Non-Interactive Distributed Point Functions

Elette Boyle[1], Lalita Devadas[2], and Sacha Servan-Schreiber[2]

[1] NTT Research and Reichman University
[2] MIT

Abstract. Distributed Point Functions (DPFs) are a useful cryptographic primitive enabling a dealer to distribute short keys to two parties, such that the keys encode additive secret shares of a secret point function. However, in many applications of DPFs, no single dealer entity has full knowledge of the secret point function, necessitating the parties to run an interactive protocol to emulate the setup. Prior works have aimed to minimize complexity metrics of such distributed setup protocols, e.g., *round complexity*, while remaining black-box in the underlying cryptography.

We construct *Non-Interactive* DPFs (NIDPF), which have a one-round (*simultaneous-message*, semi-honest) setup protocol, removing the need for a trusted dealer. Specifically, our construction allows each party to publish a special "public key" to a public channel or bulletin board, where the public key encodes the party's secret function parameters. Using the public key of another party, any pair of parties can *locally* derive a DPF key for the point function parameterized by the two parties' joint inputs.

We realize NIDPF from an array of standard assumptions, including DCR, SXDH, QR, and LWE. Each party's public key is of size $O(N^{2/3})$, for point functions with a domain of size N, which leads to a sublinear communication setup protocol. The only prior approach to realizing such a non-interactive setup required using multi-key fully-homomorphic encryption or indistinguishability obfuscation.

As immediate applications of our construction, we obtain "public-key setup" protocols for several existing constructions of pseudorandom correlation generators and round-efficient protocols for secure comparisons.

Table of Contents

1 Introduction . . . 2
1.1 Our results . . . 3
1.2 Applications . . . 4
2 Technical Overview . . . 5
2.1 Building block: Non-interactive multiplication . . . 5
2.2 Overview of the NIDPF construction . . . 6
3 Preliminaries . . . 10
3.1 Notation . . . 10
3.2 Additive secret sharing . . . 10
3.3 Cryptographic assumptions . . . 11
3.4 The NIDLS framework . . . 11
3.5 Degree-2 secret-key HSS . . . 12
4 Non-Interactive Multiplication . . . 14
4.1 NIM with multiplicative output reconstruction . . . 15
4.2 Succinct NIM for matrix multiplication . . . 15
4.3 Constructions from group-based assumptions . . . 15
4.4 Constructions from lattice-based assumptions . . . 18
5 Non-Interactive DPF . . . 18
5.1 Emulating arithmetic modulo N . . . 21
5.2 NIDPF framework . . . 22
5.3 Random-payload instantiation from SXDH . . . 24
6 Generalization to Succinct Multi-Key HSS . . . 27
7 Homomorphic Secret Sharing . . . 32

1 Introduction

A point function, denoted by $P_{i,v}$, is a function that evaluates to a message v on input i, and evaluates to zero on all other inputs $j \neq i$ in its domain. A Distributed Point Function (DPF) [GI14, BGI15] allows a trusted dealer to distribute short keys to two parties, where the keys jointly encode a point function $P_{i,v}$ for parameters (i, v) chosen by the dealer. Individually, a DPF key does not reveal any information about the secret index i or message v to the party. However, using their key, each party can locally "evaluate" the point function on a public input x, to obtain an additive *secret share* of $y := P_{i,v}(x)$.

DPFs are the backbone of many useful primitives and protocols relating to multi-party computation (MPC). In particular, DPFs enable communication-efficient generation of correlated randomness in MPC protocols [BCGI18, SGRR19, BCG+19a, BCG+19b, BCG+20b, BCG+20a, YWL+20, AS22, BCG+22, BBC+24], can be used to instantiate distributed oblivious RAM [Ds17, VHG23], privacy-preserving machine learning [RTPB22, YJG+23, JGB+24], private database queries [WYG+17, DFL+20, DRPS22, SSLD22] and analytics [BBC+21, MPD+24, MST24, RZCGP24], and mixed-mode secure computation [BGI19, BCG+21].

However, in many of these applications, there is no trusted dealer that can generate and distribute the DPF keys to the parties. Instead, the trusted dealer is emulated by the parties via a distributed key generation protocol [Ds17, BGIK22, VSH22, VHG23], which the parties invoke to obtain their respective DPF keys. More concretely, in a distributed generation protocol, each party holds a *secret share* of the parameters (i, v). After invoking the protocol, the parties end up with DPF keys that correspond to the point function $P_{i,v}$, such that neither party learns the parameters (i, v) in the process.

Early approaches to distributed key generation simply used generic secure computation, resulting in protocols with at least two rounds of communication, while being non-black-box in the underlying cryptographic primitives. It was shown by Doerner and shelat [Ds17] how to achieve black-box distributed two-party key generation with logarithmically many communication rounds. Later, the DPF construction of Boyle et al. [BGIK22] admitted a 5-round black-box protocol. In both approaches, the

DPF key size was polylogarithmic in the domain size N. If one instead relaxes the key size to, e.g., to $N^{1/2}$, these approaches can yield black-box distributed generation protocols with round complexity as low as two sequential instances of 1-out-of-$N^{1/2}$ oblivious transfer, resulting in a small constant number of rounds.

At first glance, it is tempting to think that (folklore) lower-bounds from the MPC literature would set the minimum number of rounds required for a distributed DPF generation protocol to two. However, upon closer inspection, we observe that because the parties obtain a *key* (which in some DPF constructions can even be distributed pseudorandomly [BGI15, BGIK22]), a DPF generation protocol is *not* subjected to the two-round lower bound because each key can be efficiently simulated. In particular, we can hope to achieve a *non-interactive* generation protocol mimicking non-interactive key exchange protocols like Diffie–Hellman [DH76]. Indeed, spooky encryption [DHRW16] already gives such a protocol through the use of multi-key fully homomorphic encryption (FHE) or indistinguishability obfuscation ($i\mathcal{O}$). However, to date, this has been the *only* known approach to realizing a "non-interactive" protocol (a protocol where each party only needs to read the other party's public key to locally derive a joint DPF key).

1.1 Our results

In this paper, we put forth and study the notion of a "non-interactive" DPF, and demonstrate constructions from new assumptions. This is motivated by the search for (round-efficient) protocols for eliminating the dealer, that do not require heavy tools like multi-key FHE, and can be instantiated from an array of standard assumptions.

Non-Interactive DPFs. Our definition of a *non-interactive* DPF (NIDPF) enables two parties to locally (non-interactively) derive DPF keys by simply reading each other's public keys from a bulletin board. More generally, this model is captured by a one-round, simultaneous-message semi-honest protocol. A simultaneous-message communication pattern captures the interaction of non-interactive key exchange protocols like Diffie–Hellman: (1) two parties exchange messages simultaneously, then (2) any party can use another party's message to locally derive a joint output (key). Such a model of communication is highly desirable because the first message can be reused (i.e., the message of the first party can be reused indefinitely with many different parties) and the parties do *not* need to be online at the same time to participate.

The problem of generating DPF keys in a simultaneous-message protocol is much more challenging compared to key exchange. This is due to the fact that a DPF setup requires the *total communication* between parties (i.e., the size of the public keys) to be sublinear in the domain of the point function. This requirement is generally challenging to achieve—indeed, the only way we currently know of achieving such succinctness is via multi-key FHE [DHRW16]. Moreover, this connection to "multi-key"-like primitives is inherent, as we remark on later.

Constructing NIDPF. Our primary contribution is to show that, perhaps surprisingly, we can rely on simple cryptography and assumptions to achieve the sublinearity requirements. In particular, inspired by the recent work of Abram et al. [ARS24], we show that we can realize NIDPF schemes "directly," without going through heavier primitives like multi-key FHE. A NIDPF scheme immediately implies non-interactive key exchange, and thus public-key encryption, which eliminates the possibility of using only lightweight symmetric-key cryptography (e.g., one-way functions). However, we are able to realize NIDPFs from many standard assumptions, including the decisional composite residuosity (DCR) assumption, symmetric external Diffie–Hellman (SXDH) assumption, quadratic residuosity (QR) assumption, the enhanced Diffie–Hellman (EDDH) assumption in class groups, and the learning with errors (LWE) assumption. We summarize our results in Table 1 and Theorem 1.

Theorem 1 (Informal). *Let N be a domain size. There exists a Non-Interactive DPF (NIDPF) with a key size $O(N^{2/3})$ and evaluation time $O(N^{5/3})$ under either (1) the DCR assumption, (2) the QR assumption, (3) the EDDH assumption and the uniformity assumption in class groups, (4) the SXDH assumption, or (5) the LWE assumption with a superpolynomial-modulus-to-noise ratio. Here, $O(\cdot)$ hides polynomial factors in the security parameter.*

As an independent contribution, we define a new abstraction that we call *non-interactive multiplication*, which captures all existing "non-interactive" primitives from a recent line of work. In

	Assumption	**Transparent Setup**	**Key Size**
[DHRW16, XW23]	LWE / $i\mathcal{O}$+DDH	✓	$\log(N)$
This Work	LWE / RLWE	✓	$N^{2/3}$
	DCR / QR	✗	$N^{2/3}$
	SXDH* / Class Groups+EDDH	✓	$N^{2/3}$

Table 1: Summary of our instantiations of NIDPF with domains of size N. Constants and polynomial factors (in the security parameter) are ignored in the asymptotic key size for readability. See Section 3 for details on the cryptographic assumptions used. *The SXDH-based construction only supports random payloads (output messages).

particular, we identify a surprising (but rather obvious in retrospect) connection between our abstraction and Homomorphic Secret Sharing (HSS). This connection results in constructions of "succinct *multi-key* HSS" (restricted to a special class of computations) from a variety of assumptions, including DDH, DCR, and the EDDH assumption in class groups. To the best of our knowledge, the only prior approaches for such non-interactive computation required using multi-key FHE techniques [DHRW16, XW23]. More concretely, our abstraction allows us to adapt the recent result of Abram et al. [ARS24] constructing succinct HSS for "special RMS" programs to be non-interactive, albeit restricted to a slightly weaker class of functions. Specifically, unlike with standard HSS, our construction does not require a correlated setup between parties and only requires a common reference string. Moreover, the additional succinctness property allows one party to have a large input x while maintaining that the input share is succinct in the size of x. We summarize this generalization of our techniques in Theorem 2 and provide more details in Section 6.

Theorem 2 (Informal)**.** *Let* HSS *be an HSS scheme for the function class $\mathcal{F}$ and let $\mathcal{P}$ be the set of constant-degree polynomials. There exists a succinct, multi-key HSS scheme for computing functions of the form $P(x, f(y))$, where $P \in \mathcal{P}$ and $f \in \mathcal{F}$, one party has a (large) input x, and the other party has a (short) input y. Moreover, the total size of both parties' input shares is $o(|x|) + O(|y|)$, ignoring polynomial factors in the security parameter.*

1.2 Applications

We describe two immediate applications of our NIDPF construction. The primary application is replacing multi-round DPF setup protocols with a non-interactive "public key" setup. In particular, many applications of DPFs require two parties, each holding a secret share of an index $t \in [N]$, to generate DPF keys (through a secure setup protocol) that encode a point function parameterized by t. Concretely, Alice and Bob hold shares $t_A, t_B \in [N]$, such that $t_A + t_B = t \bmod N$, jointly generate DPF keys for the point function $P_{t,1}$. (We assume additive secret sharing of the index t, following [BGIK22, BBC+24]; some protocols also consider bit-wise XOR secret shares of t, however, applications typically require working with additive secret sharing, e.g., [BCG+20b, BCCD23, BBC+24].)

PCGs with a "public-key" setup. Pseudorandom Correlation Generators (PCGs) [BCGI18, BCG+19a, BCG+19b, SGRR19, BCG+20a, BCG+20b, BCG+22, AS22, BBC+24] are a cryptographic primitive enabling parties to generate long pseudorandom correlations given access to short correlated seeds. In particular, to jointly generate long correlations, it suffices for parties to first execute a secure computation protocol to jointly sample the short PCG seeds, and then *locally* expand them into a large number of pseudorandom correlations. PCG constructions exist for a variety of correlations, including oblivious transfer (OT), vector olivious linear evaluation (VOLE), and Beaver triple correlations, and make heavy use of DPFs. Indeed, the dominant cost of the setup protocol for these constructions is jointly generating DPF keys [SGRR19, BCG+20b, AS22, BBC+24].

Interestingly, a recent line of work [OSY21, BCM+24] has shown that when it comes to OT/VOLE correlations specifically, the parties do not need to engage in the initial interactive setup protocol. Instead, two parties can non-interactively derive a pair of seeds that enables them to expand their correlations locally. Such PCGs are said to have a "public-key setup" protocol, which follows the same non-interactive communication pattern we motivated in Section 1. However, to date, the only such "public-key PCG" constructions that exist are for the OT/VOLE correlation [OSY21, BCM+24]. It has remained an open problem to realize public-key PCGs for other correlation types (e.g., Beaver triple correlations), for which we have constructions of PCGs from a variety of standard assumptions but no corresponding public-key setup protocol.

By instantiating the DPF in existing PCG constructions (for further classes of correlations) with an NIDPF, it becomes possible to obtain a semi-honest "public-key setup" protocol for the PCG.

Mixed-mode secure computation in one round. Recent works on mixed-mode secure computation, beginning with the work of Boyle et al. [BCG+21], have demonstrated that the round and communication complexity of MPC protocols can be improved by using DPFs to help directly evaluate complex functions such as comparisons and equality of secret-shared values, without needing to express the computations as Boolean or arithmetic circuits.

For example, consider the case of securely computing secret shares of an equality predicate evaluated between a public threshold t and secret shared input x, in a two-party setting. The idea, at a high level, is to have a trusted dealer distribute DPF keys to two parties for the point function $P_{t+r,1}$ that evaluates to 1 on index $t + r$, where r is uniformly random in the domain. The dealer additionally distributes additive shares of r to the parties. The parties, holding additive shares of a value x (assumed to be in the domain of the point function), can publicly open the value $y := x + r$ by locally masking their shares of x with their shares of r and sending the result. Then, observe that by evaluating the DPF on input y, the parties obtain shares of 1 if and only if $x + r = t + r$, which is the case if and only if $x = t$.

Because the DPF is "one-time-use" due to the masking term r, the efficiency gains obtained by such a protocol depend heavily on the efficiency of the DPF setup protocol used by the parties to emulate the dealer. However, when using a NIDPF to act as the dealer, the parties can *simultaneously*: (1) choose their own share $\langle r\rangle_\sigma$ of the random mask r, (2) broadcast their masked share $\langle x\rangle_\sigma + \langle r\rangle_\sigma$, and (3) generate and send their NIDPF key message for the point function $P_{t+r,1}$. This already enables the parties to locally compute additive shares of the t-equality predicate on x, yielding a *single (simultaneous) round* protocol for the equality predicate computation.

2 Technical Overview

In this section, we provide a detailed technical overview of our NIDPF construction. The main building block we use in our construction is a novel abstraction we call non-interactive multiplication (NIM), which we overview in Section 2.1, and which we view as a contribution of independent interest. In Section 2.2, we provide an overview of our NIDPF construction.

2.1 Building block: Non-interactive multiplication

At a high level, a NIM allows two parties, Alice and Bob, each holding a ring element as input, to obtain secret shares of the multiplication of their inputs by exchanging one message simultaneously (or posting their message to a public bulletin board). The NIM abstraction captures several primitives recently introduced in various contexts. In particular, NIM directly implies *non-interactive* variants of OT [BM90], VOLE [OSY21, ADOS22, ARS24, BCM+24, CDD+24], and inner-products [CZ22], where parties obtain secret-shares of the computation by exchanging one message simultaneously.

The NIM abstraction also captures the case where Alice and Bob have *matrices* as inputs, and wish to compute secret shares of the matrix product. In particular, a NIM scheme for matrix multiplication allows Alice with a matrix $\mathbf{A}$ and Bob with a matrix $\mathbf{B}$ to compute additive secret shares of $\mathbf{AB}$. Surprisingly, a NIM scheme for matrix multiplication can have *sublinear communication* (in the size of one input matrix), using techniques developed in two recent works that build *succinct* VOLE [ARS24, BCM+24]. This succinct NIM variant is the main building block we use in Section 5 to construct NIDPFs. We obtain the following instantiations of succinct NIM for matrix multiplication:

Theorem 3 (Informal; Implicit in [ARS24, BCM+24])**.** *Let $\mathcal{R}$ be a ring and N, ℓ, m be integer parameters. For $\ell = N^{2/3}$ and $m = N^{1/3}$, there exists a succinct NIM scheme computing shares of* $\mathbf{AB}$ *with $O(N^{2/3})$ communication, for all matrices $\mathbf{A} \in \mathcal{R}^{\ell\times m}$ and $\mathbf{B} \in \mathcal{R}^{m\times m}$, if one of the following assumptions hold: (1) DCR, (2) QR, (3) an "enhanced" DDH assumption in class groups, (4) LWE with a superpolynomial modulus-to-noise ratio, or (5) SXDH in bilinear groups when the NIM output sharing is defined multiplicatively. In the above, $O(\cdot)$ hides polynomial factors in the security parameter.*

Our proof of Theorem 3 follows from ideas presented in the recent work of Abram et al. [ARS24] in their construction of "succinct" Homomorphic Secret Sharing (HSS) [BGI16] and a concurrent work of [BCM+24] constructing succinct non-interactive VOLE. In particular, their constructions internally use the ability to multiply a matrix $\mathbf{A}$ by another matrix $\mathbf{B} := \Delta \cdot \mathbf{I}$ (where Δ is a scalar and $\mathbf{I}$ is the identity matrix), with sublinear communication in the size of $\mathbf{A}$. However, in both these works, the primary goal was constructing a non-interactive VOLE scheme. Non-interactive VOLE, in and of itself, neither implies NIDPFs nor succinct NIM for matrix multiplication, and is overall a weaker primitive. In this paper, we show that their constructions not only generalize to *any* matrix $\mathbf{B}$ (of appropriate size)—while still preserving sublinear communication—but also use it as a building block to construct NIDPFs in Section 5.

Comparison to HSS. While at first glance, it may appear as though using HSS would be sufficient to construct NIM, there are subtle yet important distinctions between these primitives which make them very different. Concretely, we show in Section 4 that our NIM abstraction implies a form of (2-party) *multi-key* HSS, in an analogous sense to multi-key FHE, which is a stronger primitive compared to standard HSS.

In particular, the NIM abstraction has a universal setup that consists of a common reference string used by everyone, which enables us to realize a truly "non-interactive" (or multi-key) primitive, eliminating the requirement for multi-round setup protocols. In contrast, HSS [BGI16, BCG+17], including succinct HSS [ARS24], requires a *trusted setup process* to distribute evaluation keys to each party before any computation can be performed, and the setup cannot be used in a computation involving other parties. In a network with many parties, each *pair* of parties needs to generate a unique pair of HSS evaluation keys and can only compute over inputs assigned to them, resulting in quadratic communication overheads. Compare this with *multi-key* HSS [XW23, CDH+25] or spooky encryption [DHRW16], where any pair of parties can compute a function "on the fly" over their joint inputs and *without* needing to perform a joint setup process ahead of time to do so. As such, we view our NIM abstraction as a potential stepping-stone to uncovering new constructions for efficient MPC. Indeed, in Section 6, we generalize the ideas we use to realize our NIDPF constructions to realize a form of multi-key HSS for a restricted class of computations, which may prove to have additional applications.

2.2 Overview of the NIDPF construction

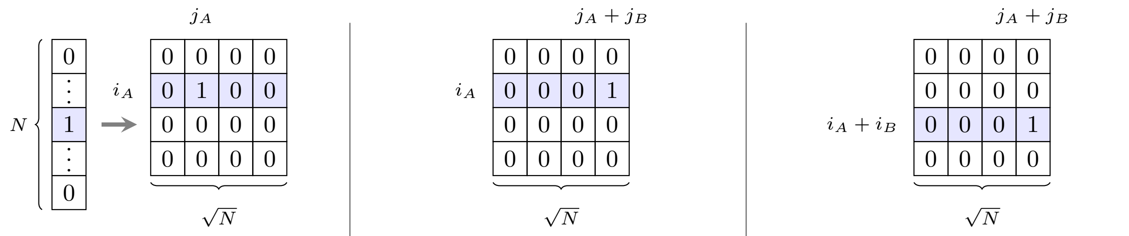
A NIDPF consists of a generation algorithm Gen and evaluation algorithm Eval. We let $\mathcal{R}$ be a finite ring and the message space of the NIDPF. A party Alice, with point function parameters (t_A, v_A), uses Gen to generate a public key pk_A and a secret key sk_A. Bob does the same with (t_B, v_B). Then, using the public key of the other party in conjunction with their own secret key, Alice and Bob can locally derive a DPF key encoding the point function with parameters $(t_A + t_B, v_A + v_B)$. Importantly, the public keys generated by Alice and Bob need to be *short*—sublinear in the truth-table size of the point function.

Similarly to some other DPF constructions (and all non-generic DPF key generation protocols) [CBM15, Ds17, BGIK22], our construction of NIDPF is tailored to the "full-domain evaluation" regime, where the parties obtain the output of the point function evaluated on all inputs in the domain. We let N denote the domain size of the point function, and view the point function evaluation on the entire domain (i.e., for every $x \in [N]$) as being a one-hot vector $\mathbf{u} \in \mathcal{R}^N$, where $\mathbf{u}[t] = 1$ and t is the special index.

For suitable choices of N, we can represent $x \in [N]$ by (i, j) where $i = x \pmod{\ell}$ and $j = x \pmod{m}$ for some coprime integers $\ell, m \in [N]$ such that $N = \ell \cdot m$. Such a mapping (*à la* Chinese Remainder Theorem) allows us to interpret the length-N vector $\mathbf{u}$ as a $\ell \times m$ matrix, while still preserving arithmetic modulo N via the residue number system. This places a restriction on N, which ideally we would like to avoid. In Section 5.1, we sketch an alternative approach that works for arbitrary (non-coprime) integers ℓ, m, but has a $2\times$ cost in efficiency.

At a high level, our approach to realizing our NIDPF construction is the following. Assume that Alice parses her index $t_A \in [N]$ as $t_A = (i_A, j_A) \in [\ell] \times [m]$ and Bob parses his index t_B as $(i_B, j_B) \in [\ell] \times [m]$. The goal is to have Alice and Bob derive secret shares of the $\ell \times m$ matrix, where the $(i_A + i_B, j_A + j_B)$-th entry is non-zero. In particular, note that this matrix will be reinterpreted as the unit vector $\mathbf{u}$ with non-zero coordinate $t = t_A + t_B \in [N]$, where $t = i_A + i_B \pmod{\ell}$ and $t = j_A + j_B \pmod{m}$. This is equivalent to the full-domain evaluation of the point function $P_{t,1}$.[3]

Our construction achieves this in two steps, which we overview next.



(a) Alice's initial input vector parsed as a matrix.

(b) Secret-shared matrix held by the parties after Step I.

(c) Secret-shared matrix held by the parties after Step II.

Fig. 1: Running example used in the overview of the NIDPF construction. The matrix represents the full evaluation of the point function with a domain of size N, where the parameters ℓ, m (as defined in Definition 9) are $\ell = m = \sqrt{N}$, for simplicity.

Step I: Shifting the columns using NIM. Alice begins by defining a $\ell \times m$ matrix $\mathbf{A}$ with 1 at entry (i_A, j_A) and zeros elsewhere. This is illustrated for the case where $\ell = m = \sqrt{N}$ in Figure 1a. Then, the main idea is to obliviously "shift" this matrix by Bob's input (i_B, j_B).

First, we observe that we can perform one dimension of this shift using a matrix multiplication: Bob defines the $m \times m$ *cyclic shift* matrix $\mathbf{S}_{j_B}$ that shifts each column of $\mathbf{A}$ cyclically to the right by j_B (wrapping around modulo m). Using NIM, Alice and Bob can non-interactively compute shares of the matrix $\mathbf{A}\mathbf{S}_{j_B}$. Note that $\mathbf{A}\mathbf{S}_{j_B}$ is a matrix where the only non-zero entry is located at row i_A and column $j_A + j_B$. This is illustrated in Figure 1b for our running example. By applying Theorem 3, we have that the communication between Alice and Bob in this process is *sublinear* in N. Moreover, by the security of the NIM scheme (see Section 4), Bob does not learn the value of (i_A, j_A) and Alice does not learn the value of j_B.

Finally, by interpreting the resulting shares back to a vector $\mathbf{y}$, the parties obtain secret shares corresponding to the full-evaluation of the point function $P_{i_A \cdot m + j_A + j_B, 1}$. Unfortunately, this is not quite what we want, since our goal is for the parties to obtain shares of the full-evaluation corresponding to the point function $P_{(i_A+i_B)\cdot m + j_A + j_B, 1} \equiv P_{t_A + t_B, 1}$. In particular, notice that following the cyclic shift, Alice still knows which *row* of the resulting matrix contains the non-zero index, since the row index does not currently depend on Bob's input i_B. To remedy this, we need a way for Bob to cyclically shift the rows of the resulting secret-shared matrix by his secret index i_B, which ideally could be done by multiplying the result with another "shift matrix" parameterized by i_B.

Sadly, multiplying by another shift matrix is not possible, since this would require a "NIM" for the degree-3 computation $\mathbf{S}_{i_B}(\mathbf{A}\mathbf{S}_{j_B})$, where $\mathbf{S}_{j_B}$ cyclically shifts the columns of $\mathbf{A}$ by j_B and $\mathbf{S}_{i_B}$ cyclically shifts the rows by i_B. We do not know how to realize such a primitive for degree-3 computations (*even* if we sacrifice the succinctness requirement) without going through "high-end" tools like multi-key FHE [DHRW16].

However, we show that Bob can cyclically shift the rows using degree-2, secret-key HSS (the weakest form of non-trivial HSS [BGI16], which can be instantiated from a wide range of assumptions). In particular, our usage of HSS to let Bob cyclically shift the rows is only possible *after* computing the NIM to cyclically shift the columns, as will become apparent later. We stress that secret-key HSS alone cannot be used to directly build NIDPFs—for one, the NIDPF abstraction directly implies public key encryption, while secret-key HSS (even for higher degree computations) does not [DIJL23].

Step II: Shifting the rows with degree-2 HSS. Our idea is to compose degree-2 HSS with NIM to allow Bob to obliviously cyclically shift the rows *and* columns of Alice's matrix $\mathbf{A}$. This composition with HSS is inspired by the multi-party DPF construction of Abram et al. [ARS24], where they use HSS to obliviously select an appropriate cyclic shift of a one-hot vector by computing

[3] For now, we assume that the point function outputs $v = 1$ at the special index i and later generalize to arbitrary outputs.

an inner product with all possible shifts. However, to apply this idea to the non-interactive setting, there are several challenges we need to overcome.

The first challenge is that HSS schemes, even for degree-2 functions, require a *trusted setup process*, which would prevent us from getting a non-interactive solution (the parties would need to engage in a multi-round setup protocol).

The second challenge is that HSS does not enable computing degree-2 functions on additive secret shares. Instead, typical HSS schemes (following [BGI16]) require parties to have "memory shares" and "input shares" of the secret values in order to perform computations over them. In particular, degree-2 HSS allows two parties to locally compute an additive sharing of xy from an input share of a value x and memory share of a value y. At a very high level, an input share of a message $x \in \mathcal{R}$ is just an encryption of x; and memory shares of a message $y \in \mathcal{R}$ are additive shares of the tuple $(x, x \cdot \mathsf{sk})$, where $\mathsf{sk} \in \mathcal{R}^k$ is the secret key used to encrypt the input share (see Section 3.5 for additional background).

If the parties can somehow obtain memory shares of $\mathbf{AS}_{j_B}$ and input shares of an input provided by Bob, then using HSS for computing degree-2 functions, we have the following solution for cyclically shifting the rows. First, Bob defines the one-hot vector $\mathbf{e}_{i_B}$ representing his row index i_B and sends HSS input shares of $\mathbf{e}_{i_B}$ to Alice. Let $\mathbf{T} := \mathbf{AS}_{j_B}$ which, for now, we assume Alice and Bob hold memory shares of at the end of Step I. Then, the parties locally define the list of ℓ "shifted" matrices $\mathbf{T}_1, \dots, \mathbf{T}_\ell$, such that $\mathbf{T}_i$ is the matrix $\mathbf{T}$ with the rows cyclically shifted down by i. Finally, using HSS, the parties compute the following degree-2 equation to "obliviously select" $\mathbf{T}_{i_B}$:

$$\langle \mathbf{e}_{i_B}, (\mathbf{T}_1, \dots, \mathbf{T}_\ell) \rangle = \mathbf{T}_{i_B}. \tag{1}$$

Observe that this allows Alice and Bob to compute shares of the one-hot matrix with a non-zero entry at index $(i_A + i_B, j_A + j_B)$, as required. A similar idea underpins the multi-party DPF construction of Abram et al. [ARS24]. However, their requirement for a trusted setup makes their approach for obtaining the necessary compatible input and memory shares inherently interactive. We make use of the following two ideas to avoid interaction:

Idea I: Bob generates the HSS setup. To avoid needing a trusted setup, we exploit the fact that Bob knows the full input $\mathbf{e}_{i_B}$, which means that he can act as the trusted dealer to generate the HSS setup in our case. Moreover, we observe that *secret-key* HSS suffices, since only Bob needs to encrypt his input $\mathbf{e}_{i_B}$. This allows us to use the most basic form of HSS, making it easy to instantiate from many standard assumptions, including a novel instantiation we present in Section 5.3.1 from the SXDH assumption in bilinear groups.

Idea II: NIM outputs memory shares. To make the output of the NIM compatible with HSS, we need a way for Alice and Bob to obtain memory shares of the matrix $\mathbf{T}$ rather than additive shares. To achieve this, we use the following trick from prior work [CMPR23, ARS24] to generate memory shares. Observe that if Bob multiplies his cyclic shift matrix $\mathbf{S}_{j_B}$ by any scalar $c \in \mathcal{R}$, the output of the NIM will be an additive share of $c \cdot \mathbf{AS}_{j_B}$. This can be generalized to computing $\mathsf{sk} \otimes \mathbf{AS}_{j_B}$ in the natural way. Then, the idea is to have Alice and Bob engage in two copies of the NIM protocol simultaneously. In both cases Alice inputs $\mathbf{A}$. Bob, on the other hand, inputs the cyclic shift matrix $\mathbf{S}_{j_B}$ in one instance, and the scaled matrix $\mathsf{sk} \otimes \mathbf{S}_{j_B}$ in the other. Together, the NIM outputs produce an HSS memory share of $\mathbf{AS}_{j_B}$ under Bob's secret key. In parallel to this, Bob generates an HSS input share for his vector $\mathbf{e}_{i_B}$ and an evaluation key ek_A, which he sends to Alice. Then, using HSS, Alice and Bob compute shares of the inner product from Equation (1).

Examining the communication costs. The communication is dominated by the NIM encodings and the length of $\mathbf{e}_{i_B}$, which is $O(\ell)$ (ignoring $\mathsf{poly}(\lambda)$ factors). Thus, the total communication is $O(\ell + m^2)$, which is sublinear in the domain size N using Theorem 3. Moreover, because this whole protocol only requires one simultaneous exchange of information, Alice and Bob can simply post their messages in the form of a public key, which aligns with our design goals.

Arbitrary payload. The above overview captures a NIDPF construction where the non-zero output (i.e., the payload) of the point function at the special index is the scalar $1 \in \mathcal{R}$. However, to satisfy a more general definition, we need to allow Alice and Bob to also jointly specify the payload v as the sum of their individual payloads v_A and v_B.

To achieve this, we observe that we can use the same "scaling trick" used by Bob to compute the product with his secret key sk to allow either Alice or Bob to specify v_A or v_B as the output.

Specifically, it is enough for one of the parties (say Alice) to simply multiply their input matrix by v_σ before generating the NIM encoding. This enables a "half-chosen" variant of the NIDPF, where only one of the two parties is allowed to specify the (secret) payload.

To generalize this to the case where the output is message $v = v_A + v_B$ jointly defined by the two parties, the parties can engage in two instances of the half-chosen protocol, in parallel, and add the resulting shares together. We explain this further in Section 5.

2.2.1 Random-payload NIDPF from SXDH

In some applications of DPFs, the payload can be *random* and only determined by the random coins of the generation algorithms.[4] Here, we overview a construction of such a NIDPF under the SXDH assumption in bilinear groups. In a nutshell, a bilinear group consists of a triple of cyclic groups: $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ with an efficient map (pairing) $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g_1, g_2, and g_T be generators for $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively. Then, for all g_1^x and g_2^y, it holds that $e(g_1^x, g_2^y) = g_T^{xy} \in \mathbb{G}_T$. This feature enables computing the multiplication "in the exponent" of the bilinear group.

Idea: Replacing Degree-2 HSS with a pairing. The main idea behind our NIDPF construction in bilinear groups is to follow the template outlined in Section 2.2 but replace Step II with a "multiplication in the exponent" using the pairing. We first construct a special succinct NIM scheme from the DDH assumption (over any suitable cyclic group $\mathbb{G}$), which outputs the result of the matrix product "in the exponent" of the group $\mathbb{G}$, restricting the ring $\mathcal{R}$ to $\mathbb{Z}_p$. This variant of NIM is appealing since it allows us to use *any* DDH group $\mathbb{G}$ to compute the matrix multiplication, but is limiting in that we cannot obtain the memory shares required for the HSS computation in Step II of Section 2.2. However, we observe that we don't need to do so if we have a pairing! Specifically, we can instead replace the HSS computation in Step II by computing the multiplication using the pairing, as sketched above.

More concretely, in our DDH-based succinct NIM variant, the parties obtain *multiplicative* shares $g^{\mathbf{R}_A}$ and $g^{\mathbf{R}_B}$, respectively, such that $g^{\mathbf{R}_A} \cdot g^{\mathbf{R}_B} = g^{\mathbf{AB}}$ (where the notation $g^{\mathbf{M}}$ denotes the matrix of group elements $g^{m_{ij}}$ for all entries $m_{ij} \in \mathbf{M}$). Therefore, by instantiating this NIM scheme in the group $\mathbb{G}_1$ of a bilinear group, the parties obtain multiplicative shares of $g_1^{\mathbf{T}}$, rather than additive shares of $\mathbf{T}$ (defined in Step II above). Nonetheless, the parties can still define multiplicative shares of the vectors $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$ "in the exponent" of g_1, as before. Then, Bob can encrypt his one-hot vector $\mathbf{e}_{i_B}$ with the aim of selecting the appropriate $\mathbf{T}_{i_B}$. However, instead of using HSS to do so, Bob simply uses ElGamal encryption in the group $\mathbb{G}_2$ to compute:

$$(g_2^{r_j}, g_2^{e_{i_B,j}} h_2^{r_j}),\ \forall j \in [m],$$

where $h_2 := g_2^{\mathsf{sk}}$ is an ElGamal public key for an sk known to Bob, and each r_j is uniformly random. Since now we need DDH to hold in both $\mathbb{G}_1$ and $\mathbb{G}_2$, we must rely on the SXDH assumption.

Given these ciphertexts, Alice and Bob compute the inner product from Equation (1) "in the exponent" using the pairing and obtain multiplicative shares of the inner product in $\mathbb{G}_T$:

$$g_T^{\langle \mathbf{e}_{i_B}, (\mathbf{T}_1, \ldots, \mathbf{T}_\ell) \rangle} = g_T^{\mathbf{T}_{i_B}}. \tag{2}$$

We can view this as replacing the HSS scheme used by Bob in the overview of Section 2.2 with a "multiplicative HSS" scheme from pairings: i.e., where the HSS outputs are multiplicatively, rather than additively, secret shared. However, as with the first scheme, Alice and Bob still need to compute "memory shares" for this multiplicative variant of HSS. Memory shares now take on the form $(g_1^{\mathbf{T}}, g_1^{\mathsf{sk}\cdot\mathbf{T}})$, which can be obtained by having Bob scale his matrix by sk.

Converting from multiplicative to additive shares. Now the issue we face is the following. The result in Equation (2) is a *multiplicative* sharing of the full-domain evaluation, which does not correspond to the desired additive shares we need for the NIDPF. To solve this problem, we need a way to "bring down" the exponent and convert it to additive shares. One way to achieve this would be using the Distributed Discrete Logarithm (DDLog) procedure [BGI16].

Using the DDLog algorithm, Alice and Bob can derive additive shares of $\mathbf{T}_{i_B}$ by applying DDLog to each entry of $g_T^{\mathbf{T}_{i_B}}$. However, there is now a problem of *correctness* for the resulting output shares

[4] Note that a secret sharing of the random payload can be derived non-interactively by each party summing all entries of its length-N DPF evaluation vector.

of the NIDPF. Specifically, DDLog has a $1/\mathsf{poly}(\lambda)$ error (in which case it outputs a uniformly random value in $\{0, 1, \ldots, M\}$), which would translate to a $1 - 1/\mathsf{poly}(\lambda)$ correctness for the output shares of the NIDPF. Having a (non-negligible) correctness error is undesirable, and prevents applying the resulting NIDPF in many contexts. We show how to sidestep this problem by making an important observation regarding use of the DDLog procedure, which we explain next.

Random payload. Surprisingly, we show that the error can in fact be avoided entirely when constructing a NIDPF with a random payload (i.e., a NIDPF which outputs a random message at the special index). In particular, we observe that existing constructions of DDLog have *no error* when given multiplicative shares of the identity element g^0 [BGI16, DKK20]. Inspired by this observation, we show that we can obtain a NIDPF with random payloads. We observe that $\mathbf{T}_{i_B}$ is a one-hot matrix, and so has only one non-zero entry. By having the parties set their payload share to a uniformly random scalar, they can further ensure the non-zero value of $\mathbf{T}_{i_B}$ has high (pseudo)entropy to both parties. Thus, we can simply use a PRF F_K with outputs in $\mathbb{Z}_M$ to generate additive shares from the multiplicative shares. To see this, note that:

- for the multiplicative shares g^{x_0}, g^{x_1} of the *non-zero* entry $g^{x_0+x_1}$ of $\mathbf{T}_{i_B}$, $F_K(g^{x_0}) - F_K(g^{-x_1})$ is a pseudorandom value in $\mathbb{Z}_M$ (even given the PRF key K); however,
- for all multiplicative shares g^{x_0}, g^{x_1} such that $g^{x_0} \cdot g^{x_1} = g^0$, we have $F_K(g^{x_0}) - F_K(g^{-x_1}) = 0$.

This means that parties obtain pseudorandom shares of zero on all entries except for the entry with the non-zero value, where they obtain shares of some pseudorandom value. We provide details on the DDLog algorithm and our NIDPF construction from SXDH in Section 5.

3 Preliminaries

In this section, we provide the necessary notational and cryptographic preliminaries that we use our construction of NIDPF.

3.1 Notation

We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, and $\mathbb{G}$ denote a finite group. We let $\mathcal{R}$ denote a finite ring. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We occasionally abuse notation and let poly denote a fixed polynomial.

Vectors and matrices. We denote a vector $\mathbf{u}$ using bold lowercase letters and let $\mathbf{u}[i]$ denote the i-th coordinate of $\mathbf{u}$. We denote matrices $\mathbf{A}$ with bold uppercase letters and let $\mathbf{A}[i, j]$ denote the element of $\mathbf{A}$ located at the i-th row, j-th column.

Sampling and assignment. We let $x \leftarrow\!\!\$\; S$ denote a uniformly random sample drawn from a set S. We let $x \leftarrow \mathcal{A}$ denote assignment from a randomized algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y (which may be the output of a deterministic algorithm).

Efficiency and indistinguishability. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

Rounding. We let $\lfloor x \rceil$ denote the rounding of a real number x to the nearest integer. For integers $q > p \geq 2$, we define the modular rounding function $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ as $\lfloor v \rceil_p = \lfloor (p/q) \cdot v \rceil$.

Party identifiers. We identify parties with letters A and B, and use $\sigma \in \{A, B\}$ to refer to a party. We will slightly abuse notation by letting $1 - \sigma$, for some $\sigma \in \{A, B\}$, refer to the party identifier in the singleton set $\{A, B\} \setminus \{\sigma\}$.

3.2 Additive secret sharing

We define the function $\mathsf{Share}_{\mathbb{G}}(\cdot)$ to be the (randomized) function that outputs a tuple of additive shares in $\mathbb{G}$, such that each share is individually uniformly random over $\mathbb{G}$. For simplicity, we will denote the tuple of additive shares of a secret s by $(\langle s \rangle_0, \langle s \rangle_1)$, such that $\langle s \rangle_0 + \langle s \rangle_1 = s \in \mathbb{G}$.

3.3 Cryptographic assumptions

In this section, we present the cryptographic assumptions we build NIDPFs from, including the DDH assumption, the SXDH assumption, and the NIDLS framework.

Definition 1 (Decisional Diffie–Hellman (DDH) Assumption). *Let λ be a security parameter. Let $\mathbb{G}$ be a cyclic group of prime order $p = p(\lambda) \in \mathsf{poly}(\lambda)$ with generator g. The DDH assumption states that: $(g, g^a, g^b, g^{ab}) \approx_c (g, g^a, g^b, g^c)$, where $a, b, c \leftarrow\!\!\$\ \mathbb{Z}_p$.*

Definition 2 (Symmetric External Diffie–Hellman (SXDH) Assumption). *Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group, where $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups of prime order $p = p(\lambda) \in \mathsf{poly}(\lambda)$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The SXDH assumption states that the DDH assumption (cf. Definition 1) holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Definition 3 (Learning With Errors Assumption). *Let χ denote a discrete Gaussian noise distribution. Let $n = n(\lambda), m = m(\lambda)$, and $q = q(\lambda)$, all polynomial in λ. The learning with errors (LWE) assumption states that $(\mathbf{A}, \mathbf{A}^\top \mathbf{s} + \mathbf{e}) \approx_c (\mathbf{A}, \mathbf{u})$ where $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}, \mathbf{s} \leftarrow\!\!\$\ \mathbb{Z}_q^n, \mathbf{e} \leftarrow\!\!\$\ \chi^m, \mathbf{u} \leftarrow\!\!\$\ \mathbb{Z}_q^m$.*

Definition 4 (Distributed Discrete Logarithm [BGI16]). *Let $\lambda \in \mathbb{N}$ be a security parameter and $\epsilon = \epsilon(\lambda)$. Let $\mathbb{G}$ be an arbitrary cyclic group with generator g and let $1 \leq M \ll |\mathbb{G}|$ be an integer. Let $\mathsf{crs} := (\mathbb{G}, g, M)$ be a common reference string. An efficient algorithm DDLog solves the distributed discrete logarithm in $\mathbb{G}$ with ϵ-correctness, if for all $x \in \mathbb{Z}_M$ and every pair of elements $h_A, h_B \in \mathbb{G}$ such that $h_A \cdot h_B = g^x$,*

$$\Pr\Big[\ \langle z\rangle_A - \langle z\rangle_B = x \quad : \quad \langle z\rangle_\sigma := \mathsf{DDLog}(\mathsf{crs}, h_\sigma),\ \forall \sigma \in \{A, B\}\Big] \geq \epsilon(\lambda).$$

3.4 The NIDLS framework

The Non-Interactive Discrete Log Sharing (NIDLS) framework [ADOS22] abstracts several HSS constructions [OSY21, RS21]. The NIDLS framework defines a finite Abelian group $\mathbb{G} = F \times H$, where the discrete log problem is easy in F and assumed to be computationally intractable in H. Essentially, this allows two parties to non-interactively compute secret shares of a discrete log in F.

Definition 5 (NIDLS Framework [ADOS22]). *The NIDLS framework consists of three efficient algorithms $(\mathsf{GGen}, \mathcal{D}, \mathsf{DDLog})$ with the following functionality:*

– $\mathsf{GGen}(1^\lambda) \to \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux})$. *The randomized group generation algorithm takes as input the security parameter and outputs a common reference string crs which consists of:*
  - *finite Abelian group $\mathbb{G}$,*
  - *subgroups F and H such that $\mathbb{G} = F \times H$,*
  - *generator g and order p of F,*
  - *positive integer t,*
  - *and auxiliary information aux.*

– $\mathcal{D}(1^\lambda, \mathsf{crs}) \to (h, \rho)$. *The randomized sampling algorithm takes as input the security parameter and common reference string, and outputs a group element $h \in \mathbb{G}$ along with some auxiliary information ρ.*

– $\mathsf{DDLog}(\mathsf{crs}, h) \to s$. *The deterministic distributed discrete log algorithm takes as input a common reference string and a group element, and outputs an element $s \in \mathbb{Z}_p$.*

The above functionality needs to satisfy the following properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$ and efficient adversaries $\mathcal{A}$, there exists a negligible function negl such that*

$$\Pr\left[\ \langle s\rangle_A - \langle s\rangle_B = m \pmod p \quad : \quad \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_A, m) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ h_B := g^m \cdot h_A \\ \langle s\rangle_A := \mathsf{DDLog}(\mathsf{crs}, h_A) \\ \langle s\rangle_B := \mathsf{DDLog}(\mathsf{crs}, h_B) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all security parameters $\lambda \in \mathbb{N}$, it holds that:*

$$\left\{ (\mathsf{crs}, h, \rho, h^r) \;\middle|\; \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, g, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}) \\ r \leftarrow\!\!\$\; [t] \end{array} \right\}$$

$$\approx_s \left\{ (\mathsf{crs}, h, \rho, h') \;\middle|\; \begin{array}{r} \mathsf{crs} := (\mathbb{G}, F, H, f, p, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}) \\ h' \leftarrow\!\!\$\; \langle h \rangle \end{array} \right\}.$$

I.e., the group elements h^r and h' are statistically *indistinguishable.*

Known instantiations. The NIDLS framework has been instantiated in the Paillier group under the DCR assumption, in class groups under a variant of the DDH assumption (see below), and in the group of elements in $\mathbb{Z}_n^*$ with a Jacobi symbol of 1 (under the quadratic residuosity assumption), where n is the product of two large random safe primes. We refer to [ADOS22, ARS24] for formal definitions of these instantiations.

To instantiate "ElGamal-like" encryption in class groups, we will need to use the Enhanced DDH assumption [ARS24]. This assumption states that given the parameters of the NIDLS group and $\ell + 1$ group elements $g_0, \dots, g_\ell$ sampled from $\mathcal{D}$ (along with the corresponding auxiliary information $\rho_0, \dots, \rho_\ell$), it is hard to distinguish between $(g_0^w, \dots, g_\ell^w)$ for a random w and $(f^{r_0} \cdot g_0^w, \dots, f^{r_\ell} \cdot g_\ell^w)$ for random $r_0, \dots, r_\ell \in \mathbb{Z}_q$.

Definition 6 (The ℓ-ary Enhanced DDH Assumption [ARS24])**.** *Let* GGen *and $\mathcal{D}$ be as defined in Definition 5. The ℓ-ary Enhanced DDH (ℓ-EDDH) assumption in the NIDLS framework states that:*

$$\left\{ \begin{array}{l} \mathsf{crs} \\ h_0, \dots, h_\ell \\ \rho_0, \dots, \rho_\ell \\ h_0^w, \dots, h_\ell^w \end{array} \;\middle|\; \begin{array}{r} (\mathbb{G}, F, H, g, q, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_j, \rho_j) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}),\ \forall j \in \{0, 1, \dots, \ell\} \\ w \leftarrow\!\!\$\; [t] \end{array} \right\}$$

$$\approx_c \left\{ \begin{array}{l} \mathsf{crs} \\ h_0, \dots, h_\ell \\ \rho_0, \dots, \rho_\ell \\ g^{r_0} \cdot h_0^w, \dots, g^{r_\ell} \cdot h_\ell^w \end{array} \;\middle|\; \begin{array}{r} (\mathbb{G}, F, H, g, q, t, \mathsf{aux}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (h_j, \rho_j) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}),\ \forall j \in \{0, 1, \dots, \ell\} \\ w \leftarrow\!\!\$\; [t] \\ r_j \overset{\$}{\leftarrow} \mathbb{Z}_q, \forall j \in \{0, 1, \dots, \ell\} \end{array} \right\}.$$

3.5 Degree-2 secret-key HSS

Here, we define the most minimal form of Homomorphic Secret Sharing (HSS), which will be sufficient for our construction of NIDPFs. The definition is adapted from a more general definition of secret-key HSS (cf. Definition 16) and is satisfied by existing HSS constructions in the NIDLS framework and from lattice-based assumptions. For completeness, we provide a general definition of secret-key HSS (for any function class) in Section 7.

Definition 7 (Degree-2 Secret-Key HSS; Adapted from [BGI16, DIJL23])**.** *Let λ be a security parameter and $\mathcal{R}$ be a finite ring. A Degree-2 (secret key) HSS scheme with message space $\mathcal{R}$ consists of four efficient algorithms* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Convert}, \mathsf{Mult})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$. *The randomized setup algorithm takes as input the security parameter and outputs a secret key* sk *and a pair of HSS evaluation keys* $(\mathsf{ek}_A, \mathsf{ek}_B)$.
- $\mathsf{Share}(\mathsf{sk}, x) \to (\llbracket x \rrbracket_A, \llbracket x \rrbracket_B)$. *The randomized share algorithm takes as input the secret key* sk *and message $x \in \mathcal{R}$. It outputs a pair of input shares of x.*
- $\mathsf{Convert}(\sigma, \mathsf{ek}_\sigma, \llbracket x \rrbracket_\sigma) \to \langle\!\langle x \rangle\!\rangle_\sigma$. *The deterministic conversion algorithm takes as input the party identifier $\sigma \in \{A, B\}$, an evaluation key ek_σ, and input share of x. It outputs a memory share of x.*

– $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma, \langle\!\langle y\rangle\!\rangle_\sigma) \to \langle z\rangle_\sigma$. *The deterministic multiplication algorithm takes as input the party identifier* $\sigma \in \{A, B\}$*, an evaluation key* ek_σ*, an input share of* x*, and a memory share of* y*. It outputs a share of* z*.*

When $\mathsf{Alg} \in \{\mathsf{Share}, \mathsf{Convert}, \mathsf{Mult}\}$ *is given as input a vector of input (or memory) shares, it outputs the vector obtained by evaluating* Alg *on each coordinate of the input vector independently.*

The above algorithms must satisfy correctness and security:

Correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, and for all messages* $x, y \in \mathcal{R}$*, we say the HSS scheme is* ϵ*-correct, for some* $0 < \epsilon \leq 1$ *if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\langle z\rangle_A - \langle z\rangle_B = xy \quad : \quad \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ ([\![x]\!]_A, [\![x]\!]_B) \leftarrow \mathsf{Share}(\mathsf{sk}, x) \\ ([\![y]\!]_A, [\![y]\!]_B) \leftarrow \mathsf{Share}(\mathsf{sk}, y) \\ \langle\!\langle y\rangle\!\rangle_\sigma := \mathsf{Convert}(\sigma, \mathsf{ek}_\sigma, [\![y]\!]_\sigma),\ \forall\sigma \in \{A, B\} \\ \langle z\rangle_\sigma := \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma, \langle\!\langle y\rangle\!\rangle_\sigma) \end{array}\right] \geq \epsilon - \mathsf{negl}(\lambda).$$

Security. *For all security parameters* $\lambda \in \mathbb{N}$ *such that, for every* $\sigma \in \{A, B\}$*, and all efficient adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$*,*

$$\Pr\left[b' = b \quad : \quad \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbf{x}_0, \mathbf{x}_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{ek}_\sigma) \\ b \leftarrow\!\!\$\ \{0, 1\} \\ ([\![\mathbf{x}_b]\!]_A, [\![\mathbf{x}_b]\!]_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, [\![\mathbf{x}_b]\!]_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where for all $b \in \{0, 1\}$*,* $\mathbf{x}_b \in \mathcal{R}^\ell$ *and* $k = k(\lambda) \in \mathsf{poly}(\lambda)$*.*

3.5.1 Memory shares in HSS schemes All existing HSS constructions in the NIDLS framework [ADOS22, ARS24], and direct constructions from DDH [BGI16, BCG+17] or lattice-based assumptions [BKS19], are constructed using the following template. The HSS secret key sk is a vector of ring elements from the ring $\mathcal{R}$ and corresponds to the decryption key of some additively-homomorphic encryption scheme with message space $\mathcal{R}$, supporting some form of linear (or nearly-linear) decryption. The evaluation keys $(\mathsf{ek}_A, \mathsf{ek}_B)$ are additive shares of the secret key sk. For degree-2 computations, *input shares* are simply encryptions of the message x under sk, while *memory shares* consist of additive shares of x and $\mathsf{sk} \cdot x$. Multiplication of an input share of x with a memory share of y can then be computed as follows. First, using the homomorphism of the encryption scheme, compute an encryption of the additive share of $z = x \cdot y$ by multiplying the encrypted message with the additive share of y. Second, using the linear decryption property of the encryption scheme, compute the decryption of the resulting ciphertext using the additive share of $y \cdot \mathsf{sk}$ to recover the additive share of z.

We formalize the property of "multiplication by a memory share," which we will use in our NIDPF construction. We note that several prior works (e.g., [CMPR23, ARS24]) make use of such "multiplication by memory shares," without explicitly formalizing the property.

Definition 8 (Multiplication by Memory Shares)**.** *Let* $\mathsf{HSS} = (\mathsf{Input}, \mathsf{Share}, \mathsf{Eval})$ *(cf. Definition 16) with a finite ring* $\mathcal{R}$ *as the message space. We say an HSS scheme supports* multiplication by a memory share *if the following three properties are simultaneously satisfied:*

(1) The secret key of the HSS scheme is a vector $\mathsf{sk} \in \mathcal{R}^k$*, for some* $k \in \mathbb{N}$*.*

(2) A memory share $\langle\!\langle y\rangle\!\rangle_\sigma$ *for any message* $y \in \mathcal{R}$ *consists of an additive share of the tuple* $(y, y \cdot \mathsf{sk})$*, defined over* $\mathcal{R}$*.*

(3) There exists an efficient, deterministic algorithm $\mathsf{MultEval}$ *with the same syntax as* Eval*, such that for all messages* $y \in \mathcal{R}$*, all memory shares* $(\langle\!\langle y\rangle\!\rangle_A, \langle\!\langle y\rangle\!\rangle_B)$ *of* y*, all input shares* $([\![x]\!]_A, [\![x]\!]_B)$ *of* $x \in \mathcal{R}$*, and all functions* f *in the family of functions computable by* HSS*, it holds that:*

$$\Pr\left[\langle z\rangle_A - \langle z\rangle_B = y \cdot f(x) \quad : \quad \begin{array}{r} \langle z\rangle_\sigma := \mathsf{MultEval}(\sigma, \mathsf{ek}_\sigma, f, [\![x]\!]_\sigma, \langle\!\langle y\rangle\!\rangle_\sigma), \\ \forall\sigma \in \{A, B\} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

In words, any computation evaluated by the HSS scheme can be "pre-multiplied" by a value y given only a memory share of y.

We note that Definition 7 explicitly captures property (3) from Definition 8. Importantly to us, Definition 8 is satisfied by all existing HSS constructions, including HSS construction from the DDH [BGI16, BCG$^+$17, BGI17], DCR [OSY21, RS21], QR [OSY21, ADOS22] (for degree-2 computations), Class Groups [ADOS22], and LWE [BKS19].

4 Non-Interactive Multiplication

In this section, we define the notion of non-interactive multiplication. As mentioned in the technical overview, this definition captures the core ingredient used in several prior works, including the non-interactive OT construction of [BM90], non-interactive VOLE (e.g., [OSY21, ARS24, BCM$^+$24]), and the notion of non-interactive inner-products [CZ22]. We believe that our abstraction is of independent interest and may aid in further studying the applications of these primitives. Indeed, in Section 6, we show that we can bootstrap the NIM abstraction to compute more expressive functions in a "non-interactive" manner, which to date, was only possible from multi-key FHE techniques and obfuscation [DHRW16].

Definition 9 (Non-Interactive Multiplication)**.** *Let λ be a security parameter and $\mathcal{R}$ be a finite ring. A Non-Interactive Multiplication (NIM) scheme consists of five efficient algorithms*

$$\mathsf{NIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma \in \{A,B\}})$$

with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string (CRS)* crs.
- $\mathsf{Encode}_\sigma(\mathsf{crs}, v) \to (\mathsf{pe}_\sigma, \mathsf{st}_\sigma)$. *The randomized encoding algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the CRS* crs *and message v. It outputs a public encoding* pe_σ *and secret state* st_σ.
- $\mathsf{Decode}_\sigma(\mathsf{crs}, \mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma) \to \langle z \rangle_\sigma$. *The deterministic decoding algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the CRS* crs*, public encoding* $\mathsf{pe}_{1-\sigma}$ *belonging to the other party, and a secret state* st_σ *belonging to party σ. It outputs a share of z over $\mathcal{R}$.*

The above functionality must satisfy correctness and security:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$ and every pair of elements $x, y \in \mathcal{R}$, a NIM scheme is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = xy \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ \langle z \rangle_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ \langle z \rangle_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all efficient adversaries $\mathcal{A}$, for all $\sigma \in \{A, B\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[b = b' \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (v_0, v_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow\!\!\$\ \{0,1\} \\ (\mathsf{pe}_\sigma, \mathsf{st}_\sigma) \leftarrow \mathsf{Encode}_\sigma(\mathsf{crs}, v_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pe}_\sigma) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In words, the public encoding hides the message.

4.1 NIM with multiplicative output reconstruction

We also define *multiplicative* rather than additive reconstruction for NIM, which will serve us in instantiating a NIDPF in bilinear groups. In this case, Decode_σ outputs a group element Z_σ for $\sigma \in \{A, B\}$, such that $Z_A / Z_B = g^{xy}$, where g is a generator of a cyclic group $\mathbb{G}$.

Definition 10 (Multiplicative Reconstruction)**.** *A NIM scheme* NIM *is said to have* multiplicative reconstruction *if the correctness property of Definition 9 is instead stated as follows.*

Multiplicative-output Correctness. *Let $\mathbb{G}$ be an abelian group of order p with generator g. For all security parameters $\lambda \in \mathbb{N}$ and every pair of elements $x, y \in \mathbb{Z}_p$, a NIM scheme (instantiated with $\mathcal{R} = \mathbb{Z}_p$) is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[Z_A \cdot Z_B = g^{xy} \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ Z_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A) \\ Z_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Remark 1 (General reconstruction). We note that we could have defined NIM (Definition 9) to have an arbitrary reconstruction algorithm, that we then instantiate either as being addition or multiplication. However, because for most applications of NIM the additive reconstruction property is more desirable, we choose to instead provide two definitions noting that the more general abstraction is possible.

4.2 Succinct NIM for matrix multiplication

When computing non-interactive *matrix* multiplication, we can realize a "batch NIM" scheme that achieves sublinear encoding size relative to the size of the output matrix, which translates to sublinearity with respect to one of the party's inputs (or the size of the joint output). Note that we cannot, in general, require succinctness in *both* of the parties inputs since this would contradict information-theoretic lower bounds [ARS24].

Definition 11 (ϵ-succinct Matrix NIM)**.** *A NIM scheme for matrix multiplication is said to be ϵ-*succinct*, for some $0 \leq \epsilon < 1$, if for all security parameters $\lambda \in \mathbb{N}$, every CRS* crs*, integers $\ell, m, k \in \mathbb{N}$ such that $\ell > k$, and every pair of matrices $(\mathbf{M}_A, \mathbf{M}_B) \in \mathcal{R}^{\ell \times m} \times \mathcal{R}^{m \times k}$, it holds that for $N := \ell \cdot m$,*

$$|\mathsf{pe}_A| \leq N^\epsilon \cdot \mathsf{poly}(\lambda, \log |\mathcal{R}|),$$

where $(\mathsf{pe}_A, \_) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, \mathbf{M}_A)$. In words, the public encoding generated by the party with the larger matrix is sublinear in the size of its matrix.

Remark 2 (Connection to "Bilinear HSS"). Abram et al. [ARS24] define the notion of Bilinear HSS, which is conceptually related to our formalization of succinct NIM. While the notions share some similarities, succinct NIM is a stronger definition due to the non-interactivity requirement. Bilinear HSS, in contrast, captures an "HSS-like" syntax, where a trusted setup process distributes keys to the parties. Succinct NIM follows straightforwardly from Bilinear HSS with (1) Strong Hasher Privacy, (2) Strong Matrix Privacy, (3) Transparent Hasher Privacy, and (4) Transparent Matrix Privacy, using the terminology and definitions from [ARS24]. However, Property (4) is not defined in [ARS24], even though we believe that it can be easily be inferred as a variant of (3).

4.3 Constructions from group-based assumptions

Our group-based construction of succinct NIM follows from the construction of succinct non-interactive VOLE protocols [ARS24, BCM+24]. Let m be an integer. Let $\mathbb{G}$ be a suitable finite-order group with generators $h_0, h_1, \ldots, h_m$ and let g be a generator for a subgroup of $\mathbb{G}$ with order p (not necessarily prime). The CRS consists of the group $\mathbb{G}$ and the generators $\mathsf{crs} := (g, h_0, h_1, \ldots, h_m)$, sampled according to some distribution we will define later. The first step in realizing matrix products is non-interactively computing the *inner-product* between two vectors [CZ22, ARS24, BCM+24]. To achieve this, we start with the construction from [ARS24]: Alice has an input vector $\mathbf{a} \in \mathbb{Z}_p^m$ and Bob has an input vector $\mathbf{b} \in \mathbb{Z}_p^m$. The goal is for the parties to obtain shares of the inner product $\langle \mathbf{a}, \mathbf{b} \rangle$ in one simultaneous round of communication. To achieve this, Alice encodes her input vector $\mathbf{a} = (a_1, \ldots, a_m)$ by computing a Pedersen commitment $d := h_0^r \prod_{i=1}^m h_i^{a_i}$, where r is uniformly random.[5] Bob encodes his input vector $\mathbf{b}$ by computing a "batched" ElGamal-like encryption $E := (h_0^s, g^{b_1} h_1^s, \ldots, g^{b_m} h_m^s)$, where s is uniformly random. Surprisingly, just these values are enough for Alice and Bob to obtain shares of the inner product. In particular, given Bob's public encoding $E = (e_0, e_1, \ldots, e_m)$, Alice computes:

$$Z_A := e_0^r \cdot \prod_{i=1}^m e_i^{a_i} = (h_0^{sr} \cdot g^{b_1 a_1} h_1^{s a_1} \cdots g^{b_m a_m} h_m^{s a_m}) = g^{\langle \mathbf{a}, \mathbf{b} \rangle} \cdot h_0^{sr} \cdot \prod_{i=1}^m h_i^{s a_i},$$

and given Alice's public encoding d, Bob computes:

$$Z_B := d^s = h_0^{rs} (\prod_{i=1}^m h_i^{a_i})^s = h_0^{sr} \cdot \prod_{i=1}^m h_i^{s a_i}.$$

Observe that $Z_A \cdot (Z_B)^{-1} = g^{\langle \mathbf{a}, \mathbf{b} \rangle}$, meaning that the parties obtain *multiplicative* shares of the inner-product. To obtain *additive* shares from the multiplicative shares (i.e., to "bring down" the exponent), the parties can use the Distributed Discrete Logarithm (DDLog) procedure [BGI16] (see also Definition 5). However, an algorithm for solving the DDLog in an *arbitrary* group $\mathbb{G}$ requires tolerating a polynomial correctness error [DKK20], which is undesirable. Fortunately, however, in was shown in [ADOS22] that for certain instantiations of $\mathbb{G}$, the DDLog procedure can be made "error free" provided that g is chosen to be a generator of a subgroup in which the discrete logarithm is easy.[6] Using the DDLog algorithm, and choosing g according to the non-interactive discrete logarithm sharing (NIDLS) framework of [ADOS22], Alice and Bob can non-interactively derive additive shares of $\langle \mathbf{a}, \mathbf{b} \rangle$, as required.

Succinct matrix products from inner products. Using a non-interactive protocol for inner-products we can clearly construct NIM for matrix multiplication by invoking the inner-product protocol in parallel. In particular, simultaneous round protocols have the appealing feature that the first messages can be reused, meaning that we can "mix-and-match" encodings of different vectors generated by Alice and Bob.

Thus, it is enough for Alice to encode her matrix $\mathbf{A} \in \mathbb{Z}_p^{\ell \times m}$ by generating a commitment d_i using randomness r_i for the row vector $\mathbf{a}_i$, for each $i \in [\ell]$. The randomness masks the entries of $\mathbf{A}$, so Alice's public encoding does not leak information. Likewise, Bob encodes his matrix $\mathbf{B} \in \mathbb{Z}_p^{m \times k}$ by encrypting the column vectors $\mathbf{b}_i$, for each $i \in [k]$. These encryptions hide the entries of $\mathbf{B}$ by the semantic security of the NIDLS-ElGamal encryption, so Bob's public encoding also does not leak information.

This allows Alice and Bob to then non-interactively compute shares of the product $\mathbf{AB}$ by locally computing the inner-products between the $\mathbf{a}_i$'s and $\mathbf{b}_j$'s. We refer to Figure 2 for a formal construction of the succinct NIM for matrix multiplication from group-based assumptions.

As observed in [ARS24, BCM+24], the above protocol has a special property that enables a *communication-succinct* variant for certain parameters of ℓ, m, and k: Alice's message is of size $O(\ell)$ and is entirely independent of m! Moreover, Bob's message only depends on m and k and is independent of ℓ.

In particular, when the size of the output matrix is $N = \ell \cdot k$, there exists a sublinear protocol (in N) by letting $\ell = N^{2/3}$ and choosing m such that $m \cdot k = N^{2/3}$. To see this, note that Alice's encoding is of size $\ell = N^{2/3}$ while Bob's encoding is of size $m \cdot k = N^{2/3}$, resulting in an optimal communication tradeoff with respect to N. This "balancing trick" works for any matrices $\mathbf{A}$ and $\mathbf{B}$, but has previously only been used in realizing succinct VOLE [ARS24, BCM+24]. Our construction of non-interactive DPFs (Section 5) is the first to exploit this property explicitly for general matrix multiplication, which proves that this technique may be of independent interest.

Lemma 1 (Extended from [ARS24])**.** *Let λ be a security parameter, $\mathcal{R}$ be a finite ring (as defined below), ℓ, m be integer parameters, and $N = \ell \cdot m$. There exist the following instantiations of succinct*

[5] Note that Alice's encoding is information-theoretically hiding, regardless of how the CRS was generated.
[6] Examples of such groups include the Paillier group $\mathbb{Z}_{n^2}^*$ and the group of quadratic residues [ADOS22].

Succinct NIM for Matrix Multiplication from Groups

Public Parameters. Matrix dimensions ℓ, m, k. NIDLS framework $(\mathsf{GGen}, \mathcal{D}, \mathsf{DDLog})$.

```
$\mathsf{NIM.Setup}(1^\lambda)$:
1 : $\mathsf{crs}_\mathbb{G} \leftarrow \mathsf{GGen}(1^\lambda)$
2 : foreach $i \in \{0, \dots, m\}$:
3 :     $(h_i, \rho) \leftarrow \mathcal{D}(1^\lambda, \mathsf{crs}_\mathbb{G})$
4 : return $\mathsf{crs} := (\mathsf{crs}_\mathbb{G}, h_0, \dots, h_m)$

$\mathsf{NIM.Encode}_A(\mathsf{crs}, \mathbf{A})$:
1 : parse $\mathsf{crs} = (\mathsf{crs}_\mathbb{G}, h_0, \dots, h_m)$
2 : foreach $i \in [\ell]$:
3 :     $r_i \leftarrow\$ [t]$
4 :     $d_i := h_0^{r_i} \prod_{j=1}^{m} h_j^{a_{i,j}}$
5 : $\mathsf{pe}_A := (d_1, \dots, d_\ell)$
6 : $\mathsf{st}_A := (\mathbf{A}, r_1, \dots, r_\ell)$
7 : return $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{NIM.Encode}_B(\mathsf{crs}, \mathbf{B})$:
1 : parse $\mathsf{crs} = (\mathsf{crs}_\mathbb{G}, h_0, \dots, h_m)$
2 : foreach $i \in [k]$:
3 :     $s_i \leftarrow\$ [t]$
4 :     $E_{i,0} := h_0^{s_i}$
5 :     foreach $j \in [m]$: $E_{i,j} = g^{b_{j,i}} h_j^{s_i}$
6 : $\mathsf{pe}_B := (E_{i,0}, \dots, E_{i,m})_{i \in [k]}$
7 : $\mathsf{st}_B := (s_1, \dots, s_k)$
8 : return $(\mathsf{pe}_B, \mathsf{st}_B)$

$\mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$:
1 : parse $\mathsf{crs} = (\mathsf{crs}_\mathbb{G}, h_0, \dots, h_m)$
2 : parse $\mathsf{pe}_B = (E_{j,0}, \dots, E_{j,m})_{j \in [k]}$
3 : parse $\mathsf{st}_A = (\mathbf{A}, r_1, \dots, r_\ell)$
4 : foreach $(i, j) \in [\ell] \times [k]$:
5 :     $Z_{i,j} := E_{j,0}^{r_i} \cdot \prod_{j'=1}^{m} E_{j,j'}^{a_{i,j'}}$
6 :     $\mathbf{Z}_A[i, j] := \mathsf{DDLog}(\mathsf{crs}_\mathbb{G}, Z_{i,j})$
7 : return $\mathbf{Z}_A$

$\mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$:
1 : parse $\mathsf{crs} = (\mathsf{crs}_\mathbb{G}, h_0, \dots, h_m)$
2 : parse $\mathsf{pe}_A = (d_1, \dots, d_\ell)$
3 : parse $\mathsf{st}_B = (s_1, \dots, s_k)$
4 : foreach $(i, j) \in [\ell] \times [k]$:
5 :     $D_{i,j} := d_i^{s_j}$
6 :     $\mathbf{Z}_B[i, j] := \mathsf{DDLog}(\mathsf{crs}_\mathbb{G}, D_{i,j})$
7 : return $\mathbf{Z}_B$
```

Fig. 2: Group-based NIM construction.

NIM with $O(N^{2/3})$ encoding size, $O(N)$ encoding time, and $O(N^{4/3})$ decoding time, for all matrices $\mathbf{A} \in \mathcal{R}^{\ell \times m}$ and $\mathbf{B} \in \mathcal{R}^{m \times m}$, where $O(\cdot)$ hides a factor of $\mathsf{poly}(\lambda, \log|\mathcal{R}|)$*:*

(1) under the DCR assumption over the Paillier group $\mathbb{Z}_{n^2}^*$, when $\mathcal{R} \subseteq \mathbb{Z}_n$;
(2) under the QR assumption over the RSA group $\mathbb{Z}_n^*$ where n is the product of two large safe-primes, when $\mathcal{R} = \mathbb{Z}_2$;
(3) under the $N^{1/3}$-ary EDDH assumption and the uniformity assumption in class groups, when $\mathcal{R} = \mathbb{Z}_p$, for any suitable prime $p = \Omega(2^\lambda)$; and
(4) under the DDH assumption in a cyclic group $\mathbb{G}$ of order p when $\mathcal{R} = \mathbb{Z}_p$ and when the negligible correctness error of Definition 9 is relaxed to $1/T^2$, for $T = T(\lambda) \in$ $\mathsf{poly}(\lambda)$*, where the* Decode *is allowed to run in time $O(T)$.*

4.3.1 Multiplicative-output NIM from DDH We observe that NIM (with multiplicative output reconstruction; Definition 10) can be instantiated under the DDH assumption over a suitable cyclic group $\mathbb{G}$ with generators g and $h_1, \dots, h_m$. To see this, observe that if we forego the DDLog procedure in the overview above, then the parties *already* obtain multiplicative shares Z_A and Z_B such that $Z_A/Z_B = g^{\langle \mathbf{a}, \mathbf{b} \rangle}$. This then carries through to the succinct NIM instantiation via the balancing trick. We will use this variant of NIM in Section 5 to realize a NIDPF scheme in bilinear groups under the SXDH assumption (an analog of the DDH assumption for bilinear groups).

4.4 Constructions from lattice-based assumptions

The LWE and RingLWE constructions follow a similar template to the NIDLS-based approach. The idea, first described in [ARS24], is to replace the Pedersen commitment computed by Alice with an SIS-based commitment[7] and replace the ElGamal encryption computed by Bob with a dual-Regev variant.

For LWE parameters (n, q, χ) and plaintext space $\mathbb{Z}_p$ for $p \ll q$, Alice and Bob encode their input vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^m$ as elements of $\mathbb{Z}_q^m$, in the natural way. For an integer $t \gg n$, determined by the SIS problem, let $\mathbf{V} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{U} \in \mathbb{Z}_q^{n\times t}$ be matrices. The CRS consists of the LWE parameters, and matrices $\mathbf{V}$ and $\mathbf{U}$.

Then, Alice computes a commitment $\mathbf{d} := \mathbf{Ur} + \mathbf{Va}$, where $\mathbf{r} \in \mathbb{Z}_q^t$ is a random vector from a short distribution that she saves as her secret state. The ElGamal encryption computed by Bob is replaced with a dual-Regev encryption variant of the form: $(\mathbf{e}_0, \mathbf{e}_1)$ where

$$\mathbf{e}_0 := \mathbf{V}^\top \mathbf{s} + \lfloor q/p \rceil\, \mathbf{b} + \mathsf{noise} \quad \text{and} \quad \mathbf{e}_1 := \mathbf{U}^\top \mathbf{s} + \mathsf{noise},$$

for a secret $\mathbf{s} \leftarrow\!\!\$\ \mathbb{Z}_q^n$ that he saves as his secret state. As before, Alice and Bob publish $\mathbf{d}$ and $\mathbf{e} := (\mathbf{e}_0, \mathbf{e}_1)$. Again, it holds that $\mathbf{d}$ is of size n (the LWE security parameter), which is independent of the vector length m. For correctness, it is enough to note that:

$$\begin{aligned}
(\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) - (\mathbf{s}^\top \cdot \mathbf{d}) &= (\mathbf{s}^\top \cdot \mathbf{V} + \lfloor q/p \rceil \cdot \mathbf{b}^\top + \mathsf{noise}) \cdot \mathbf{a} + (\mathbf{s}^\top \cdot \mathbf{U} + \mathsf{noise}) \cdot \mathbf{r} \\
&\quad - \mathbf{s}^\top \cdot (\mathbf{Ur} + \mathbf{Va}) \\
&= \lfloor q/p \rceil \cdot \mathbf{b}^\top \cdot \mathbf{a} + (\mathsf{noise}) \cdot \mathbf{a} + (\mathsf{noise}) \cdot \mathbf{r} \\
&= \lfloor q/p \rceil \cdot \mathbf{b}^\top \cdot \mathbf{a} + \mathsf{noise} \\
&\approx \lfloor q/p \rceil \cdot \langle \mathbf{a}, \mathbf{b} \rangle \bmod p.
\end{aligned}$$

Using the rounding lemma [DHRW16, BKS19], it holds that for sufficiently large q relative to p, we can locally round the shares such that

$$\Pr\Big[\big\lfloor (\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) \big\rceil_p - \big\lfloor (\mathbf{s}^\top \cdot \mathbf{d}) \big\rceil_p = \big\lfloor (\mathbf{e}_0^\top \cdot \mathbf{a} + \mathbf{e}_1^\top \cdot \mathbf{r}) - (\mathbf{s}^\top \cdot \mathbf{d}) \big\rceil_p \Big] \geq 1 - \mathsf{negl}(\lambda),$$

where $\lfloor \cdot \rceil_p$ denotes the modular rounding from $\mathbb{Z}_q$ to $\mathbb{Z}_p$. To ensure security for Bob, we require that the LWE assumption holds for $\mathbf{V}$ and $\mathbf{U}$, i.e., $\mathbf{V}^\top \mathbf{s}+\mathsf{noise}$ and $\mathbf{U}^\top \mathbf{s}+\mathsf{noise}$ look uniform, so Bob's public encoding $(\mathbf{e}_0, \mathbf{e}_1)$ computationally hides his secret input $\mathbf{b}$. To ensure security for Alice, we require that SIS holds for $\mathbf{U}$ (which follows from a hybrid argument), i.e., $\mathbf{Ur}$ looks uniform, so Alice's public encoding $\mathbf{d}$ computationally hides her secret input $\mathbf{a}$. We refer to Figure 3 for a formal description of the succinct NIM for matrix multiplication from LWE. The security of the full construction follows from the security of the dot product construction and a standard hybrid argument. We note that the construction can be equivalently instantiated from the Ring LWE (RLWE) assumption by replacing the matrices with a suitable polynomial ring.

5 Non-Interactive DPF

In this section, we define and construct Non-Interactive DPFs (NIDPFs). Our construction makes use of the NIM abstraction and constructions from Section 4.

Definition 12 (Non-Interactive Distributed Point Function)**.** *Let λ be a security parameter and $\mathbb{G}$ be a cyclic group. A non-interactive distributed point function (NIDPF) with input domain $\mathbb{Z}_N$ consists of four efficient algorithms,*

$$\mathsf{NIDPF} = (\mathsf{Setup}, (\mathsf{Gen}_\sigma, \mathsf{KeyDer}_\sigma, \mathsf{Eval}_\sigma)_{\sigma \in \{A,B\}}),$$

with the following syntax:

[7] The Short Integer Solution (SIS) problem [Ajt96] is implied by LWE.

Succinct NIM for Matrix Multiplication from Lattices

Public Parameters. Matrix dimensions ℓ, m, k. Plaintext space $\mathbb{Z}_p$. LWE parameters n and $q \gg p$, error distribution χ. SIS parameter $t \gg n$, short distribution $\mathcal{B}$.

Notation. We write $\mathbf{B}[:, i]$ to denote the ith column of the matrix $\mathbf{B}$.

```
$\mathsf{NIM.Setup}(1^\lambda)$:
  $\mathbf{V} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$
  $\mathbf{U} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times t}$
  **return** $\mathsf{crs} := (\mathbf{V}, \mathbf{U})$

$\mathsf{NIM.Encode}_A(\mathsf{crs}, \mathbf{A})$:
  **parse** $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
  **foreach** $i \in [\ell]$:
    $\mathbf{r}_i \leftarrow\!\!\$\ \mathcal{B}^t$
    $\mathbf{d}_i := \mathbf{U}\mathbf{r}_i + \mathbf{V}\mathbf{A}[i,]$
  $\mathsf{pe}_A := (\mathbf{d}_1, \ldots, \mathbf{d}_\ell)$
  $\mathsf{st}_A := (\mathbf{A}, \mathbf{r}_1, \ldots, \mathbf{r}_\ell)$
  **return** $(\mathsf{pe}_A, \mathsf{st}_A)$

$\mathsf{NIM.Encode}_B(\mathsf{crs}, \mathbf{B})$:
  **parse** $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
  **foreach** $i \in [k]$:
    $\mathbf{s}_i \leftarrow\!\!\$\ \mathbb{Z}_q^n,\ \mathbf{w}_i, \mathbf{w}'_i \leftarrow\!\!\$\ \chi^m$
    $\mathbf{e}_{i,0} := \mathbf{V}^\top \mathbf{s}_i + \lfloor q/p \rceil\, \mathbf{B}[:, i] + \mathbf{w}_i$
    $\mathbf{e}_{i,1} := \mathbf{U}^\top \mathbf{s}_i + \mathbf{w}'_i$
  $\mathsf{pe}_B := (\mathbf{e}_{i,0}, \mathbf{e}_{i,1})_{i\in[k]}$
  $\mathsf{st}_B := (\mathbf{s}_1, \ldots, \mathbf{s}_k)$
  **return** $(\mathsf{pe}_B, \mathsf{st}_B)$

$\mathsf{NIM.Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A)$:
  **parse** $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
  **parse** $\mathsf{pe}_B = (\mathbf{e}_{i,0}, \mathbf{e}_{i,1})_{j\in[k]}$
  **parse** $\mathsf{st}_A = (\mathbf{A}, \mathbf{r}_1, \ldots, \mathbf{r}_\ell)$
  **foreach** $(i, j) \in [\ell] \times [k]$:
    $\mathbf{Z}_A[i, j] := \mathbf{e}_{j,0}^\top \cdot \mathbf{A}[i,] + \mathbf{e}_{j,1}^\top \cdot \mathbf{r}_i$
  **return** $\mathbf{Z}_A$

$\mathsf{NIM.Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B)$:
  **parse** $\mathsf{crs} = (\mathbf{V}, \mathbf{U})$
  **parse** $\mathsf{pe}_A = (\mathbf{d}_1, \ldots, \mathbf{d}_\ell)$
  **parse** $\mathsf{st}_B = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$
  **foreach** $(i, j) \in [\ell] \times [k]$:
    $\mathbf{Z}_B[i, j] := \mathbf{s}_j^\top \cdot \mathbf{d}_i$
  **return** $\mathbf{Z}_B$
```

Fig. 3: Lattice-based NIM construction.

– $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. *The randomized setup algorithm takes as input the security parameter and outputs a common reference string* crs.
– $\mathsf{Gen}_\sigma(\mathsf{crs}, t_\sigma, v_\sigma) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$. *The randomized generation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, an index $t_\sigma \in \mathbb{Z}_N$, and a message $v_\sigma \in \mathbb{G}$. It outputs a public key pk_σ and secret key sk_σ.*
– $\mathsf{KeyDer}_\sigma(\mathsf{crs}, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma) \to \kappa_\sigma$. *The deterministic key derivation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, public key $\mathsf{pk}_{1-\sigma}$ belonging to the other party, and a secret key sk_σ belonging to party σ. It outputs a DPF key κ_σ for party σ.*
– $\mathsf{Eval}_\sigma(\mathsf{crs}, \kappa_\sigma, x) \to \langle y \rangle_\sigma$. *The deterministic evaluation algorithm is parameterized by a party identifier $\sigma \in \{A, B\}$. It takes as input the* crs*, the party's DPF key κ_σ, and an input $x \in \mathbb{Z}_N$. It outputs a share of the evaluation result y.*

Let $P_{i,v} : \mathbb{Z}_N \to \mathbb{G}$ be the point function that outputs 0 for all inputs $x \neq i$ and outputs v otherwise. The above functionality must satisfy the following correctness and security properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$, every pair of indices $t_A, t_B \in \mathbb{Z}_N$ such that $t = t_A + t_B \in \mathbb{Z}_N$, every pair of messages $v_A, v_B \in \mathbb{G}$ such that $v = v_A + v_B \in \mathbb{G}$, and every input $x \in \mathbb{Z}_N$, a NIDPF scheme is correct if there exists a negligible function* negl *such that:*

$$\Pr\left[\langle y\rangle_A - \langle y\rangle_B = P_{t,v}(x) \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{Gen}_\sigma(\mathsf{crs}, t_\sigma, v_\sigma),\ \forall \sigma \in \{A, B\} \\ \kappa_A := \mathsf{KeyDer}_A(\mathsf{crs}, \mathsf{pk}_B, \mathsf{sk}_A) \\ \kappa_B := \mathsf{KeyDer}_B(\mathsf{crs}, \mathsf{pk}_A, \mathsf{sk}_B) \\ \langle y\rangle_\sigma := \mathsf{Eval}_\sigma(\mathsf{crs}, \kappa_\sigma, x),\ \forall \sigma \in \{A, B\} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins used by Gen*.*

Security. *For all efficient adversaries $\mathcal{A}$, for all $\sigma \in \{A, B\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[b = b' \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (t_0, v_0, t_1, v_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow\!\!\$ \{0, 1\} \\ (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \mathsf{Gen}_\sigma(\mathsf{crs}, t_b, v_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pk}_\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In words, the public key computationally hides the encoded index and message.

We now define two variants of NIDPF that we will consider in this paper. The first variant, which we call a *half-chosen* NIDPF, only allows one of the parties to specify the output message, forcing the second party to input $\perp$ to Gen. The second variant, which we call a *random-output* NIDPF, does not allow the parties to specify the output message: the output v is uniformly random and determined solely based on the random coins of Gen_A and Gen_B.

Definition 13 (Half-Chosen NIDPF)**.** *We say that a NIDPF scheme has a* half-chosen payload *if for a fixed $\sigma \in \{A, B\}$,* Gen_σ *only accepts $v_\sigma = 0$.*

Definition 14 (Random-Payload NIDPF)**.** *We say that a NIDPF scheme has a* random payload *if both* Gen_A *and* Gen_B *do not take any message parameter, and the NIDPF correctness property from Definition 12 instead holds with respect to a random payload $v \in \mathbb{G}$ (determined by the random coins of* Gen*).*

Lemma 2 (Half-Chosen NIDPF $\Longrightarrow$ NIDPF)**.** *Given a half-chosen NIDPF with (public and secret) encoding size S and evaluation time T, we can obtain a NIDPF with encoding size size $2S$ and evaluation time $2T$.*

The lemma follows directly from the composition theorem of function secret sharing [BGI15, Section 3.2]. In particular, the parties run two instances of the half-chosen NIDPF in parallel, where each party specifies its own payload in turn by reversing roles, then the outputs of the two instances are summed together.

Remark 3 (Full-domain evaluation). Our construction will focus on settings where the evaluation algorithm Eval is applied on all inputs in the domain $\mathbb{Z}_N$ (in which case we need to assume that N is polynomial in the security parameter, for efficiency). That is, given a NIDPF scheme $\mathsf{NIDPF} = (\mathsf{Setup}, (\mathsf{Gen}_\sigma, \mathsf{KeyDer}_\sigma, \mathsf{Eval}_\sigma)_{\sigma \in \{A,B\}})$, we will denote by $\mathsf{EvalAll}_\sigma$ the algorithm that runs $\mathsf{NIDPF.Eval}_\sigma$ on every input $x \in \mathbb{G}$. As such, $\mathsf{EvalAll}_\sigma$ only takes as input the crs and key κ_σ.

This setting captures a motivated range of applications and implemented systems, including constructions of pseudorandom correlation generators, private "reading" applications such as Private Information Retrieval, and private "writing" applications such as secure distributed storage, voting, and aggregation. (See, e.g., [BGIK22] for further discussion.)

We additionally remark that all present black-box distributed DPF setup protocols require domains of feasible size. Indeed, removing this limitation while remaining black-box in the underlying cryptography would seem to pose a significant challenge.

5.1 Emulating arithmetic modulo N

Here, we briefly describe two natural approaches for representing arithmetic over the inputs of Alice and Bob. We want our construction to give parties the ability to generate keys for the point function with a non-zero index at $t_A + t_B \bmod N$. Unfortunately, our NIDPF construction requires Alice and Bob to parse their inputs $t_A, t_B \in [N]$ as $(i_A, j_A), (i_B, j_B) \in \mathbb{Z}_\ell \times \mathbb{Z}_m$, where $N = \ell \cdot m$ and only allows them to compute a DPF key encoding a point function with special index:

$$(i_A + i_B \bmod \ell) \cdot m + (j_A + j_B \bmod m). \tag{3}$$

If we parse the indices t_A and t_B of each party in the simplest way, i.e., $t_A = i_A \cdot m + j_A$ and $t_B = i_B \cdot m + j_B$, the above operation does not capture addition of t_A and t_B modulo N. Specifically, it is possible that $j_A + j_B$ has a "carry bit" b in the case when $j_A + j_B \geq m$, which then has to be added to the $i_A + i_B$ component as:

$$(i_A + i_B + b \bmod \ell) \cdot m + (j_A + j_B \bmod m). \tag{4}$$

Concretely, in our NIDPF construction, this will require shifting the rows of the matrix $\mathbf{T}$ by $i_B + b$ in the case that the carry bit is set (recall Step II from Section 2.2).

Remark 4 (Random point function). We remark that in the case that Alice and Bob need a point function with a *random* non-zero index, they do not need to emulate addition modulo N and can instead simply sample uniformly random (i_σ, j_σ) for their inputs to the NIDPF.

Here, we present two approaches for emulating addition (modulo N) using arithmetic represented over ℓ and m, as in Equation (3).

Approach I: Emulating arithmetic via a residue number system. As described in Section 2, we can let ℓ be coprime to m, which immediately allows us to emulate arithmetic modulo N in a residue number system, using ℓ and m as the coprime moduli. In this case, we no longer need to worry about the carry bit, since we can compute locally modulo $\mathbb{Z}_\ell$ and $\mathbb{Z}_m$ and then map back to $\mathbb{Z}_N$. Alice and Bob represent their indices $t_A, t_B \in [N]$ as (i_A, j_A) and (i_B, j_B) where

$$\begin{aligned} t_A &\equiv i_A \pmod{\ell}, & t_B &\equiv i_B \pmod{\ell}, \\ t_A &\equiv j_A \pmod{m}, & t_B &\equiv j_B \pmod{m}. \end{aligned}$$

After executing the protocol, they hold secret shares of a matrix that is nonzero in location $(i_A + i_B \pmod{\ell}, j_A + j_B \pmod{m})$. Let α, α' be integers such that $\alpha\ell + \alpha' m = 1$, which exist since ℓ and m are coprime. Alice and Bob can each map location (i, j) of their $\ell \times m$ matrix to location $l \in [N]$ in a vector of length N where $l \equiv i\alpha' m + j\alpha\ell \pmod{N}$. Suppose t is the resulting one-hot index in the vector Alice and Bob now hold shares for. By the Chinese Remainder Theorem, we have that

$$\left.\begin{aligned} t &\equiv i_A + i_B \pmod{\ell} \\ t &\equiv j_A + j_B \pmod{m} \end{aligned}\right\} \implies t \equiv t_A + t_B \pmod{N}.$$

Approach II: Emulating the carry. For some applications, it may be inconvenient or impossible for ℓ and m to be coprime, such as if $N = \ell \cdot m$ is a power of 2. An alternative strategy we can use in this case is to prevent the "erasure" of the carry bit modulo m. Specifically, we observe that if the cyclic shift is performed modulo $2m$, then we do not lose information on the carry: the non-zero entry of Alice's matrix $\mathbf{A} \in \mathcal{R}^{\ell \times 2m}$ will either contain the non-zero entry in the "left half" or the "right half" of the columns depending on whether or not the carry occurred. At this point, Alice and Bob will hold shares of a matrix $\mathbf{T}$ that can be parsed as $[\mathbf{T}_0\ \mathbf{T}_1]$, where $\mathbf{T}_0, \mathbf{T}_1 \in \mathcal{R}^{\ell \times m}$ and $\mathbf{T}_{1-b} = \mathbf{0}$ when b is the value of the carry bit. Then, Alice and Bob can cyclically shift their rows of $\mathbf{T}_1$ down by one (this operation is a linear function over their shares of $\mathbf{T}_1$), and compute $\mathbf{T} := \mathbf{T}_0 + \mathsf{ShiftDown}(\mathbf{T}_1, 1)$. Observe that because $\mathbf{T}_{1-b}$ is always zero, they obtain shares of the matrix $\mathbf{T}$ that has exactly one non-zero entry and the rows cyclically shifted down precisely if the carry bit is set. Note that this approach works regardless of the choice of ℓ and m.

5.2 NIDPF framework

Here, we formalize the NIDPF construction, closely following the technical overview from Section 2.2. We present a construction for the "half-chosen payload" (Definition 13) NIDPF in Figure 4, which can be extended to satisfy the full NIDPF definition via Lemma 2 (however, for the applications described in Section 1.2, the payload is public, and so the half-chosen variant sufficient on its own).

Our construction uses the following auxiliary functions as building blocks.

Auxiliary functions. We define two deterministic functions that simplify the presentation of our NIDPF construction in Figure 4.

- $\mathsf{Shift}\colon \mathcal{R}^{\ell\times m} \times [\ell] \to \mathcal{R}^{\ell\times m}$. Shift takes as input a $\ell \times m$ matrix (for arbitrary integers ℓ, m) and a shift $i \in [\ell]$. It outputs the matrix with the rows cyclically shifted down by i.
- $\mathsf{Mat2Vec}\colon \mathcal{R}^{\ell\times m} \to \mathcal{R}^{\ell\cdot m}$. Mat2Vec takes as input a $\ell \times m$ matrix (for arbitrary integers ℓ, m) and outputs the vector obtained by concatenating the rows of the matrix together.

NIDPF Framework

Public Parameters. Domain size N and matrix dimensions ℓ, m such that $\ell \cdot m = N$. Set of cyclic shift matrices $\mathcal{S}_m = \{\mathbf{S}_j : j \in [m]\}$, where $\mathbf{S}_j$ is the j-th canonical cyclic shift matrix. Succinct NIM scheme $\mathsf{NIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma\in\{A,B\}})$ and (degree-2, secret-key) HSS scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Convert}, \mathsf{Mult})$. We instantiate the full-domain evaluation algorithm $\mathsf{NIDPF.EvalAll}$, described in Remark 3.

```
$\mathsf{NIDPF.Setup}(1^\lambda)$:
  1 : $\mathsf{crs} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$
  2 : $\textbf{return } \mathsf{crs}$

$\mathsf{NIDPF.Gen}_A(\mathsf{crs}, t_A, v)$:
  1 : $\textbf{parse } t_A = (i_A, j_A) \in [\ell] \times [m]$
  2 : $\mathbf{A} := \mathbf{0} \in \mathcal{R}^{\ell\times m},\ \mathbf{A}[i_A, j_A] := v$
  3 : $(\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, \mathbf{A})$
  4 : $(\mathsf{pk}_A, \mathsf{sk}_A) := (\mathsf{pe}_A, \mathsf{st}_A)$
  5 : $\textbf{return } (\mathsf{pk}_A, \mathsf{sk}_A)$

$\mathsf{NIDPF.KeyDer}_\sigma(\mathsf{crs}, \mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma)$:
  1 : $\textbf{parse } (\mathsf{pk}_{1-\sigma}, \mathsf{sk}_\sigma) = (\mathsf{pe}_{1-\sigma}, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma, \mathsf{st}_\sigma)$
  2 : $\mathbf{U}_\sigma := \mathsf{Decode}_\sigma(\mathsf{pe}_{1-\sigma}, \mathsf{st}_\sigma)$
  3 : $\textbf{parse } \mathbf{U}_\sigma = \langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma$
  4 : $\kappa_\sigma := (\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma)$
  5 : $\textbf{return } \kappa_\sigma$

$\mathsf{NIDPF.Gen}_B(\mathsf{crs}, t_B, \perp)$:
  1 : $\textbf{parse } t_B = (i_B, j_B) \in [\ell] \times [m]$
  2 : $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$
  3 : $\textbf{parse } \mathsf{sk} = (\mathsf{sk}_B, \ldots, \mathsf{sk}_k) \in \mathcal{R}^k$
  // $k$ is determined by the HSS scheme.
  4 : $([\![\mathbf{e}_{i_B}]\!]_A, [\![\mathbf{e}_{i_B}]\!]_B) \leftarrow \mathsf{HSS.Share}(\mathsf{sk}, \mathbf{e}_{i_B})$
  // $\mathbf{e}_{i_B}$ is the $i_B$-th canonical unit vector.
  5 : $\mathbf{E} := \left[\mathbf{S}_{j_B} \mid \mathsf{sk}_B \cdot \mathbf{S}_{j_B} \mid \cdots \mid \mathsf{sk}_k \cdot \mathbf{S}_{j_B}\right]$
  6 : $(\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, \mathbf{E})$
  7 : $\mathsf{pk}_B := (\mathsf{pe}_B, \mathsf{ek}_A, [\![\mathbf{e}_{i_B}]\!]_A)$
  8 : $\mathsf{sk}_B := (\mathsf{st}_B, \mathsf{ek}_B, [\![\mathbf{e}_{i_B}]\!]_B)$
  9 : $\textbf{return } (\mathsf{pk}_B, \mathsf{sk}_B)$

$\mathsf{NIDPF.EvalAll}_\sigma(\mathsf{crs}, \kappa_\sigma)$:
  1 : $\textbf{parse } \kappa_\sigma = (\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, \mathsf{ek}_\sigma, [\![\mathbf{e}_{i_B}]\!]_\sigma)$
  // $[\![\mathbf{e}_{i_B}]\!]_\sigma = ([\![e_{i_B,1}]\!]_\sigma, \ldots, [\![e_{i_B,k}]\!]_\sigma)$.
  2 : $\textbf{foreach } j \in [\ell]$ :
  3 :     $\langle\!\langle \mathbf{T}_j \rangle\!\rangle_\sigma := \mathsf{Shift}(\langle\!\langle \mathbf{T} \rangle\!\rangle_\sigma, i)$
  4 :     $\mathbf{Z}_\sigma^{(j)} := \mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, [\![e_{i_B,j}]\!]_\sigma, \langle\!\langle \mathbf{T}_j \rangle\!\rangle_\sigma)$
  5 : $\mathbf{Z}_\sigma := \sum_{j=1}^{\ell} \mathbf{Z}_\sigma^{(j)}$
  6 : $\langle \mathbf{y} \rangle_\sigma := \mathsf{Mat2Vec}(\mathbf{Z}_\sigma)$
  7 : $\textbf{return } \langle \mathbf{y} \rangle_\sigma$
```

Fig. 4: NIDPF framework.

Proposition 1. *Let* NIM *be a succinct NIM scheme (Definition 11) and let* HSS *be a (degree-2, secret-key) HSS scheme (Definition 7). The construction presented in Figure 4 is a half-chosen NIDPF (Definition 13).*

Proof. We prove correctness and security in turn.

Correctness. By correctness of NIM, we have that

$$\mathbf{U}_A + \mathbf{U}_B = \mathbf{A}\mathbf{E} = \Big[\mathbf{A}\mathbf{S}_{j_B} \mid \mathsf{sk}_B \cdot \mathbf{A}\mathbf{S}_{j_B} \mid \cdots \mid \mathsf{sk}_k \cdot \mathbf{A}\mathbf{S}_{j_B}\Big].$$

In words, $\mathbf{U}_\sigma$ is an HSS memory share of $\mathbf{T} = \mathbf{A}\mathbf{S}_{j_B}$. Recall that the vector $\mathbf{e}_{i_B}$ has entries $e_{i_B,j} = 0$ for $j \neq i_B$ and $e_{i_B,i_B} = 1$. Then, by correctness of HSS, we have that

$$\begin{aligned}\mathbf{Z}_A - \mathbf{Z}_B &= \sum_{j\in[\ell]} (\mathbf{Z}_A^{(j)} - \mathbf{Z}_B^{(j)}) = (\mathbf{Z}_A^{(i_B)} - \mathbf{Z}_B^{(i_B)}) + \mathbf{0} \\ &= \mathbf{T}_{i_B} = \mathsf{Shift}(\mathbf{T}, i_B) = \mathsf{Shift}(\mathbf{A}\mathbf{S}_{j_B}, i_B).\end{aligned}$$

$\mathbf{A}$ is zero everywhere except location (i_A, j_A), so $\mathbf{A}\mathbf{S}_{j_B}$ is zero everywhere except location (i_A, j_A+j_B), and $\mathsf{Shift}(\mathbf{A}\mathbf{S}_{j_B}, i_B)$ is zero everywhere except location $(i_A + i_B, j_A + j_B)$, as desired.

Security. Note that $\mathsf{pk}_A = \mathsf{pe}_A$ where pe_A is a public encoding generated by Encode_A for message $\mathbf{A}$ defined by t_A, v. This computationally hides t_A and v by NIM security. To prove security for $\mathsf{pk}_B = (\mathsf{pe}_B, \mathsf{ek}_A, [\![\mathbf{e}_{i_B}]\!]_A)$, we proceed via a hybrid argument.

– *Hybrid* $\mathcal{H}_0$. This hybrid consists of the NIDPF game instantiated using the construction from Figure 4 with index t_0 and payload v_0 (i.e., when $b = 0$).

– *Hybrid* $\mathcal{H}_1$. In this hybrid, we set $\mathbf{E} := \mathbf{0} \in \mathcal{R}^{m\times m(k+1)}$.

  Claim. $\mathcal{H}_1 \approx_c \mathcal{H}_0$ assuming the security of NIM.

  Proof. Suppose an efficient adversary $\mathcal{A}$ distinguishes between $\mathcal{H}_1$ and $\mathcal{H}_0$. The existence of $\mathcal{A}$ contradicts NIM security: the reduction asks the NIM challenger for pe_B to be either a public encoding of $\mathbf{E}$ or $\mathbf{0}$, samples $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ itself, and uses $\mathcal{A}$ to break NIM security of Encode. □

– *Hybrid* $\mathcal{H}_2$. In this hybrid, we set $\mathbf{e}_{i_B} := \mathbf{0} \in \mathcal{R}^\ell$.

  Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ assuming the security of HSS.

  Proof. Suppose an efficient adversary $\mathcal{A}$ distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$. The existence of $\mathcal{A}$ contradicts HSS security: the reduction asks the HSS challenger for pe_B to be either a share of $\mathbf{e}_{i_B}$ or $\mathbf{0}$, samples $\mathsf{crs} \leftarrow \mathsf{NIM.Setup}(1^\lambda)$, computes $\mathsf{pe}_B \leftarrow \mathsf{Encode}_B(\mathsf{crs}, \mathbf{0})$ itself, and finally uses $\mathcal{A}$ to break HSS security of $\mathsf{HSS.Share}$. □

At this point, there is no information on t_0 and v_0. By continuing with the same sequence of hybrids in reverse order, we can show that $\mathcal{H}_0$ is indistinguishable from the NIDPF game instantiated using the construction from Figure 4 with index t_1 and payload v_1 (i.e., when $b = 1$). As such, no efficient adversary can distinguish between the $b = 0$ and $b = 1$ case with better than negligible advantage, which concludes the proof of security. ■

Corollary 1. *There exist the following instantiations of Figure 4 with a* $O(N^{2/3})$ *public key size,* $O(N)$ *key generation time,* $O(N^{4/3})$ *key derivation time, and* $O(N^{5/3})$ *full domain evaluation time, where* $O(\cdot)$ *hides a factor of* $\mathsf{poly}(\lambda, \log|\mathcal{R}|)$*:*

– *under the DCR assumption over the Paillier group* $\mathbb{Z}_{n^2}^*$*, when* $\mathcal{R} \subseteq \mathbb{Z}_n$*;*

– *under the QR assumption over the RSA group* $\mathbb{Z}_n^*$ *where* n *is the product of two large safe-primes, when* $\mathcal{R} = \mathbb{Z}_2$*;*

– *under the $N^{1/3}$-ary EDDH assumption and the uniformity assumption in class groups, when $\mathcal{R} = \mathbb{Z}_p$, for any suitable prime $p = \Omega(2^\lambda)$; and*
– *under the LWE/RLWE assumption with a superpolynomial modulus-to-noise ratio, when $\mathcal{R} = \mathbb{Z}_p$, for any integer p.*

The class group and LWE/RLWE instantiations have a transparent setup.

Proof. We set parameters $\ell = N^{2/3}$ and $m = N^{1/3}$. Since the HSS secret key length $k = k(\lambda) = \mathsf{poly}(\lambda)$, we will ignore factors of k in our analysis below.

The size of the public key generated for the larger matrix with dimensions $\ell \times m$ grows with the number of rows, resulting in an asymptotic size of $N^{2/3}$. The size of the public key generated for a smaller matrix with dimensions $m \times m$ grows with the size of the matrix, resulting in an asymptotic size of $N^{1/3} \cdot N^{1/3} = N^{2/3}$. Key generation time is dominated by the NIM encoding algorithm for a matrix of size $\ell \times m$, resulting in an asymptotic runtime of $N^{2/3} \times N^{1/3} = N$.

Key derivation time is dominated by the NIM decoding algorithm for matrices with dimensions $\ell \times m$ and $m \times m$, resulting in an asymptotic runtime of $N^{2/3} \cdot N^{1/3} \cdot N^{1/3} = N^{4/3}$. Full domain evaluation time is dominated by running the HSS multiplication algorithm ℓ times for a matrix with dimensions $\ell \times m$, resulting in an asymptotic runtime of $N^{2/3} \cdot N^{2/3} \cdot N^{1/3} = N^{5/3}$. ■

5.3 Random-payload instantiation from SXDH

Here, we provide a construction of Figure 4 with random payload (see Definition 14) from the SXDH assumption over bilinear groups. Our starting point is the observation that if we replace the NIM in Figure 4 with the multiplicative-output NIM (Definition 10), we can avoid the error introduced from the DDLog procedure converting the multiplicative shares into additive shares. However, by having the output of NIM be multiplicative, we lose the ability to compute the HSS multiplication in $\mathsf{EvalAll}$, since HSS requires additive memory shares.

We overcome this by constructing a new degree-2 HSS scheme satisfying Definition 7 and which has “multiplicative” memory shares (i.e., additive memory shares “in the exponent”) that are compatible with the outputs of the multiplicative NIM.

5.3.1 Degree-2 HSS with Multiplicative Memory Shares In Figure 5, we construct a (secret-key, degree-2) HSS scheme satisfying Definition 7 under the SXDH assumption in bilinear groups.[8] The construction follows the standard template for realizing HSS in cyclic groups. However, one difference is that we define input shares over the group $\mathbb{G}_1$ and memory shares over the group $\mathbb{G}_2$, which allows us to compute the multiplication using a pairing operations. This slightly complicates the scheme since now we need to convert input shares to memory shares using $\mathsf{Convert}$ by “hopping between groups,” which necessitates defining two independent encryptions of the message in $\mathsf{HSS.Share}$ when generating an input share. Importantly, the encryptions in $\mathbb{G}_1$ need to be generated using an encryption key α that is independent from the encryption key β used to encrypt the messages in $\mathbb{G}_2$ (otherwise the security of the encryption would be trivially broken via the pairing).

Proposition 2. *The construction presented in Figure 5 satisfies Definition 7 (degree-2, secret-key) HSS with $\epsilon = 1 - 1/\mathsf{poly}(\lambda)$ correctness assuming the SXDH assumption holds in the bilinear group $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$.*

Proof. We prove correctness and security in turn.

Correctness. To simplify analysis later on, we show that $\mathsf{Convert}$ outputs memory shares that are partially consistent with the template outlined in Section 3.5.1. Notice that given an input share $[\![x]\!]_\sigma$ of the form $(g_1^r,\ g_1^{x+\alpha\cdot r},\ g_2^{r'},\ g_2^{x+\beta\cdot r'})$, $\langle\!\langle x\rangle\!\rangle_\sigma$ consists of a tuple of multiplicative shares of $(x, x\cdot\alpha)$ defined over $\mathbb{G}_2$, since

$$E_1 \cdot E_0^{-\langle\beta\rangle_\sigma} = g_2^{x+\beta\cdot r'} \cdot (g_2^{r'})^{-\langle\beta\rangle_\sigma} = g_2^{\langle x\rangle_\sigma} \text{ and}$$
$$E_1^{-\langle\alpha\rangle_\sigma} \cdot E_0^{-\langle\gamma\rangle_\sigma} = (g_2^{x+\beta\cdot r'})^{\langle\alpha\rangle_\sigma} \cdot (g_2^{r'})^{-\langle\alpha\beta\rangle_\sigma} = g_2^{\langle x\cdot\alpha\rangle_\sigma},$$

[8] The construction can easily be made *public key*, but we present the secret-key variant for consistency with the rest of this paper.

Degree-2 Secret-key HSS from SXDH

Public Parameters. Bilinear group of order p defined by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. For convenience, we define $g_T := e(g_1, g_2) \in \mathbb{G}_T$. Additive secret sharing algorithm $\mathsf{Share}_{\mathbb{G}}(\cdot)$ outputting two-out-of-two shares in the group $\mathbb{G}$ (see Section 3.2). Distributed discrete logarithm algorithm DDLog (Definition 4) and an integer $2 \leq M \leq \mathsf{poly}(\lambda)$ defining the message space $\{0, 1, \ldots, M-1\}$.

```
$\mathsf{HSS.Setup}(1^\lambda)$:
  1 : $\alpha, \beta \leftarrow\$ \mathbb{Z}_p,\ \gamma := \alpha\beta,\ \mathsf{sk} := (\alpha, \beta, \gamma)$
  2 : $(\langle\alpha\rangle_A, \langle\alpha\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\alpha)$
  3 : $(\langle\beta\rangle_A, \langle\beta\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\beta)$
  4 : $(\langle\gamma\rangle_A, \langle\gamma\rangle_B) \leftarrow \mathsf{Share}_{\mathbb{Z}_p}(\gamma)$
  5 : foreach $\sigma \in \{A, B\}$:
  6 :     $\mathsf{ek}_\sigma := (\langle\alpha\rangle_\sigma, \langle\beta\rangle_\sigma, \langle\gamma\rangle_\sigma)$
  7 : return $(\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$
```

```
$\mathsf{HSS.Convert}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma)$:
  1 : parse $\mathsf{ek}_\sigma = (\langle\alpha\rangle_\sigma, \langle\beta\rangle_\sigma, \langle\gamma\rangle_\sigma)$
  2 : parse $[\![x]\!]_\sigma = (\_, \_, E_0, E_1)$
  3 : $\langle\!\langle x \rangle\!\rangle_\sigma := (E_1 \cdot E_0^{-\langle\beta\rangle_\sigma}, E_1^{\langle\alpha\rangle_\sigma} \cdot E_0^{-\langle\gamma\rangle_\sigma})$
  4 : return $\langle\!\langle x \rangle\!\rangle_\sigma$
```

```
$\mathsf{HSS.Share}(\mathsf{sk}, x)$:
  1 : $r, r' \leftarrow\$ \mathbb{Z}_p$
  2 : foreach $\sigma \in \{A, B\}$:
  3 :     $[\![x]\!]_\sigma := (g_1^r,\ g_1^{x+\alpha\cdot r},\ g_2^{r'},\ g_2^{x+\beta\cdot r'})$
  4 : return $([\![x]\!]_A, [\![x]\!]_B)$
```

```
$\mathsf{HSS.Mult}(\sigma, \mathsf{ek}_\sigma, [\![x]\!]_\sigma, \langle\!\langle y \rangle\!\rangle_\sigma)$:
  1 : parse $[\![x]\!]_\sigma = (D_0, D_1, \_, \_)$
  2 : parse $\langle\!\langle y \rangle\!\rangle_\sigma = (S_\sigma^{(0)}, S_\sigma^{(1)})$
  3 : $Z_\sigma := e(D_0, S_\sigma^{(1)}) \cdot e(D_1, S_\sigma^{(0)})$
  4 : $\tau := 0$ if $\sigma = A$; else $\tau := 1$
  5 : $\langle z \rangle_\sigma := \mathsf{DDLog}((\mathbb{G}_T, g_T, M), Z_\sigma^{-1^\tau})$
  6 : return $\langle z \rangle_\sigma$
```

Fig. 5: Degree-2 secret-key HSS from SXDH.

which follows the template from Section 3.5.1, when memory shares are defined multiplicatively in the group $\mathbb{G}_2$ with respect to secret key α. Specifically, we can "ignore" the secret key β, which is only used to convert from input to memory shares and does not aid in decryption when performing $\mathsf{HSS.Mult}$.

Now, we turn to proving correctness, as defined in Definition 7. First, observe that Z_σ, as computed in $\mathsf{HSS.Mult}$ (line 3), is defined as

$$Z_A = e(g_1^r, g_2^{\langle -y\cdot\alpha\rangle_A}) \cdot e(g_1^{x+\alpha\cdot r}, g_2^{\langle y\rangle_A}) = g_T^{\langle -\alpha\cdot(ry)\rangle_A} \cdot g_T^{\langle yx+\alpha\cdot(ry)\rangle_A} = g_T^{\langle xy\rangle_A}$$
$$Z_B = e(g_1^r, g_2^{\langle -y\cdot\alpha\rangle_B}) \cdot e(g_1^{x+\alpha\cdot r}, g_2^{\langle y\rangle_B}) = g_T^{\langle -\alpha\cdot(ry)\rangle_B} \cdot g_T^{\langle yx+\alpha\cdot(ry)\rangle_B} = g_T^{\langle xy\rangle_B}.$$

Therefore, party $\sigma \in \{A, B\}$ obtains Z_σ, which is a multiplicative share of g_T^{xy}. By correctness of the DDLog, the parties obtain an additive share of $z := xy$ with probability $\epsilon := 1 - \mathsf{poly}(\lambda)$. As such, the error of the HSS scheme is also ϵ.

Security. We prove security via a hybrid argument.

– *Hybrid $\mathcal{H}_0$.* This hybrid consists of the HSS security game instantiated using the construction in Figure 5 with message $\mathbf{x}_b$.

– *Hybrid $\mathcal{H}_1$.* In this hybrid, we replace the evaluation key ek_σ produced by $\mathsf{HSS.Setup}$ (and given to the adversary) with uniformly random value in $\mathbb{Z}_p^3$. It is trivial to verify that this hybrid game is perfectly indistinguishable from $\mathcal{H}_0$.

– *Hybrid $\mathcal{H}_2$.* In this hybrid, we replace the elements of $\mathbb{G}_1$ of each input share produced by $\mathsf{HSS.Share}$ with uniformly random group elements in $\mathbb{G}_1$.

Claim. $\mathcal{H}_2 \approx_c \mathcal{H}_1$ assuming DDH in $\mathbb{G}_1$.

Proof. Notice that in $\mathcal{H}_1$, the adversary $\mathcal{A}$ receives a vector of input shares

$$[\![\mathbf{x}_b]\!]_\sigma := ([\![x_b^{(1)}]\!]_\sigma, \ldots, [\![x_b^{(k)}]\!]_\sigma),$$

distributed as:

$$\left((g_1^{r_1}, g_1^{x_b^{(1)}+\alpha \cdot r_1}, g_2^{r'_1}, g_2^{x_b^{(1)}+\beta \cdot r'_1}), \ldots, (g_1^{r_k}, g_1^{x_b^{(k)}+\alpha \cdot r_k}, g_2^{r'_k}, g_2^{x_b^{(k)}+\beta \cdot r'_k}) \right)$$
$$= \left((g_1^{r_1}, g_1^{x_b^{(1)}} h_1^{r_1}, g_2^{r'_1}, g_2^{x_b^{(1)}} h_2^{r'_1}), \ldots, (g_1^{r_k}, g_1^{x_b^{(k)}} h_1^{r_k}, g_2^{r'_k}, g_2^{x_b^{(k)}} h_2^{r'_k}) \right),$$

where $h_1 := g_1^\alpha$ and $h_2 := g_2^\beta$. In turn, this is distributed identically to:

$$\left((g_1^{r_1}, \ldots, g_1^{r_k}, g_1^{x_b^{(1)}} h_1^{r_1}, \ldots, g_1^{x_b^{(k)}} h_1^{r_k}), (g_2^{r'_1}, \ldots, g_2^{r'_k}, g_2^{x_b^{(1)}} h_2^{r'_1}, \ldots, g_2^{x_b^{(k)}} h_2^{r'_k}) \right),$$

by rearranging the terms. Note that all elements of the input share from $\mathbb{G}_2$ are independent of the $\mathbb{G}_1$ elements (they are computed with different randomness r'_i and using a different secret key β, sampled independently of r_i and α).

Suppose, towards contradiction, that $\mathcal{A}$ distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$ with non-negligible advantage. By a separate hybrid argument, it follows that $\mathcal{A}$ distinguishes between some $(g_1^{r_i}, g_1^{x_b^{(i)}} h_1^{r_i})$ and uniformly random $(u_i, v_i) \in \mathbb{G}_1^2$, for some $i \in [k]$ (note that all elements of $\mathbb{G}_2$ in each input share are still computed as in hybrid $\mathcal{H}_1$). This contradicts the DDH assumption in $\mathbb{G}_1$, since by a straightforward reduction, the adversary is able to distinguish between $(g_1^\alpha, g_1^{r_i}, g_1^{\alpha r_i})$ and (g_1^α, u_i, v_i). □

– *Hybrid $\mathcal{H}_3$.* In this hybrid, we replace the elements of $\mathbb{G}_2$ of each input share produced by $\mathsf{HSS.Share}$ with uniformly random group elements in $\mathbb{G}_2$.

Claim. $\mathcal{H}_3 \approx_c \mathcal{H}_2$ assuming DDH in $\mathbb{G}_2$.

Proof. The proof follows the same strategy as in the proof of the previous claim, except that now the adversary contradicts the DDH assumption in $\mathbb{G}_2$. □

At this point, we have proven that under the SXDH assumption, $\mathcal{A}$ cannot distinguish between $[\![\mathbf{x}_b]\!]_\sigma$ and uniform tuple over $(\mathbb{G}_1^2 \times \mathbb{G}_2^2)^k$, with better than negligible advantage. In turn, it follows that $\mathcal{A}$'s distinguishing advantage between $[\![\mathbf{x}_0]\!]_\sigma$ and $[\![\mathbf{x}_1]\!]_\sigma$ is negligible, which concludes the proof. ■

5.3.2 Random "DDLog" procedure In Figure 5, correctness of the output shares depends on the correctness of the DDLog procedure. However, in cyclic groups $\mathbb{G}$, the DDLog procedure has an inherent $1/\mathsf{poly}(\lambda)$ error [BGI16, DKK20], which the NIDLS framework overcomes by using specific groups $\mathbb{G}$ (e.g., $\mathbb{Z}_{n^2}^*$) and requiring different assumptions (e.g., DCR). This is why Figure 5 only achieves $\epsilon = 1 - 1/\mathsf{poly}(\lambda)$ correctness, in general. The crucial observation we make here is that the DDLog procedure has *no error* when given multiplicative shares of $g^0 \in \mathbb{G}$ and, moreover, because we additionally only require uniformly random payloads in the case where the parties hold multiplicative shares of g^u, for some $u \neq 0$, we can construct a trivial algorithm "DDLog" procedure using just a PRF, as we show in the following lemma.

Lemma 3 (Random Distributed Discrete Logarithm)**.** *Let $\mathbb{G}$ be an arbitrary cyclic group with generator g and $1 \leq M \ll |\mathbb{G}|$ be an integer. There exists an efficient algorithm* DDLog *satisfying Definition 4 such that:*

(1) For all elements $h_A, h_B \in \mathbb{G}$ where $h_A \cdot h_B = g^0$,

$$\Pr\Big[\langle z \rangle_A - \langle z \rangle_B = 0 \quad : \quad \langle z \rangle_\sigma := \mathsf{DDLog}((\mathbb{G}, g, M), h_\sigma),\ \forall \sigma \in \{A, B\} \Big] = 1;$$

(2) For all $h_A, h_B \in \mathbb{G}$ where $h_A \cdot h_B \neq g^0$, it holds that for all $\sigma \in \{A, B\}$:

$$\left\{ (\langle z \rangle_\sigma, h_{1-\sigma}) \;\middle|\; \langle z \rangle_\sigma := \mathsf{DDLog}((\mathbb{G}, g, M), h_\sigma) \right\} \approx_c \left\{ (\langle z \rangle_\sigma, h_{1-\sigma}) \;\middle|\; \langle z \rangle_\sigma \leftarrow\!\!\$\, \mathbb{Z}_M \right\}.$$

Proof. We construct DDLog satisfying the two required properties using any PRF family $F\colon \{0,1\}^\lambda \times \mathbb{G} \to \mathbb{Z}_M$. Define DDLog as $((\mathbb{G}, g, M), h_\sigma) \mapsto F_K((h_\sigma)^{-1^\tau})$,[9] where K is a public uniformly random PRF key. Then, (1) the output of DDLog consists of pseudorandom shares of zero whenever $h_A \cdot h_B = g^0$ and (2) the output of DDLog for all non-zero values is uniformly random in $\mathbb{Z}_M$ (thus satisfying the second property).

To see (1), note that if $h_A \cdot h_B = g^0$, then it holds that $h_A = h_B^{-1}$, which in turn implies that $F_K(h_A) - F_K(h_B) = 0$, with probability 1.

To see (2), note that if $h_A \cdot h_B = g^u$, for some non-zero u with high (pseudo)entropy independent of the PRF key K, then DDLog outputs two pseudorandom and independent elements of $\mathbb{Z}_M$, which guarantees computational indistinguishability by the security of the PRF. ■

Corollary 2. *Under the SXDH assumption over a bilinear group, there exists an instantiation of Figure 4 with random payloads (cf. Definition 14) in the message space $\mathbb{Z}_M$, for any integer M, with a $O(N^{2/3})$ public key size, $O(N)$ key generation time, $O(N^{4/3})$ key derivation time, and $O(N^{5/3})$ full domain evaluation time, where $O(\cdot)$ hides a factor of $\mathsf{poly}(\lambda, \log|\mathcal{R}|)$.*

Proof. The construction consists of Figure 4 instantiated with a multiplicative-output NIM (Definition 10) and the degree-2 HSS construction from SXDH (Figure 5) using the modified DDLog from Lemma 3.

The public key size, key generation time, and evaluation time follows from the proof of Corollary 1. Then, by Lemma 3 we immediately get correctness and a random payload on the non-zero coordinate. However, to additionally ensure pseudorandomness of the payload given the PRF key K, the parties must set their payload to a uniformly random scalar. That is, party σ sets the non-zero coordinate to be Δ_σ (for some uniformly random $\Delta_\sigma \leftarrow\!\!\$\, \mathbb{Z}_p$, where p is the prime order of $\mathbb{G}$). Then, the parties obtain multiplicative shares of a uniformly random value in $\Delta_A \cdot \Delta_B \in \mathbb{Z}_p$ which guarantees the resulting PRF output is pseudorandom.

■

Remark 5 (Guaranteeing a non-zero output). We remark that because the payload is uniformly random in $\mathbb{Z}_M$, it may be zero with noticeable probability if M is small. To guarantee a negligible probability of the payload being zero, we can choose the $M \geq \Omega(2^\lambda)$ so as to ensure the output is non-zero with all but negligible probability. In particular, the DDLog procedure of Lemma 3 does not require M to be polynomial in the security parameter.

6 Generalization to Succinct Multi-Key HSS

In this section, we show that we can generalize the ideas behind our NIDPF construction to construct succinct, two-party "multi-key" HSS for a restricted class of computations. In particular, with multi-key HSS [XW23, CDH+25], two parties can provide inputs to a computation without having to agree on a common public key ahead of time (similarly to spooky encryption, which is based on multi-key FHE [DHRW16]). We consider a setting where Alice has a large input x and Bob has a short input y, where we will require succinctness in the large input (similarly to Definition 11 and succinct HSS [ARS24]). Then, by exchanging input shares, Alice and Bob can compute secret shares of $P(x, f(y))$, where f is any NC^1 function and P is a constant-degree polynomial.

Abram et al. [ARS24] achieved succinct HSS for a similar, slightly larger computational class which they call "Special RMS" programs. Effectively, this class supports computations of the form

$$P(x_A, x_B, f(y_A, y_B)),$$

where again $f \in \mathsf{NC}^1$ and P is a constant-degree polynomial, and Alice and Bob can each contribute inputs x_σ, y_σ for each computation role. In their construction, they have $x = (x_A, x_B)$ as a "special"

[9] Here, DDLog is implicitly parameterized by the party identifier σ and the global PRF key K. We leave this implicit for readability.

input share and $y = (y_A, y_B)$ as a standard HSS input share. However, since they let the HSS input y be defined by *both* parties, this prevents their construction from being "multi-key" or, in other words, non-interactive. That is, in their construction, the parties need to have a common HSS public key to generate the inputs y_A and y_B used in the evaluation of f through HSS.[10]

We show that we can extend succinct NIM for "special RMS" programs provided that only one party specifies the input to the function f—which is exactly the generalization of our NIDPF construction in Section 5. That is, instead of letting both parties provide inputs to the function f, we consider computations of the form $P(x, f(y_\sigma))$, where only party σ is allowed to input y. However, we still get the succinctness property since the total communication incurred is only $o(|x|) + O(|y_\sigma|)$,[11] which is sublinear in the size of the large input. Formally, we capture the following extension to NIM for computing general functions:

Definition 15 (Extended NIM)**.** *Let ℓ_0, ℓ_1 be integers. An* extended NIM *scheme for a function class $\mathcal{F}$, consists of five efficient algorithms*

$$\mathsf{ExtNIM} = (\mathsf{Setup}, (\mathsf{Encode}_\sigma, \mathsf{Decode}_\sigma)_{\sigma \in \{A,B\}})$$

with the same syntax and security property as defined in Definition 9 except that Decode_σ *takes an additional input $f \in \mathcal{F}$. The algorithms must satisfy the following extended correctness property:*

Extended Correctness. *For all security parameters $\lambda \in \mathbb{N}$, every function $f \in \mathcal{F}$, and every pair of inputs $x, y \in \{0,1\}^{\ell_0} \times \{0,1\}^{\ell_1}$, an extended NIM scheme is said to be correct if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = f(x,y) \quad : \quad \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pe}_A, \mathsf{st}_A) \leftarrow \mathsf{Encode}_A(\mathsf{crs}, x) \\ (\mathsf{pe}_B, \mathsf{st}_B) \leftarrow \mathsf{Encode}_B(\mathsf{crs}, y) \\ \langle z \rangle_A := \mathsf{Decode}_A(\mathsf{crs}, \mathsf{pe}_B, \mathsf{st}_A, f) \\ \langle z \rangle_B := \mathsf{Decode}_B(\mathsf{crs}, \mathsf{pe}_A, \mathsf{st}_B, f) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

We can additionally consider succinctness, analogously to Definition 11; we specify the succinctness directly in the following theorem:

Theorem 4. *Let $\ell_0, \ell_1, \ell_2 \in \mathbb{N}$, let $\mathcal{R}$ be a ring, and let $\mathcal{P}$ be the set of all constant-degree polynomials defined over $\mathcal{R}$. Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval})$ *be a secret-key HSS scheme (cf. Definition 16) for the function class $\mathcal{F}\colon \{0,1\}^{\ell_1} \to \mathcal{R}^{\ell_2}$ and satisfying multiplication by memory shares (cf. Definition 8). Then, there exists an extended NIM (cf. Definition 15) for the class of functions described by $P(x, f(y))$, where $P \in \mathcal{P}$, $f \in \mathcal{F}$, $x \in \mathcal{R}^{\ell_0}$ and $y \in \{0,1\}^{\ell_1}$. Moreover, the size of the public encoding output by* Encode_A *is bounded by:*

$$\mathsf{poly}(\lambda, \log|\mathcal{R}|) \cdot (\ell_0)^\epsilon,$$

for some $0 \leq \epsilon < 1$.

Proof (sketch). The main idea is that Alice, given an HSS input share of y from Bob, as well as the evaluation key ek_A, can locally compute memory shares of $f(y)$ under Bob's secret key sk. Simultaneously, Alice and Bob can use NIM to compute memory shares of x under Bob's secret key sk. More concretely, Alice inputs x and Bob inputs sk, to instantiate non-interactive VOLE [ARS24,BCM+24]. This then allows Alice and Bob to compute $x \cdot f(y)$ via the "multiplication by memory shares" supported by the HSS scheme (recall Definition 8). This immediately generalizes to computing $P(x, f(y))$, for any constant-degree polynomial P. To see this, note that Alice can input the vector $\mathbf{x}$ corresponding to the coefficients of all the monomials of P and Bob can input y for the function $\hat{f}$ that outputs a vector $\mathbf{y} := (1, f(y)^1, f(y)^2, \ldots, f(y)^{d-1})$, where $d \in O(1)$ is the degree of P. Then, $P(x, f(y)) = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{d} x_i \cdot f(y)^{i-1}$. Moreover, when using succinct NIM (Definition 11), Alice's encoding is sublinear in ℓ_0 and Bob's encoding is independent of d. ■

[10] In particular, this approach requires a trusted setup to distribute the evaluation keys to both parties before parties can share their inputs y_A and y_B, respectively.

[11] Ignoring polynomial factors in the security parameter.

Acknowledgments

We thank Srini Devadas and the anonymous reviewers for helpful comments and suggestions. Elette Boyle's research is supported, in part, by AFOSR Award FA9550-21-1-0046 and ERC Project HSS (852952).

References

ADOS22. D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 421–452. Springer, Cham, August 2022.

Ajt96. M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

ARS24. D. Abram, L. Roy, and P. Scholl. Succinct homomorphic secret sharing. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 301–330. Springer, Cham, May 2024.

AS22. D. Abram and P. Scholl. Low-communication multiparty triple generation for SPDZ from ring-LPN. In *PKC 2022, Part I*, *LNCS* 13177, pages 221–251. Springer, Cham, March 2022.

BBC$^+$21. D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776. IEEE Computer Society Press, May 2021.

BBC$^+$24. M. Bombar, D. Bui, G. Couteau, A. Couvreur, C. Ducros, and S. Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$OLE-based multi-party computation for boolean circuits. In *ASIACRYPT 2024, Part VI, LNCS 15489*, pages 69–101. Springer, Singapore, 2024.

BCCD23. M. Bombar, G. Couteau, A. Couvreur, and C. Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In *CRYPTO 2023, Part IV*, *LNCS* 14084, pages 567–601. Springer, Cham, August 2023.

BCG$^+$17. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

BCG$^+$19a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG$^+$19b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Cham, August 2019.

BCG$^+$20a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

BCG$^+$20b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 387–416. Springer, Cham, August 2020.

BCG$^+$21. E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 871–900. Springer, Cham, October 2021.

BCG$^+$22. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022, Part II*, *LNCS* 13508, pages 603–633. Springer, Cham, August 2022.

BCGI18. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BCM$^+$24. D. Bui, G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 88–118. Springer, Cham, May 2024.

BGI15. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT 2015, Part II*, *LNCS* 9057, pages 337–367. Springer, Berlin, Heidelberg, April 2015.

BGI16. E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, *LNCS* 9814, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

BGI17. E. Boyle, N. Gilboa, and Y. Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT 2017, Part II*, *LNCS* 10211, pages 163–193. Springer, Cham, April / May 2017.

BGI19. E. Boyle, N. Gilboa, and Y. Ishai. Secure computation with preprocessing via function secret sharing. In *TCC 2019, Part I*, *LNCS* 11891, pages 341–371. Springer, Cham, December 2019.

BGIK22. E. Boyle, N. Gilboa, Y. Ishai, and V. I. Kolobov. Programmable distributed point functions. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 121–151. Springer, Cham, August 2022.

BKS19. E. Boyle, L. Kohl, and P. Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 3–33. Springer, Cham, May 2019.

BM90. M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO'89*, *LNCS* 435, pages 547–557. Springer, New York, August 1990.

CBM15. H. Corrigan-Gibbs, D. Boneh, and D. 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.

CDD$^+$24. G. Couteau, L. Devadas, S. Devadas, A. Koch, and S. Servan-Schreiber. QuietOT: Lightweight oblivious transfer with a public-key setup. In *ASIACRYPT 2024, Part II, LNCS 15485*, pages 197–231. Springer, Singapore, 2024.

CDH$^+$25. G. Couteau, L. Devadas, A. Hegde, A. Jain, and S. Servan-Schreiber. Multi-key homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/094, 2025.

CMPR23. G. Couteau, P. Meyer, A. Passelègue, and M. Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In *EUROCRYPT 2023, Part III*, *LNCS* 14006, pages 194–224. Springer, Cham, April 2023.

CZ22. G. Couteau and M. Zarezadeh. Non-interactive secure computation of inner-product from LPN and LWE. In *ASIACRYPT 2022, Part I*, *LNCS* 13791, pages 474–503. Springer, Cham, December 2022.

DFL$^+$20. E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica. DORY: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.

DH76. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DHRW16. Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 93–122. Springer, Berlin, Heidelberg, August 2016.

DIJL23. Q. Dao, Y. Ishai, A. Jain, and H. Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In *CRYPTO 2023, Part II*, *LNCS* 14082, pages 315–348. Springer, Cham, August 2023.

DKK20. I. Dinur, N. Keller, and O. Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. *Journal of Cryptology*, 33(3):824–873, July 2020.

DRPS22. E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy*, pages 2450–2468. IEEE Computer Society Press, May 2022.

Ds17. J. Doerner and a. shelat. Scaling ORAM for secure computation. In *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.

GI14. N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, *LNCS* 8441, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

JGB$^+$24. N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. 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.

MPD$^+$24. D. Mouris, C. Patton, H. Davis, P. Sarkar, and N. G. Tsoutsos. Mastic: Private weighted heavy-hitters and attribute-based metrics. Cryptology ePrint Archive, Report 2024/221, 2024.

MST24. D. Mouris, P. Sarkar, and N. G. Tsoutsos. PLASMA: Private, lightweight aggregated statistics against malicious adversaries. *PoPETs*, 2024(3):4–24, July 2024.

OSY21. C. Orlandi, P. Scholl, and S. Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, *LNCS* 12696, pages 678–708. Springer, Cham, October 2021.

RS21. L. Roy and J. Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 687–717, Virtual Event, August 2021. Springer, Cham.

RTPB22. T. Ryffel, P. Tholoniat, D. Pointcheval, and F. R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *PoPETs*, 2022(1):291–316, January 2022.

RZCGP24. M. Rathee, Y. Zhang, H. Corrigan-Gibbs, and R. A. Popa. Private analytics via streaming, sketching, and silently verifiable proofs. *Cryptology ePrint Archive*, 2024.

SGRR19. P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

SSLD22. S. Servan-Schreiber, S. Langowski, and S. Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy*, pages 911–929. IEEE Computer Society Press, May 2022.

VHG23. A. Vadapalli, R. Henry, and I. Goldberg. Duoram: A bandwidth-efficient distributed ORAM for 2- and 3-party computation. In *USENIX Security 2023*, pages 3907–3924. USENIX Association, August 2023.

VSH22. A. Vadapalli, K. Storrier, and R. Henry. Sabre: Sender-anonymous messaging with fast audits. In *2022 IEEE Symposium on Security and Privacy*, pages 1953–1970. IEEE Computer Society Press, May 2022.

WYG$^+$17. F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017.

XW23. P. Xu and L.-P. Wang. Multi-key homomorphic secret sharing from LWE without multi-key HE. In *ACISP 23*, *LNCS* 13915, pages 248–269. Springer, Cham, July 2023.

YJG$^+$23. P. Yang, Z. L. Jiang, S. Gao, J. Zhuang, H. Wang, J. Fang, S. Yiu, and Y. Wu. FssNN: Communication-efficient secure neural network training via function secret sharing. Cryptology ePrint Archive, Report 2023/073, 2023.

YWL$^+$20. K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

Supplementary Material

7 Homomorphic Secret Sharing

Here, we provide the full definition of Homomorphic Secret Sharing (HSS).

Definition 16 (Secret-Key HSS; Adapted from [BGI16, BKS19, DIJL23]). *Let λ be a security parameter, $\mathcal{R}$ be a finite ring, and $\mathcal{F}$ be a class of ℓ input functions defined over $\mathcal{R}$. A (secret key) HSS scheme with message space $\mathcal{R}$ consists of six efficient algorithms* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Output})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B))$. *The randomized setup algorithm takes as input the security parameter. It outputs a secret key* sk *and a pair of HSS evaluation keys* $(\mathsf{ek}_A, \mathsf{ek}_B)$.
- $\mathsf{Share}(\mathsf{sk}, x) \to (\llbracket x \rrbracket_A, \llbracket x \rrbracket_B)$. *The randomized share algorithm takes as input the secret key* sk *and message $x \in \mathcal{R}$. It outputs an input share of x.*
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, f, \llbracket \mathbf{x} \rrbracket_\sigma) \to \langle\!\langle y \rangle\!\rangle_\sigma$. *The deterministic evaluation algorithm takes as input the party identifier $\sigma \in \{A, B\}$, an evaluation key* ek_σ*, function $f \in \mathcal{F}$, and input shares of $\mathbf{x} := (x_1, \ldots, x_\ell)$. It outputs a memory share of $y := f(\mathbf{x})$.*
- $\mathsf{Output}(\sigma, \langle\!\langle y \rangle\!\rangle_\sigma) \to \langle y \rangle_\sigma$. *The deterministic output algorithm takes as input the party identifier $\sigma \in \{A, B\}$ and a memory share of y. It outputs a share of y.*

When $\mathsf{Alg} \in \{\mathsf{Share}, \mathsf{Output}\}$ *is given as input vector of input (or memory) shares, it outputs the vector obtained by evaluating* Alg *on each coordinate of the input vector independently.*

The above algorithms must satisfy correctness and security:

Correctness. *We say the HSS scheme is ϵ-correct, for some $0 < \epsilon \leq 1$, if for all functions $f \in \mathcal{F}$, and for all vectors $\mathbf{x} \in \mathcal{R}^\ell$, it holds that:*

$$\Pr\left[\langle z \rangle_A - \langle z \rangle_B = f(\mathbf{x}) \quad : \quad \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\llbracket \mathbf{x} \rrbracket_A, \llbracket \mathbf{x} \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}) \\ \langle\!\langle y \rangle\!\rangle_\sigma := \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, f, \llbracket \mathbf{x} \rrbracket_\sigma) \\ \langle z \rangle_\sigma := \mathsf{Output}(\sigma, \langle\!\langle y \rangle\!\rangle_\sigma) \end{array} \right] \geq \epsilon - \mathsf{negl}(\lambda).$$

Security. *For every $\sigma \in \{A, B\}$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that for all $\lambda \in \mathbb{N}$ we have that:*

$$\Pr\left[b' = b \quad : \quad \begin{array}{r} (\mathsf{sk}, (\mathsf{ek}_A, \mathsf{ek}_B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbf{x}_0, \mathbf{x}_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{ek}_\sigma) \\ b \leftarrow\!\!\$ \{0,1\} \\ (\llbracket \mathbf{x}_b \rrbracket_A, \llbracket \mathbf{x}_b \rrbracket_B) \leftarrow \mathsf{Share}(\mathsf{sk}, \mathbf{x}_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \llbracket \mathbf{x}_b \rrbracket_\sigma) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where for all $b \in \{0,1\}$, $\mathbf{x}_b \in \mathcal{R}^\ell$ and $k = k(\lambda) \in$ $\mathsf{poly}(\lambda)$.