests produced below.

Party B proceeds dually. Starting from its input $\mathbf{y} \in \mathbb{Z}_p^\ell$, it runs hOTE $_\ell$.En $_B$ once to obtain a leaf encoding E_{r-1} and an encoder secret ϕ_{r-1} ; it then bit-decomposes ϕ_{r-1} and repeatedly re-encodes this bit-decomposed secret upwards using hOTE $_n$.En $_B$, producing encodings E_j and secrets ϕ_j for $j = r-2, \dots, 1$. At the root it runs OTE.En $_B$ on $\hat{\phi}_1$ to obtain E_0 and the final secret ϕ_0 . The overall encoding is the tuple $E = (E_0, \dots, E_{r-1})$, while B keeps only the root secret $\phi = \phi_0$.

Decoding follows the same tree structure and is driven by two ingredients. First, by bilinear encoder evaluation (Lemma 7), the “ B -side” decoding of the half-succinct NIOTE can be expressed as a public linear map (the matrices $\mathbf{P}, \mathbf{P}'$) applied to a tensor of bit-decomposed values; this enables the parties to propagate shares across levels using only public linear operations. Second, to obtain perfect correctness, each level produces local outputs in $\mathbb{Z}_q$ that are *locally rounded* modulo p , and the resulting $\mathbb{Z}_p$ -shares are then lifted back to $\mathbb{Z}_q$ via $\text{Lift}_K^0, \text{Lift}_K^1$ so they can be fed into the next layer without accumulating rounding error. At the leaf level, these operations yield additive shares in $\mathbb{Z}_p^{tn\ell}$ of each block tensor $\mathbf{x}_i \otimes \mathbf{y}$, and concatenating over all i gives shares $\mathbf{v}, \mathbf{w} \in \mathbb{Z}_p^{m\ell}$ of $\mathbf{x} \otimes \mathbf{y}$.

In the following we prove the correctness of our scheme, but before we need some preliminary results.

Succinct Hasher-Private NIOTE (Part I)

This is a NIOTE scheme for $\mathbb{Z}_p^m \otimes \mathbb{Z}_p^\ell$ succinct in m .

PARAMETERS: Let $m = t^r n$. A half-succinct (not hasher-private) NIOTE scheme hOTE where n denotes the digest size for $\mathbb{Z}_p^{tn} \otimes \mathbb{Z}_p^\ell$ (denoted hOTE_ℓ) and for $\mathbb{Z}_p^{tn\ell_q} \otimes \mathbb{Z}_p^n$ (denoted hOTE_n); A hasher-private (not succinct) NIOTE scheme OTE (e.g., [ARS24]) for $\mathbb{Z}_q^{tn\ell_q} \otimes \mathbb{Z}_q^n$;

Let $\mathbf{P}$ be the public matrix derived from $\text{pk}_{\text{hOTE}_\ell}$ (for $\mathbb{Z}_p^{tn} \otimes \mathbb{Z}_p^\ell$) and let $\mathbf{P}'$ be the public matrix derived from $\text{pk}_{\text{hOTE}_n}$ (for $\mathbb{Z}_p^{tn\ell_q} \otimes \mathbb{Z}_p^n$).

Setup(1^λ): Run $\text{pk}_{\text{hOTE}_\ell} \leftarrow \text{hOTE}_\ell.\text{Setup}(1^\lambda)$, $\text{pk}_{\text{hOTE}_n} \leftarrow \text{hOTE}_n.\text{Setup}(1^\lambda)$ and $\text{pk}_{\text{OTE}} \leftarrow \text{OTE}.\text{Setup}(1^\lambda)$.

Sample $K \leftarrow \{0, 1\}^\lambda$. Output $\text{pk} := (\text{pk}_{\text{hOTE}_\ell}, \text{pk}_{\text{hOTE}_n}, \text{pk}_{\text{OTE}}, K)$.

En_A($\text{pk}, \mathbf{x}$): Let $\mathbf{x} \in \mathbb{Z}_p^{tn}$ and $m = t^r n$. Split $\mathbf{x}$ into t^{r-1} blocks $\mathbf{x}_i \in \mathbb{Z}_p^{t \cdot n}$, then

1. $(\mathbf{h}_{r-1}^i, \psi_{r-1}^i) \leftarrow \text{hOTE}_\ell.\text{En}_A(\text{pk}_{\text{hOTE}_\ell}, \mathbf{x}_i)$, for $i \in \{1, \dots, t^{r-1}\}$.

2. Set $\widehat{\mathbf{d}}_{r-1} := \text{Bits}\left(\begin{pmatrix} \mathbf{h}_{r-1}^1 \\ \vdots \\ \mathbf{h}_{r-1}^{t^{r-1}} \end{pmatrix}\right) \in \mathbb{Z}_2^{n t^{r-1} \ell_q}$

3. For $j \in \{r-2, \dots, 1\}$, compute

- (a) Reshape $\widehat{\mathbf{d}}_{j+1}$ into t^j blocks $\widehat{\mathbf{d}}_j^i \in \mathbb{Z}_2^{tn\ell_q}$.

- (b) $(\mathbf{h}_j^i, \psi_j^i) \leftarrow \text{hOTE}_n.\text{En}_A(\text{pk}_{\text{hOTE}_n}, \widehat{\mathbf{d}}_j^i)$, for $i \in \{1, \dots, t^j\}$.

- (c) Set $\widehat{\mathbf{d}}_j := \text{Bits}\left(\begin{pmatrix} \mathbf{h}_j^1 \\ \vdots \\ \mathbf{h}_j^{t^j} \end{pmatrix}\right) \in \mathbb{Z}_2^{n t^j \ell_q}$.

4. Compute $(\mathbf{d}, \psi_0) \leftarrow \text{OTE}.\text{En}_A(\text{pk}_{\text{OTE}}, \widehat{\mathbf{d}}_1)$.

Return digest $\mathbf{d} \in \mathbb{Z}_q^n$ and secret information $\psi := \{\psi_j^i\}_{j \in \{0, 1, \dots, r-1\}, i \in \{1, \dots, t^j\}}$.

En_B($\text{pk}, \mathbf{y}$):

1. Compute $(E_{r-1}, \phi_{r-1}) \leftarrow \text{hOTE}_\ell.\text{En}_B(\text{pk}_{\text{hOTE}_\ell}, \mathbf{y})$. Set $\widehat{\phi}_{r-1} := \text{Bits}(\phi_{r-1}) \in \mathbb{Z}_2^n$.

2. For $j \in \{r-2, \dots, 1\}$, compute

- (a) $(E_j, \phi_j) \leftarrow \text{hOTE}_n.\text{En}_B(\text{pk}_{\text{hOTE}_n}, \widehat{\phi}_{j+1})$. Set $\widehat{\phi}_j := \text{Bits}(\phi_j) \in \mathbb{Z}_2^n$.

3. Compute $(E_0, \phi_0) \leftarrow \text{OTE}.\text{En}_B(\text{pk}_{\text{OTE}}, \widehat{\phi}_1)$.

Return encoding $E := (E_0, \dots, E_{r-1})$ and secret information $\phi := \phi_0$.

Figure 13: Our perfectly correct, succinct and hasher-private NIOTE Part I.

Useful Results. We first show that $\text{hOTE}.\text{De}_B^{\text{Bits}}$ (Figure 12) can be evaluated distributively among the two parties. Given $\mathbf{v} + \mathbf{w} = \widehat{\mathbf{d}} \otimes \widehat{\mathbf{s}}$, where $\widehat{\mathbf{d}} := \text{Bits}(\mathbf{d}) \in \mathbb{Z}_q^{n\ell_q}$ and $\widehat{\mathbf{s}} := \text{Bits}(\mathbf{s}) \in \mathbb{Z}_q^n$, then

$$\begin{aligned} & \text{hOTE}.\text{De}_A^{\text{raw}}(\text{pk}, E, \psi) + \text{hOTE}.\text{De}_{B, \text{raw}}^{\text{Bits}}(\text{pk}, \widehat{\mathbf{d}}, \widehat{\mathbf{s}}) \\ &= \text{hOTE}.\text{De}_A^{\text{raw}}(\text{pk}, E, \psi) + \mathbf{P} \cdot \mathbf{v} + \mathbf{P} \cdot \mathbf{w} \in \mathbb{Z}_q^{\bar{m}\ell}. \end{aligned}$$

The existence of $\mathbf{P}$ is shown in Lemma 7 below.

Lemma 7 (Bilinear Encoder Evaluation of Figure 12). *The half-succinct encoder-private NIOTE in Figure 12 has bilinear encoder evaluation for bit-decomposed digest and encoder secret: there exists a public matrix $\mathbf{P} \in \mathbb{Z}_q^{\bar{m}\ell \times nk\ell_q^2}$, derived from pk , such that for all $\mathbf{d} \in \mathbb{Z}_q^n$, $\phi \in \mathbb{Z}_q^k$,*

$$\text{De}_{B, \text{raw}}^{\text{Bits}}(\text{pk}, \widehat{\mathbf{d}}, \phi = (\widehat{\mathbf{s}})) = \mathbf{P} \cdot (\widehat{\mathbf{d}} \otimes \widehat{\mathbf{s}}),$$

where $\widehat{\mathbf{d}} = \text{Bits}(\mathbf{d}) \in \mathbb{Z}_q^{n\ell_q}$ and $\widehat{\mathbf{s}} = \text{Bits}(\mathbf{s}) \in \mathbb{Z}_q^{k\ell_q}$.

Succinct Hasher-Private NIOTE (Part II)

$\text{De}_A(\text{pk}, E, \psi)$: Parse E as $(E_0, \dots, E_{r-1})$, then

1. Compute $\tilde{\mathbf{v}}_0 := \text{OTE.De}_A(\text{pk}_{\text{OTE}}, E_0, \psi_0) \in \mathbb{Z}_p^{tn^2\ell_q}$. Lift $\mathbf{v}_0 := \text{Lift}_K^0(\tilde{\mathbf{v}}_0)$. // $\mathbf{v}_0 = \langle \hat{\mathbf{d}}_1 \otimes \hat{\phi}_1 \rangle \in \mathbb{Z}_q^{tn^2\ell_q}$
2. for $j = \{1, \dots, r-2\}$,
 - (a) Reshape $(\mathbf{v}_{j-1}^1 \dots \mathbf{v}_{j-1}^{t^{j-1}})$ into t^j blocks $\hat{\mathbf{v}}_j^1, \dots, \hat{\mathbf{v}}_j^{t^j}$, where $\hat{\mathbf{v}}_j^i \in \mathbb{Z}_q^{n^2\ell_q}$
 - (b) Compute $\tilde{\mathbf{v}}_j^i := \lceil \text{hOTE}_n.\text{De}_A^{\text{raw}}(\text{pk}_{\text{hOTE}_n}, E_j, \psi_j^i) + \mathbf{P}' \cdot \hat{\mathbf{v}}_j^i \rceil_p \in \mathbb{Z}_p^{tn^2\ell_q}$, for $i \in \{1, \dots, t^j\}$.
 - (c) Lift $\mathbf{v}_j^i := \text{Lift}_K^0(\tilde{\mathbf{v}}_j^i)$, for $i \in \{1, \dots, t^j\}$ // $\mathbf{v}_j^i = \langle \hat{\mathbf{d}}_{j+1}^i \otimes \hat{\phi}_{j+1} \rangle \in \mathbb{Z}_q^{tn^2\ell_q}$
3. Reshape $(\mathbf{v}_{r-2}^1 \dots \mathbf{v}_{r-2}^{t^{r-2}})$ into t^{r-1} blocks $\hat{\mathbf{v}}_{r-1}^1, \dots, \hat{\mathbf{v}}_{r-1}^{t^{r-1}}$, where $\hat{\mathbf{v}}_{r-1}^i \in \mathbb{Z}_q^{n^2\ell_q}$.
4. Compute $\mathbf{v}_{r-1}^i := \lceil \text{hOTE}_\ell.\text{De}_A^{\text{raw}}(\text{pk}_{\text{hOTE}_\ell}, E_{r-1}, \psi_{r-1}^i) + \mathbf{P} \cdot \hat{\mathbf{v}}_{r-1}^i \rceil_p$, for $i \in \{1, \dots, t^{r-1}\}$ // $\mathbf{v}_{r-1}^i = \langle \mathbf{x}_i \otimes \mathbf{y} \rangle \in \mathbb{Z}_p^{tn\ell}$

Concatenate vertically $\mathbf{v} := (\mathbf{v}_{r-1}^1, \dots, \mathbf{v}_{r-1}^{t^{r-1}}) \in \mathbb{Z}_p^{m\ell}$ and return $\mathbf{v}$.

$\text{De}_B(\text{pk}, \mathbf{d}, \phi)$: Parse ϕ as ϕ_0 .

1. Compute $\tilde{\mathbf{w}}_0 := \text{OTE.De}_B(\text{pk}_{\text{OTE}}, \mathbf{d}, \phi_0) \in \mathbb{Z}_q^{tn^2\ell_p}$. Lift $\mathbf{w}_0 := \text{Lift}_K^1(\tilde{\mathbf{w}}_0)$. // $\mathbf{w}_0 = \langle \hat{\mathbf{d}}_1^1 \otimes \hat{\phi}_1 \rangle \in \mathbb{Z}_q^{tn^2\ell_q}$
2. for $j = \{1, \dots, r-2\}$, compute:
 - (a) Reshape $(\mathbf{w}_{j-1}^1 \dots \mathbf{w}_{j-1}^{t^{j-1}})$ into t^j blocks $\hat{\mathbf{w}}_j^1, \dots, \hat{\mathbf{w}}_j^{t^j}$, where $\hat{\mathbf{w}}_j^i \in \mathbb{Z}_q^{n^2\ell_q}$.
 - (b) Compute $\tilde{\mathbf{w}}_j^i := \lceil \mathbf{P}' \cdot \hat{\mathbf{w}}_j^i \rceil_p \in \mathbb{Z}_p^{tn^2\ell_q}$, for $i \in \{1, \dots, t^j\}$.
 - (c) Lift $\mathbf{w}_j^i := \text{Lift}_K^1(\tilde{\mathbf{w}}_j^i)$, for $i \in \{1, \dots, t^j\}$. // $\mathbf{w}_j^i = \langle \hat{\mathbf{d}}_{j+1}^i \otimes \hat{\phi}_{j+1} \rangle$
3. Reshape $(\mathbf{w}_{r-2}^1 \dots \mathbf{w}_{r-2}^{t^{r-2}})$ into t^{r-1} blocks $\hat{\mathbf{w}}_{r-1}^1, \dots, \hat{\mathbf{w}}_{r-1}^{t^{r-1}}$, where $\hat{\mathbf{w}}_{r-1}^i \in \mathbb{Z}_q^{n^2\ell_q}$.
4. Compute $\mathbf{w}_{r-1}^i := \lceil \mathbf{P} \cdot \hat{\mathbf{w}}_{r-1}^i \rceil_p$, for $i \in \{1, \dots, t^{r-1}\}$. // $\mathbf{w}_{r-1}^i = \langle \mathbf{x}^i \otimes \mathbf{y} \rangle \in \mathbb{Z}_q^{tn\ell}$

Concatenate vertically $\mathbf{w} := (\mathbf{w}_{r-1}^1 \dots \mathbf{w}_{r-1}^{t^{r-1}}) \in \mathbb{Z}_q^{m\ell}$ and return $\mathbf{w}$.

Figure 14: Our perfectly correct, succinct and hasher-private NIOTE Part II.

Proof. By construction, $\text{De}_B^{\text{Bits}}(\text{pk}, \hat{\mathbf{d}}, \phi = (\hat{\mathbf{s}})) = -(\mathbf{I}_{m\ell} \otimes \hat{\mathbf{s}}^\top \mathbf{G}_k^\top) \mathbf{B}^\top \mathbf{G}_n \hat{\mathbf{d}}$. By the linearization identity (see, e.g., Section 2 or [AMR25, Lemma 2.1]),

$$\begin{aligned}
 -(\mathbf{I}_{m\ell} \otimes \hat{\mathbf{s}}^\top \mathbf{G}_k^\top) \mathbf{B}^\top \mathbf{G}_n \hat{\mathbf{d}} &= -\text{Lin}(\mathbf{B}^\top \mathbf{G}_n) (\hat{\mathbf{d}} \otimes (\hat{\mathbf{s}}^\top \mathbf{G}_k^\top)^\top) \\
 &= -\text{Lin}(\mathbf{B}^\top \mathbf{G}_n) (\hat{\mathbf{d}} \otimes (\mathbf{G}_k \hat{\mathbf{s}})) \\
 &= -\text{Lin}(\mathbf{B}^\top \mathbf{G}_n) (\mathbf{I}_{nk\ell_q^2} \otimes \mathbf{g}_q^\top) (\hat{\mathbf{d}} \otimes \hat{\mathbf{s}}) = \mathbf{P} \cdot (\hat{\mathbf{d}} \otimes \hat{\mathbf{s}}).
 \end{aligned}$$

□

We also recall the following lemma about lifting (with a proof in [BKS19]).

Lemma 8 (Lifting the Modulus of Shares [BKS19]). *Let p be even with $p \geq \lambda^{\omega(1)}$. Let $m \in \mathbb{Z}$ with $|m| \leq B$ and $B < p/2$. Sample $z_0 \leftarrow \mathbb{Z}_p$ uniformly and set $z_1 = m - z_0 \bmod p$. Then*

$$\text{center}_p(z_0) + \text{center}(z_1) = m$$

with probability at least $1 - \frac{B+1}{p}$ over z_0 .

We will make use of the following observation that $\text{Lift}_K^0(\mathbf{x}) + \text{Lift}_K^1(\mathbf{y})$ lifts a secret sharing of $\mathbf{x} + \mathbf{y} \pmod{p}$ into a sharing mod q .

Corollary 2 (Integer-Correct Lifting of Small Sums). *Let $B < p/2$. Let $\tilde{\mathbf{x}}, \tilde{\mathbf{y}} \in \mathbb{Z}_p^n$ satisfy $\|\tilde{\mathbf{x}} + \tilde{\mathbf{y}}\|_\infty \leq B$. Let $\mathbf{X} := \text{Lift}_K^0(\tilde{\mathbf{x}}) \in \mathbb{Z}_q^n$ and $\mathbf{Y} := \text{Lift}_K^1(\tilde{\mathbf{y}}) \in \mathbb{Z}_q^n$. Then, over the choice of the masks $F_K(\text{cnt})$ (equivalently over K),*

$$\mathbf{X} + \mathbf{Y} \bmod q = \tilde{\mathbf{x}} + \tilde{\mathbf{y}} \bmod p$$

holds coordinate-wise with probability at least $1 - n \cdot \frac{B+1}{p}$.

Proof. Consider one coordinate and write $r := F_K(\text{cnt}) \in \mathbb{Z}_p$. Let $m := \tilde{x} + \tilde{y} \bmod p$, so $|m| \leq B$. Define $z_0 := \tilde{x} - r \in \mathbb{Z}_p$ and $z_1 := \tilde{y} + r \in \mathbb{Z}_p$, so $z_0 + z_1 \equiv m \pmod{p}$. Since r is uniform in $\mathbb{Z}_p$, z_0 is uniform in $\mathbb{Z}_p$ and (z_0, z_1) is a uniformly random additive sharing of m . By Lemma 8,

$$\text{center}(z_0) + \text{center}(z_1) = m$$

with probability at least $1 - \frac{B+1}{p}$. This implies $\text{center}(z_0) \geq \text{center}(z_1)$ with the same probability. Let $\hat{z}_0 = \text{center}(z_0)$, and $\hat{z}_1 = \text{center}(z_1)$. Note further that

$$\begin{aligned} \text{uncenter}_q(\hat{z}_0) + \text{uncenter}_q(\hat{z}_1) \bmod q \\ = \hat{z}_0 + c_0 q + \hat{z}_1 + c_1 q \bmod q = m, \end{aligned}$$

since $\hat{z}_\sigma + c_\sigma q < q$, for $\sigma \in \{0, 1\}$ by definition of uncenter , and $\hat{z}_0 + \hat{z}_1 = m$ over the integers. The definition of Lift and the fact that $\hat{z}_0 \leq p/2 < q/2$ and $p \mid q$, together with a union bound over n coordinates give the claim. $\square$

Correctness.

Theorem 4 (Correctness of the NIOTE in Figures 13 and 14). *Let integers p and q be chosen s.t. $p \geq \lambda^{\omega(1)}$ and $p/q \leq \lambda^{-\omega(1)}$, then*

$$\mathbf{v} + \mathbf{w} \bmod p = \mathbf{x} \otimes \mathbf{y}$$

with probability at least $1 - \lambda^{-\omega(1)}$ in Figures 13 and 14.

Proof. We show correctness of outputs at different levels in the tree, starting from the root.

Root. We show that the root output $\mathbf{v}_0$ and $\mathbf{w}_0$ are additive shares of the tensor product of bit-decomposed digest and bit-decomposed encoder secret of layer 1, i.e.,

$$\mathbf{v}_0 + \mathbf{w}_0 \bmod q = \hat{\mathbf{d}}_1 \otimes \hat{\phi}_1 \bmod p. \quad (5)$$

Note that,

$$\begin{aligned} \mathbf{v}_0 + \mathbf{w}_0 \bmod q \\ &= \text{Lift}_K^0(\tilde{\mathbf{v}}_0) + \text{Lift}_K^1(\tilde{\mathbf{w}}_0) \bmod q \\ &= \tilde{\mathbf{v}}_0 + \tilde{\mathbf{w}}_0 \bmod p && \text{(by Corollary 2, with } B = 1) \\ &= \text{OTE.De}_A(\text{pk}_{\text{OTE}}, E_0, \psi_0) + \text{OTE.De}_B(\text{pk}_{\text{OTE}}, \mathbf{d}, \phi_0) \bmod p \\ &= \hat{\mathbf{d}}_1 \otimes \hat{\phi}_1 \bmod p, \end{aligned}$$

by construction and correctness of the OTE with probability at least $1 - (2tn^2\ell_q/p)$, from the lifting since each coordinate of $\hat{\mathbf{d}}_1 \otimes \hat{\phi}_1$ lies in $\{0, 1\}$.

Inner Level. We show the following invariant of the output $\mathbf{v}_j$ and $\mathbf{w}_j$ for each inner level j in the tree in the following lemma.

Lemma 9. Let $m = t^r n$, then for any $j \in \{0, \dots, r-2\}$,

$$\mathbf{v}_j + \mathbf{w}_j \mod q = (\hat{\mathbf{d}}_{j+1} \otimes \hat{\phi}_{j+1}) \mod p,$$

with probability at least $1 - (2^{t^{j+1}} n^2 \ell_q / p + t^{j+1} n^2 ((n+1)tnB + 1)p/q)$, where $\mathbf{v}_j \in \mathbb{Z}_q^{n^{2t^{j+1}} \ell_q}$ and $\mathbf{w}_j \in \mathbb{Z}_q^{n^{2t^{j+1}} \ell_q}$ denote the vertical concatenation of all $\mathbf{v}_j^i$ and $\mathbf{w}_j^i$, for $i \in \{1, \dots, t^j\}$, respectively. $\hat{\mathbf{d}}_{j+1} \in \mathbb{Z}_2^{nt^{j+1} \ell_q}$ is defined as in Figure 13.

Proof. We prove by induction over j . For $j = 0$, we have proven the statement already in Equation (5). Assume the statement holds for $j-1 < r-2$, then, for all $i \in \{1, \dots, t^j\}$,

$$\begin{aligned} & \mathbf{v}_j^i + \mathbf{w}_j^i \mod q \\ &= \text{Lift}_K^0(\tilde{\mathbf{v}}_j^i) + \text{Lift}_K^1(\tilde{\mathbf{w}}_j^i) \mod q \\ &= \tilde{\mathbf{v}}_j^i + \tilde{\mathbf{w}}_j^i \mod p \quad (\text{see Corollary 2}) \\ &= \left[\text{hOTE}_n \cdot \text{De}_A^{\text{raw}}(\text{pk}_{\text{hOTE}_n}, E_j, \psi_j^i) + \mathbf{P}' \cdot \hat{\mathbf{v}}_j^i \right]_p + \left[\mathbf{P}' \cdot \hat{\mathbf{w}}_j^i \right]_p \mod p \\ &= \text{hOTE}_n \cdot \text{De}_A(\text{pk}_{\text{hOTE}_n}, E_j, \psi_j^i) + \mathbf{P}' \cdot (\hat{\mathbf{v}}_j^i + \hat{\mathbf{w}}_j^i) \mod p \quad (\text{Lemma 5}) \end{aligned}$$

Note that $\hat{\mathbf{v}}_j^i, \hat{\mathbf{w}}_j^i \in \mathbb{Z}_q^{n^{2\ell_q}}$. By assumption, we have $\mathbf{v}_{j-1} + \mathbf{w}_{j-1} \mod q = \hat{\mathbf{d}}_j \otimes \hat{\phi}_j \mod p$. Reshaping $\mathbf{v}_{j-1} + \mathbf{w}_{j-1}$ into t^j blocks yields

$$\begin{pmatrix} \hat{\mathbf{v}}_{j-1}^1 \\ \vdots \\ \hat{\mathbf{v}}_{j-1}^{t^j} \end{pmatrix} + \begin{pmatrix} \hat{\mathbf{w}}_{j-1}^1 \\ \vdots \\ \hat{\mathbf{w}}_{j-1}^{t^j} \end{pmatrix} = \begin{pmatrix} \text{Bits}(\mathbf{h}_j^1) \\ \vdots \\ \text{Bits}(\mathbf{h}_j^{t^j}) \end{pmatrix} \otimes \hat{\phi}_j.$$

Thus $\hat{\mathbf{v}}_j^i + \hat{\mathbf{w}}_j^i \mod q = \text{Bits}(\mathbf{h}_j^i) \otimes \hat{\phi}_j^i \mod p$, for $i \in \{1, \dots, t^j\}$. We continue,

$$\begin{aligned} & \text{hOTE}_n \cdot \text{De}_A(\text{pk}_{\text{hOTE}_n}, E_j, \psi_j^i) + \mathbf{P}' \cdot (\hat{\mathbf{v}}_j^i + \hat{\mathbf{w}}_j^i) \mod p \\ &= \text{hOTE}_n \cdot \text{De}_A(\text{pk}_{\text{hOTE}_n}, E_j, \psi_j^i) + \mathbf{P}' \cdot (\text{Bits}(\mathbf{h}_j^i) \otimes \hat{\phi}_j^i) \mod p \\ &= \hat{\mathbf{d}}_j^i \otimes \hat{\phi}_{j+1}^i \mod p \quad (\text{Lemma 7}) \end{aligned}$$

as required. $\square$

Leaf Level. To complete the correctness claim, observe that, for $i \in \{1, \dots, t^{r-1}\}$,

$$\begin{aligned} & \mathbf{v}_{r-1}^i + \mathbf{w}_{r-1}^i \mod q \\ &= \left[\text{hOTE}_\ell \cdot \text{De}_A^{\text{raw}}(\text{pk}_{\text{hOTE}_\ell}, E_{r-1}, \psi_{r-1}^i) + \mathbf{P} \cdot \hat{\mathbf{v}}_{r-1}^i \right]_p + \left[\mathbf{P} \cdot \hat{\mathbf{w}}_{r-1}^i \right]_p \mod p \\ &= \text{hOTE}_\ell \cdot \text{De}_A(\text{pk}_{\text{hOTE}_\ell}, E_{r-1}, \psi_{r-1}^i) + \mathbf{P} \cdot (\hat{\mathbf{v}}_{r-1}^i + \hat{\mathbf{w}}_{r-1}^i) \quad (\text{Lemma 5}) \end{aligned}$$

From Lemma 9, we have $\mathbf{v}_{r-2} + \mathbf{w}_{r-2} = \hat{\mathbf{d}}_{r-1} \otimes \hat{\phi}_{r-1}$, and by appropriate reshaping, we obtain $\hat{\mathbf{v}}_{r-1}^i + \hat{\mathbf{w}}_{r-1}^i = \text{Bits}(\mathbf{h}_{r-1}^i) \otimes \hat{\phi}_{r-1}^i$, and thus

$$\begin{aligned} & \text{hOTE}_\ell \cdot \text{De}_A(\text{pk}_{\text{hOTE}_\ell}, E_{r-1}, \psi_{r-1}^i) + \mathbf{P} \cdot (\hat{\mathbf{v}}_{r-1}^i + \hat{\mathbf{w}}_{r-1}^i) \mod p \\ &= \text{hOTE}_\ell \cdot \text{De}_A(\text{pk}_{\text{hOTE}_\ell}, E_{r-1}, \psi_{r-1}^i) + \mathbf{P} \cdot (\text{Bits}(\mathbf{h}_{r-1}^i) \otimes \hat{\phi}_{r-1}^i) \mod p \\ &= \mathbf{x}^i \otimes \mathbf{y} \mod p \quad (\text{Lemma 7}) \end{aligned}$$

Finally note that vertically stacking $\mathbf{x}^i \otimes \mathbf{y}$ yields the desired $\mathbf{x} \otimes \mathbf{y}$. The failure probability is at most $\frac{2n\ell_q m}{(t-1) \cdot p}$ from lifting and at most $(n\ell_q m \cdot (n+1)tn\ell_q B) \frac{p}{q(t-1)} + (m\ell \cdot \ell_q^2 tn(n+1)B) \frac{p}{q}$ from hOTE correctness. $\square$

B ORAM: Extensions and Proofs

In this section, we add supporting details for the ORAM application in Section 6.1. Throughout, we emphasize that we are only addressing the address-to-point-query subroutine that appears inside many DORAM and RAM-MPC protocols, and not proposing a complete ORAM construction.

We start with the proof of Proposition 4.

B.1 Proof of Proposition 4

Proof. We give explicit read and write protocols and prove correctness.

WRITE. Assume the memory is additively shared as $\mathbf{x} = \sum_i \mathbf{x}^i$ and the parties hold additive shares of the address bits and of a value $\beta = (\beta_1, \dots, \beta_w) \in \mathbb{Z}_p^w$. For each coordinate $t \in [w]$, run Π_{DPF}^B in parallel on the shared address and the shared scalar payload β_t to obtain shares $\{\mathbf{u}_{\alpha, \beta_t}^i\}_{i \in [N]}$ of $\mathbf{u}_{\alpha, \beta_t} \in \mathbb{Z}_p^L$. Each party P_i locally packs these w length- L vectors into a block-update share $\mathbf{U}_{\alpha, \beta}^i \in (\mathbb{Z}_p^w)^L$ by setting, for every $j \in [L]$,

$$\mathbf{U}_{\alpha, \beta}^i[j] := (\mathbf{u}_{\alpha, \beta_1}^i[j], \dots, \mathbf{u}_{\alpha, \beta_w}^i[j]) \in \mathbb{Z}_p^w.$$

Then $\sum_i \mathbf{U}_{\alpha, \beta}^i = \mathbf{U}_{\alpha, \beta}$, because for each j and each t , $\sum_i \mathbf{u}_{\alpha, \beta_t}^i[j] = \mathbf{u}_{\alpha, \beta_t}[j] = \mathbf{u}_{\alpha, 1}[j] \cdot \beta_t$. Finally, each party updates its local memory share by

$$\mathbf{x}^i \leftarrow \mathbf{x}^i + \mathbf{U}_{\alpha, \beta}^i,$$

component-wise in $(\mathbb{Z}_p^w)^L$. Correctness is immediate by linearity. No further interaction is required after the Π_{DPF}^B calls.

READ. We distinguish public and shared memory.

Public memory. Let $\mathbf{x}$ be public and assume the parties hold additive shares of the address bits.

1. Run Π_{DPF}^B with payload $\beta = 1$ to obtain shares $\{\mathbf{e}_\alpha^i\}_i$ of $\mathbf{u}_{\alpha, 1}$.
2. Each party outputs the block $\mathbf{y}^i = \sum_{j=0}^{L-1} \mathbf{e}_\alpha^i[j] \cdot \mathbf{x}[j] \in \mathbb{Z}_p^w$.

Correctness follows by:

$$\sum_i \mathbf{y}^i = \sum_i \sum_j \mathbf{e}_\alpha^i[j] \cdot \mathbf{x}[j] = \sum_j \left(\sum_i \mathbf{e}_\alpha^i[j] \right) \cdot \mathbf{x}[j] = \sum_j \mathbf{u}_{\alpha, 1}[j] \cdot \mathbf{x}[j] = \mathbf{x}[\alpha].$$

This requires no multiplication since $\mathbf{x}$ is public.

Shared memory. If the memory is additively shared as $\mathbf{x} = \sum_i \mathbf{x}^i$, then the parties proceed as follows.

1. Run Π_{DPF}^B with payload $\beta = 1$ to obtain shares $\{\mathbf{e}_\alpha^i\}_i$ of $\mathbf{u}_{\alpha, 1}$.
2. Define the (secret) blockwise inner product $\mathbf{y} = \sum_{j=0}^{L-1} \mathbf{u}_{\alpha, 1}[j] \cdot \mathbf{x}[j] \in \mathbb{Z}_p^w$. To obtain additive shares of $\mathbf{y}$, the parties compute, for each $j \in [L]$, additive shares of the product $\langle \mathbf{z}_j \rangle = \langle \mathbf{e}_\alpha[j] \rangle \cdot \langle \mathbf{x}[j] \rangle \in \mathbb{Z}_p^w$, using standard multiplication techniques (e.g., Beaver triples). This is w parallel scalar multiplications per j , and all j can be processed in parallel.

3. Each party outputs $\mathbf{y}^i := \sum_{j=0}^{L-1} \mathbf{z}_j^i \in \mathbb{Z}_p^w$.

Correctness holds because $\sum_i \mathbf{z}_j^i = \mathbf{e}_\alpha[j] \cdot \mathbf{x}[j]$ for each j , and then

$$\sum_i \mathbf{y}^i = \sum_i \sum_j \mathbf{z}_j^i = \sum_j \left(\sum_i \mathbf{z}_j^i \right) = \sum_j \mathbf{u}_{\alpha,1}[j] \cdot \mathbf{x}[j] = \mathbf{x}[\alpha].$$

The only component whose depth depends on L is generating the selector shares from the shared address, which is exactly the $\mathcal{O}(\log_2 b + \log_B L)$ -round protocol Π_{DPF}^B . The remaining read steps use (i) local linear operations and (ii) at most one batched multiplication layer, whose round complexity depends on the underlying MPC protocol and is typically constant in preprocessing-based MPC. In particular, when $b = \ell$, the round complexity of Π_{DPF}^B is $\mathcal{O}(\log \log L)$. $\square$

B.2 Batched Address Expansion

As mentioned before, in many ORAM and DORAM constructions, physical memory is partitioned into buckets of a fixed public size Bkt , typically a small power of two, e.g., $\text{Bkt} \in \{16, 32, 64\}$. Bucket $\tau \in [L/\text{Bkt}]$ contains the Bkt consecutive physical indices $\{\tau\text{Bkt}, \tau\text{Bkt}+1, \dots, \tau\text{Bkt}+(\text{Bkt}-1)\} \subseteq [L]$. ORAM protocols often access memory at bucket level: they read or write an entire bucket so that the transcript does not reveal which of the Bkt positions is the real target, while the remaining entries may contain other blocks or dummies.

Optimization Idea. Suppose an ORAM scheme must touch all Bkt offsets in a bucket whose ID τ is secret-shared, i.e., it needs unit vectors for the Bkt addresses $\tau\text{Bkt} + o$ for all $o \in [\text{Bkt}]$. A naive batched expansion would treat these as Bkt independent indices in $[L]$.

Instead, we note that all Bkt indices share the same bucket-ID prefix τ and differ only in the public offset o . We first expand only the bucket ID on the smaller domain $[L/\text{Bkt}]$: run Π_{DPF}^B on the $k = \ell - r$ high bits to obtain additive shares $\{\delta_i\}_{i \in [N]} \subseteq \mathbb{Z}_p^{L/\text{Bkt}}$ with $\sum_i \delta_i = \mathbf{u}_{\tau,1}$. Then, for each public offset $o \in [\text{Bkt}]$, each party locally outputs

$$\mathbf{u}^{(o),i} = \delta_i \otimes \mathbf{e}_o \in \mathbb{Z}_p^{(L/\text{Bkt}) \cdot \text{Bkt}} = \mathbb{Z}_p^L,$$

where $\mathbf{e}_o \in \mathbb{Z}_p^{\text{Bkt}}$ is the public standard basis vector at position o . By the tensor indexing rule $(\mathbf{a} \otimes \mathbf{b})[x \cdot \text{Bkt} + y] = \mathbf{a}[x] \cdot \mathbf{b}[y]$, we have $\sum_i \mathbf{u}^{(o),i} = \mathbf{u}_{\tau\text{Bkt}+o,1}$ for every o . The only interaction is the single prefix expansion on domain size L/Bkt , while the embedding step is purely local and uses only public offsets.

Tries and Prefix-Collision Leakage. For binary addresses of length ℓ , the prefix tree has one node per prefix $\mathbf{p} \in \{0,1\}^{\leq \ell}$ and a leaf for each full address. Given a batch of secret addresses $\alpha^{(1)}, \dots, \alpha^{(T)}$, the induced prefix tree is the minimal sub tree containing these T leaves. Its shape is determined by the *prefix-collision pattern* of the batch: for each depth d , which indices share the same d -bit prefix.

A tempting optimization is to execute NIOTE only on nodes in this induced prefix tree. However, the number of executed nodes per depth, that is, the transcript size and instance structure, would then depend on the prefix-collision pattern among the *secret* addresses. In an ORAM setting this is not allowed, as it leaks nontrivial information about the hidden access pattern, e.g., whether two secret indices share a long prefix.

ORAM-Compatible Leakage Requirement. One possibility is to restrict to structure-aware batches, where the prefix-sharing structure is fixed by the public ORAM control flow. Formally, for each ORAM access primitive we define a public *pattern* that determines a fixed schedule of subqueries and NIOTE invocations (independent of the secret indices). We allow the transcript to depend only on

$$\text{Leak}_{\text{ORAM}} := (N, \ell, T, \text{pattern}).$$

Concretely this means that once $(N, \ell, T, \text{pattern})$ is fixed, the set of NIOTE instances executed, their input/output dimensions, and the message flow are input-independent.

In what follows we give two concrete ORAM constructions that exploit prefix sharing while satisfying this requirement.

Let $L = 2^\ell$ and let $\alpha \in [L] = \{0, \dots, L-1\}$ have binary expansion $\alpha = \sum_{t=0}^{\ell-1} \alpha_t 2^t$ with bits $\alpha_t \in \{0, 1\} \subseteq \mathbb{Z}_p$. We use selector vectors $\mathbf{s}_t(\alpha_t) = (1 - \alpha_t, \alpha_t) \in \mathbb{Z}_p^2$ such that

$$\mathbf{u}_{\alpha,1} = \mathbf{s}_{\ell-1}(\alpha_{\ell-1}) \otimes \dots \otimes \mathbf{s}_0(\alpha_0) \in \mathbb{Z}_p^{2^\ell}.$$

We assume an N -party NIOTE primitive $\text{NIOTE} = (\text{Setup}, \text{En}, \text{De})$ that, on additive shares of $(\mathbf{x}, \mathbf{y})$, outputs additive shares summing to $(\sum_i x_i) \otimes (\sum_i y_i)$.

Full-Bucket Multi-Offset Queries. Every address $\alpha \in [L]$ has a unique decomposition $\alpha = \tau \cdot \text{Bkt} + o$ with $\tau \in [2^k]$ and $o \in [2^r]$. In this case, we assume that the bucket ID τ is secret-shared among the parties, while the offsets $o^{(1)}, \dots, o^{(T)} \in [2^r]$ are public constants, typically $T = \text{Bkt}$ and $o^{(\iota)} = \iota - 1$. The protocol proceeds as follows.

1. **PREFIX EXPANSION (INTERACTIVE).** Run our address-to-point compiler on the k high-order bits of τ , obtaining additive shares $\{\delta_i\}_{i \in [N]} \subseteq \mathbb{Z}_p^{2^k}$ such that $\sum_{i=1}^N \delta_i = \mathbf{u}_{\tau,1} \in \mathbb{Z}_p^{2^k}$. This is the only step that invokes NIOTE.
2. **OFFSET EMBEDDING (LOCAL, FOR EACH QUERY).** For each public offset $o^{(\iota)} \in [2^r]$, let $\mathbf{e}_{o^{(\iota)}} \in \mathbb{Z}_p^{2^r}$ denote the public standard basis vector with a 1 at coordinate $o^{(\iota)}$. Each party outputs

$$\mathbf{u}^{(\iota),i} := \delta_i \otimes \mathbf{e}_{o^{(\iota)}} \in \mathbb{Z}_p^{2^k \cdot 2^r} = \mathbb{Z}_p^{2^\ell} = \mathbb{Z}_p^L.$$

Equivalently, $\mathbf{u}^{(\iota),i}[x \cdot 2^r + y] = \delta_i[x]$ if $y = o^{(\iota)}$ and 0 otherwise.

Correctness. For every $\iota \in [T]$,

$$\sum_{i=1}^N \mathbf{u}^{(\iota),i} = \left(\sum_i \delta_i \right) \otimes \mathbf{e}_{o^{(\iota)}} = \mathbf{u}_{\tau,1} \otimes \mathbf{e}_{o^{(\iota)}} = \mathbf{u}_{\tau \cdot 2^r + o^{(\iota)},1}.$$

The transcript consists of a single execution of the compiler on a domain of size $2^k = L/\text{Bkt}$. Hence the number, dimensions, and round structure of all NIOTE invocations depend only on (N, k) (equivalently on (N, ℓ, Bkt)) and not on the secret bucket ID τ . The embedding step is purely local.

Whole-Path Access. Tree-based ORAMs access one bucket per level along a root-to-leaf path. If the accessed leaf label $\alpha \in [L]$ is represented by secret-shared bits, then the bucket touched at depth d is determined by the length- d prefix of α 's binary representation. Our goal is to generate, for every depth $d \in \{0, \dots, \ell\}$, an additive sharing of the unit vector selecting that depth- d bucket. For $d \in \{0, \dots, \ell\}$, let $\text{pref}_d(\alpha) = \lfloor \alpha / 2^{\ell-d} \rfloor \in [2^d]$, i.e., the integer encoded by the d most-significant bits of α . We aim to output additive shares of $U^{(d)} = \mathbf{u}_{\text{pref}_d(\alpha),1} \in \mathbb{Z}_p^{2^d}$ ($d = 0, \dots, \ell$), without leaking anything about α beyond the fact that this is a “path” access.

Let $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{\ell-1}$ denote the selector vectors for α 's bits in MSB-to-LSB order: $\mathbf{a}_0 = \mathbf{s}_{\ell-1}(\alpha_{\ell-1})$, $\mathbf{a}_1 = \mathbf{s}_{\ell-2}(\alpha_{\ell-2})$, $\dots$, $\mathbf{a}_{\ell-1} = \mathbf{s}_0(\alpha_0)$. By the unit-vector identity, for each $d \geq 1$ we have $U^{(d)} = \mathbf{a}_0 \otimes \mathbf{a}_1 \otimes \dots \otimes \mathbf{a}_{d-1} \in \mathbb{Z}_p^{2^d}$, and $U^{(0)} = (1) \in \mathbb{Z}_p$. Thus, generating the whole path reduces to computing all prefix tensor-products of the sequence $(\mathbf{a}_0, \dots, \mathbf{a}_{\ell-1})$.

Let $n = 2^{\lceil \log_2(\ell+1) \rceil}$ and pad the list with identity elements so we have n leaves: for $j < \ell$ set $A_j = \mathbf{a}_j$ and for $\ell \leq j < n$ set $A_j := (1) \in \mathbb{Z}_p^1$. Each party locally holds additive shares of every A_j .

We compute the prefix products using a standard Blleloch scan [Ble89, Ble90], where every combine step is implemented by one NIOTE call on the two shared child values. Concretely, the protocol proceeds as follows.

1. UP-SWEEP. Build a complete binary tree over the n leaves. At height $h = 0, 1, \dots, \log_2 n - 1$, for every node representing a segment of length 2^{h+1} , compute additive shares of the tensor product of its left and right child segment-values. At the end of the up-sweep, the root holds additive shares of the full product $A_0 \otimes \dots \otimes A_{n-1}$.
2. DOWN-SWEEP (PROPAGATE PREFIXES). Initialize the root prefix to the identity (1). For each internal node with left segment value L and right segment value R , and with parent prefix value P , set:

$$\begin{aligned} \text{prefix(left)} &:= P, \\ \text{prefix(right)} &:= P \otimes L, \end{aligned}$$

where the tensoring $P \otimes L$ is computed by one NIOTE call on shared inputs. After this pass, each leaf position j holds additive shares of the exclusive prefix $P_j := A_0 \otimes \dots \otimes A_{j-1}$.

3. OUTPUT. For every $d \in \{0, \dots, \ell\}$ output $U^{(d)} := P_d$. For $d = 0$ this is the identity, for $d \geq 1$, this equals $\mathbf{a}_0 \otimes \dots \otimes \mathbf{a}_{d-1}$ since the padding identities contribute nothing.

Lemma 10 (Correctness). *For every $d \in \{0, \dots, \ell\}$, the protocol outputs additive shares $\{U_i^{(d)}\}_{i \in [N]}$ such that $\sum_i U_i^{(d)} = \mathbf{u}_{\text{pref}_d(\alpha),1}$.*

Proof. By correctness of NIOTE, every combine step outputs additive shares of the tensor product of the two reconstructed child values. Since $\otimes$ is associative, the Blleloch scan invariants [Ble89, Ble90] imply that each leaf d receives additive shares of $P_d = A_0 \otimes \dots \otimes A_{d-1}$. For $d \leq \ell$, this equals $\mathbf{a}_0 \otimes \dots \otimes \mathbf{a}_{d-1}$ because $A_j = \mathbf{a}_j$ for $j < \ell$. Finally, by the unit-vector identity applied to the d MSB selectors, this tensor product equals $\mathbf{u}_{\text{pref}_d(\alpha),1} \in \mathbb{Z}_p^{2^d}$. $\square$

Lemma 11. *The full transcript, i.e., number of NIOTE calls, their input/output dimensions, and message flow, depends only on (N, ℓ) and the public tag pattern = path, and is independent of α .*

Proof. The scan executes a fixed complete-tree schedule for $n = 2^{\lceil \log_2(\ell+1) \rceil}$ leaves, hence a fixed set of NIOTE invocations. By NIOTE privacy, each NIOTE message is simulatable without knowledge of the parties' secret inputs. No branching depends on α . $\square$

The scan performs $(n - 1)$ tensorings in the up-sweep and $(n - 1)$ tensorings in the down-sweep, i.e., $2n - 2 \leq 4\ell + 2$ NIOTE calls total, in $2 \log_2 n = \mathcal{O}(\log \ell) = \mathcal{O}(\log \log L)$ synchronous rounds.