Locality-Preserving Hashing for Shifts with Connections to Cryptography[*]

Elette Boyle[†] Itai Dinur[‡] Niv Gilboa[§] Yuval Ishai [¶] Nathan Keller[||]
Ohad Klein[**]

January 9, 2022

Abstract

Can we sense our location in an unfamiliar environment by taking a sublinear-size sample of our surroundings? Can we efficiently encrypt a message that only someone physically close to us can decrypt? To solve this kind of problems, we introduce and study a new type of hash functions for finding shifts in sublinear time. A function $h : \{0,1\}^n \to \mathbb{Z}_n$ is a (d, δ) *locality-preserving hash function for shifts* (LPHS) if: (1) h can be computed by (adaptively) querying d bits of its input, and (2) $\Pr\left[h(x) \neq h(x \ll 1) + 1\right] \leq \delta$, where x is random and $\ll 1$ denotes a cyclic shift by one bit to the left. We make the following contributions.

- **Near-optimal LPHS via Distributed Discrete Log.** We establish a general two-way connection between LPHS and algorithms for *distributed discrete logarithm* in the generic group model. Using such an algorithm of Dinur et al. (Crypto 2018), we get LPHS with near-optimal error of $\delta = \tilde{O}(1/d^2)$. This gives an unusual example for the usefulness of group-based cryptography in a post-quantum world. We extend the positive result to non-cyclic and worst-case variants of LPHS.
- **Multidimensional LPHS.** We obtain positive and negative results for a multidimensional extension of LPHS, making progress towards an optimal 2-dimensional LPHS.
- **Applications.** We demonstrate the usefulness of LPHS by presenting cryptographic and algorithmic applications. In particular, we apply multidimensional LPHS to obtain an efficient "packed" implementation of *homomorphic secret sharing* and a sublinear-time implementation of *location-sensitive encryption* whose decryption requires a significantly overlapping view.

[*] This is a full version of [9].

[†] IDC Herzliya, Israel and NTT Research, USA. `elette.boyle@idc.ac.il`

[‡] Ben-Gurion University, Be'er Sheva, Israel. `dinuri@cs.bgu.ac.il`

[§] Ben-Gurion University, Be'er Sheva, Israel. `gilboan@bgu.ac.il`

[¶] Technion, Haifa Israel. `yuvali@cs.technion.ac.il`

[||] Bar-Ilan University, Ramat Gan, Israel. `nathan.keller27@gmail.com`

[**] Bar-Ilan University, Ramat Gan, Israel. `ohadkel@gmail.com`

Contents

1 Introduction

A *locality-preserving hash function* [35, 33] is a distance-respecting mapping from a complex input space to a simpler output space. Inspired by recent results in cryptography, we study a new kind of locality-preserving hash functions that map strings to integers while respecting the *shift* distance between pairs of input strings with high probability. A distinctive feature of these hash functions is that they can be computed in *sublinear time* with low error probability.

Why shifts? Why sublinear? Our hash functions for shifts can be thought of as *sublinear-time location sensors* that measure a relative position in an unfamiliar environment by taking a sublinear-size sample of the surroundings. This can apply in a variety of settings. For instance, "surroundings" may refer to a local view of an unexplored territory, a long string such as a DNA sequence, an external signal such as a GPS synchronization sequence, a digital document such as big pdf file or a virtual world, or a huge mathematical object such as a cryptographic group. See [7, 36] for applications of shift finding to GPS synchronization, image alignment, motion estimation, and more.[1] We will discuss additional cryptographic and algorithmic applications in Section 1.1.3 below. We are motivated by scenarios in which the local view contains an enormous amount of relevant information that cannot be naively sub-sampled or compressed. This calls for *sublinear-time* solutions.

Simple shift-finding solutions. To motivate the new primitive, consider the following simple shift-finding problem. An n-bit string x is picked uniformly at random, and then cyclically shifted by s bits the left, for some $0 \leq s < n$. Let y be the resulting string. For instance, x, y may be obtained by measuring the same periodic signal at different phases. We write $y = x \lll s$. The shift-finding problem is to find the shift amount s given x and y.

In a centralized setting, where x and y are both given as inputs, it is easy to solve the problem in sublinear time (with small error probability), querying only $\tilde{O}(n^{1/2})$ bits of the input, by matching substrings of x of length $\ell = O(\log n)$ starting at positions $1, 2, \ldots, \sqrt{n}$ with length-ℓ substrings of y whose starting position is a multiple of $\sqrt{n}$. (This is a simplified version of a noise-resilient algorithm from [7].) This algorithm is nearly optimal, since any shift-finding algorithm for an unbounded shift amount s should read $\Omega(\sqrt{n})$ bits of the input [8].

In a distributed setting, a natural goal is to design a *sketching* algorithm that compresses a single input into a short sketch, such that given the sketches of x and y one can recover s with high probability. Note that the previous centralized algorithm does imply such a sublinear-size sketch, but only with $\tilde{O}(\sqrt{n})$ output size, which is far from optimal. Instead, one could use the following classical approach [21]: let the sketch of x be an integer $0 \leq z_x < n$ that minimizes $x \lll z_x$ (viewed as an n-bit integer), and similarly for y. It can be easily seen that $s = z_x - z_y \mod n$ whenever the minimum is uniquely defined.

The logarithmic sketch size of this simple solution is clearly optimal. Moreover, it realizes something even stronger than sketching: a hash function $h : \{0,1\}^n \to \mathbb{Z}_n$ that respects cyclic shifts in the sense that for a random input x, we have $h(x) = h(x \lll s) + s$ except with small probability. That is, shifting the input by s changes the output by s in the same direction. This is

[1] While previous related works study a *noise tolerant* variant of shift distance, which arises naturally in the applications they consider, in this work we focus on the simpler noiseless case. Beyond theoretical interest, the simpler notion is motivated by applications. For instance, a local view of a digital document or a mathematical object is noiseless. The noisy case is studied in a follow-up work [10], which obtains nearly tight bounds on the (sublinear) amount of random noise that can be tolerated.

useful for applications. For instance, given t hashes $z_i = h(y_i)$, where $y_i = x \ll s_i$ for $i = 1, \ldots, t$, one can easily compute in time $\tilde{O}(t)$ the relative offsets of all y_i.

The main downside of the above hashing-based solution compared to the centralized algorithm is its linear running time. A natural question is whether one can enjoy the best of both worlds:

> *Can we combine the sublinear running time of the centralized algorithm with the optimal sketch size and locality-sensing features of the hashing-based solution?*

1.1 Our Contribution

We initiate a study of hashing-based solutions to the shift-finding problem. We capture such solutions via the following notion of locality-preserving hash function for shifts.

Definition 1.1. *A function $h : \{0,1\}^n \to \mathbb{Z}_n$ is a (d, δ)* locality-preserving hash function for shifts *(LPHS) if: (1) h can be computed by (adaptively) querying d bits of its input, and (2)* $\Pr[h(x) \neq h(x \ll 1) + 1] \leq \delta$*, where x is random and $\ll 1$ denotes a cyclic shift by one bit to the left.*

Note that, by a union bound, an LPHS as above satisfies $\Pr[h(x) \neq h(x \ll s) + s] \leq s \cdot \delta$ for any shift amount $0 \leq s < n$. Thus, an LPHS has a better accuracy guarantee for smaller shifts. Intuitively, an LPHS can be thought of as a sublinear-time computable location identifier that suffices (with high probability) for determining the exact relative location with respect to adjacent identifiers.

Other LPHS flavors. The above notion of LPHS addresses the basic shift-finding problem as discussed above, but is limited in several important ways: it only considers cyclic shifts and 1-dimensional inputs, and it only guarantees average-case correctness for uniformly random inputs. To address these limitations, we additionally consider other flavors of the basic LPHS notion defined above that are more suitable for applications. These include a *non-cyclic* variant, where instead of $x \ll 1$ we remove the leftmost bit of x and add a random bit on the right; a *k-dimensional* variant, where the input is a k-dimensional matrix and the output is in $\mathbb{Z}_n^k$; and a *worst-case* variant where the quantification is over an arbitrary x that is "far from periodic" and the probability is over the choice of h. (The latter variant better corresponds to the typical notion of a randomized hash function.) The applications we present crucially depend on these extensions.

1.1.1 Near-Optimal LPHS via Distributed Discrete Log

We establish a general two-way connection between LPHS and algorithms for the *distributed discrete logarithm* (DDL) problem [12]. Before explaining this connection, we start with relevant background.

The traditional discrete logarithm (DL) problem is parameterized by a cyclic group $\mathbb{G}$ of order n with a generator g, where n is typically a large prime. The challenge is to recover a random $u \in \mathbb{Z}_n$ from g^u. Many cryptographic applications rely on the conjectured intractability of the DL problem in special types of groups, including subgroups of $\mathbb{Z}_p^*$ and certain families of elliptic curves.

The DDL problem is a distributed variant of the DL problem that was recently introduced in the context of group-based homomorphic secret sharing [12]. In DDL there are two parties, where the first party's input is g^u for a random u, and the second party's input is g^{u+s} where $s \in \{0, 1\}$ (more generally, s can be a small integer). The goal is for each party to locally output an integer, such that the difference between the two outputs is s. One can assume without loss of generality that the two parties run the same algorithm.

Note that a DL algorithm can be used to perfectly solve the DDL problem. However, this is computationally infeasible in a cryptographically hard group, where n is enormous. Instead, a DDL algorithm uses a bounded running time (typically polylogarithmic in the group order n) to obtain the correct difference except with error probability δ. For instance, the initial solution proposed in [12] uses a pseudorandom function to mark each group element as "distinguished" with probability δ, and makes each party, on input v, output the smallest $z \geq 0$ such that $v \cdot g^z$ is distinguished. The (expected) running time of this algorithm is roughly $1/\delta$, and the error probability is δ (corresponding to the case where $s = 1$ and g^u is distinguished).

The DDL problem can be related to the LPHS problem (over a non-binary alphabet) by associating each party's DDL input v with an LPHS input consisting of the sequence of group elements $x = (v, gv, g^2v, \ldots, g^{n-1}v)$. Indeed, multiplication of the DDL input by g corresponds to a cyclic shift of x by one symbol to the left. We formalize this intuition by proving a general two-way relation between LPHS and DDL algorithms in the *generic group model* [41], where group elements are assigned random labels and the algorithm is only given oracle access to the group operation.[2]

The applications we derive from the above connection give an unusual example for the usefulness of results on group-based cryptography in a post-quantum world. Indeed, all traditional applications of group-based cryptography are subject to quantum polynomial-time attacks using Shor's algorithm, and are thus useless in a post-quantum world. If scalable quantum computers become a reality, cryptosystems that are "quantum broken" will become obsolete. In contrast, sublinear-time classical algorithms will still be meaningful even in a post-quantum world.

LPHS constructions. The simple DDL algorithm from [12] corresponds to a (d, δ)-LPHS where $\delta = \tilde{O}(1/d)$. Another simple LPHS construction with similar parameters, implicit in a DDL algorithm from [13], makes a simple use of MinHash [15]: let $h(x)$ output the index i, $1 \leq i \leq d$, that minimizes the value of a MinHash applied to a polylogarithmic-length substring of x starting from x_i.

It is tempting to conjecture that the above simple LPHS constructions are near-optimal, in the sense that $\delta = o(1/d)$ is impossible. It turns out, however, that a quadratic improvement can be obtained from a recent optimal DDL algorithm due to Dinur et al. [23]. Their *Iterated Random Walk (IRW)* algorithm, whose self-contained description appears in Appendix A is based on a carefully chosen sequence of random walks in the group. It can be viewed as a non-trivial extension of Pollard's classical "kangaroo" DL algorithm [39], which runs in time $\tilde{O}(\sqrt{n})$ and has low space complexity. Applying the LPHS vs. DDL connection to the positive and negative results on DDL from [23], we get the following theorem.

Theorem 1.2 (Near-optimal LPHS). *There exist (d, δ)-LPHS with: (1) $\delta = \tilde{O}(d^{-2})$ for $d \leq \sqrt{n}$, and (2) $\delta = n^{-\omega(1)}$ for $d = \tilde{O}(\sqrt{n})$. Furthermore, both "$\delta = \tilde{O}(d^{-2})$" in (1) and "$d = \tilde{O}(n^{1/2})$" in (2) are optimal up to polylogarithmic factors.*

Interestingly, any sublinear-time LPHS must inherently make adaptive queries to its input. Adaptive queries are unusual in the context of sublinear metric algorithms, but were previously

[2] Specifically, in Section 3 we prove that any LPHS gives a DDL algorithm (in the generic group model) with similar parameters, while any DDL algorithm gives an LPHS with a negligible cost in error probability assuming $d = O(n^{1/4})$ and n is prime.

used in sublinear algorithms for approximating edit distance [40, 17, 31]. A random walk technique was recently used in [34] to obtain a sublinear-time embedding of edit distance to Hamming distance.

Additional variants. We prove similar bounds for the *worst-case* and *non-cyclic variants* of LPHS, which are motivated by the applications we discuss in Section 1.1.3. The result for the worst-case variant is obtained via a general reduction, and inevitably excludes a small set of inputs that are close to being periodic. The result for the non-cyclic case does not follow generically from the cyclic case (except when $d < n^{1/3}$), and requires a special analysis of the IRW algorithm [23]. Also, in this case only (1) holds, since the error probability of a non-cyclic LPHS must satisfy $\delta = \Omega(1/n)$ regardless of d. In fact, whereas $d = \sqrt{n}$ is the hardest case for Theorem 1.2 in the non-cyclic case (in that (1) for $d = \sqrt{n}$ easily implies (1) for smaller d), it is the easiest for the cyclic case (in that it is implied by the simpler algorithm of Pollard [39]).

In the context of *sketching* for shifts, the above results imply solutions that simultaneously achieve near-optimal sketch size of at most polylog(n), near-optimal running time of $\tilde{O}(\sqrt{n})$, and negligible error probability, for both cyclic and non-cyclic shifts, and for arbitrary "far-from-periodic" inputs.

1.1.2 Multidimensional LPHS

Viewing LPHS as a location identifier, it is natural to consider a generalization to two dimensions and beyond. Indeed, a 2-dimensional (non-cyclic) LPHS can be useful for aligning or sequencing local views of a big 2-dimensional (digital or physical) object. A k-dimensional LPHS maps a k-dimensional matrix (with entries indexed by $\mathbb{Z}_n^k$) into a vector in $\mathbb{Z}_n^k$ so that (cyclically) shifting the input matrix by 1 in axis i changes the output vector by the unit vector e_i, except with δ error probability. As before, this guarantees recovering an arbitrary shift vector with error probability that scales with the ℓ_1 norm of the shift.

Upper bounds. In the 2-dimensional case, the algorithm can be viewed as allowing two non-communicating parties, who are given points (x, y) and $(x + \alpha, y + \beta)$ in the same random for unknown $\alpha, \beta \in \{0, 1\}$, to maximize the probability of synchronizing at the same point, where only d queries are allowed. A straightforward approach is to use a MinHash algorithm in which the parties take the minimal hash value computed on values of a $d^{1/2} \times d^{1/2}$ matrix of elements beginning at the location of each party, resulting in a (d, δ)-LPHS with $\delta = \tilde{O}(d^{-1/2})$. A better error bound of $\delta = \tilde{O}(d^{-2/3})$ can be obtained by combining the application of MinHash on one axis with the application of the aforementioned optimal IRW algorithm on the other axis.

We present three improved algorithms in Section 4. The simplest of those, with a bound of $\delta = \tilde{O}(d^{-4/5})$, is obtained by applying IRW on both axes. A natural idea is to first synchronize on the column; then, synchronizing on the row is easy, using the 1-dimensional IRW algorithm. To synchronize on the column, the parties perform the 1-dimensional IRW algorithm with d/d' 'horizontal' steps, where the information used to determine each step is distilled from the column in which the current point resides by using an IRW algorithm with d' 'vertical' steps. The analysis in Section 4 (Lemma 4.2) shows that the bound $\delta = \tilde{O}(d^{-4/5})$ is obtained for the parameter $d' = d^{3/5}$.

Our main upper bound is obtained by a more complex algorithm, which – perhaps, surprisingly – does not rely on the optimal 1-dimensional IRW algorithm at all. We prove:

Theorem 1.3. *For $n = \tilde{\Omega}(d)$, there is a 2-dimensional (d, δ)-LPHS with $\delta = \tilde{O}(d^{-7/8})$. There is also a non-cyclic 2-dimensional LPHS with the same parameters.*

The algorithm we use to prove Theorem 1.3 consists of three stages. After each stage the parties either converge to the same location, in which case they stay synchronized to the end of the algorithm, or the two walks are within a bounded distance from each other. Stage 1 begins with a distance of at most 1 on each axis, and is a straightforward application of the 2-dimensional MinHash-based algorithm. Stage 2 begins with a distance of at most $\sqrt{d}$ on each axis and uses an asymmetric deterministic walk that consists of $\sqrt{d}$ horizontal steps of size $\sim d^{1/4}$, where each step is pseudo-randomly determined by information distilled from a vertical walk of length $\sqrt{d}$ and step sizes $\sim d^{1/4}$. Stage 3 begins with a distance of at most $d^{3/4}$ on each axis and uses a different deterministic walk. This time, the horizontal steps are of size 1, while the vertical steps are of size about $d^{3/8}$, and unlike all other steps, can be negative. The analysis of the algorithm relies on martingale techniques.

Finally, we present another 2-dimensional LPHS algorithm, which seems harder to analyze, but for which we conjecture that the error rate is at most $\tilde{O}(d^{-1})$. This bound is essentially the best one can hope for given the lower bound discussed below. The idea behind this algorithm is to not treat the axes separately but rather to perform a series of deterministic walks over $\mathbb{Z}^2$, with step sizes of about $d^{1/4}, d^{3/8}, d^{7/16}, \ldots, \sqrt{d}/2$. Our experiments suggest that the error rate of this algorithm is indeed $\tilde{O}(d^{-1})$. However, the analysis (and especially deterministic resolution of cycles in the random walk) is quite involved, and settling our conjecture is left open for future work. Nevertheless, the heuristic algorithm can be used in cryptographic applications (such as packed homomorphic secret sharing which is described next) without compromising their security. Moreover, the worst-case scenario in which the error is larger than predicted by our experiments can be easily detected by applications.

Lower bound. We complement our positive results by proving the following lower bound, extending in a nontrivial way the lower-bound for 1-dimensional LPHS obtained from [23] via the DDL connection.

Theorem 1.4. *For $n = \Omega(d^{2/k})$, any k-dimensional (d, δ)-LPHS satisfies $\delta = \Omega(d^{-2/k})$.*

The intuition is related to the birthday bound. A k-dimensional box with edge length $d^{2/k}$ contains d^2 points. If the k-dimensional shift is uniform within this box, we expect the d queries of the two parties not to intersect with constant probability, implying that $\delta = \Omega(1)$. Given this, the proof for smaller shifts follows by a union bound. While the intuition is simple, is it not clear how to directly apply the birthday bound, and the formal proof is based on an argument involving Minkowski sums and differences of sets in $\mathbb{R}^k$.

1.1.3 Applications

We present several cryptographic and algorithmic applications that motivate different variants of LPHS, exploiting both the functionality and the sublinearity feature. All of the applications can benefit from our 2-dimensional LPHS constructions, and most require the non-cyclic, worst-case variant. See Section 6 for a taxonomy of the LPHS variants required by different applications.

Packed homomorphic secret sharing. We demonstrate a cryptographic application of k-*dimensional* LPHS in trading computation for communication in group-based homomorphic secret sharing (HSS). In a nutshell, we use LPHS to further improve the succinctness of the most succinct approach for simple "homomorphic" computations on encrypted data, by packing 2 or more plaintexts into a single ciphertext. Compared to competing approaches (see, e.g., [1, 37] for recent examples), group-based packed HSS can provide much better succinctness and client efficiency. The key technical idea is to use a non-cyclic k-dimensional LPHS for implementing a k-dimensional variant of DDL, where k independent group generators are used for encoding k small integers by a single group element, and where multiplication by each generator is viewed as a (non-cyclic) shift in the corresponding direction. This k-dimensional generalization of DDL can be potentially useful for other recent applications of DDL that are unrelated to HSS [26, 29, 14]. See Section 6.1 for a detailed discussion of this cryptographic application of LPHS along with the relevant background.

Location-sensitive encryption. We apply LPHS to obtain a sublinear-time solution for *location-sensitive encryption* (LSE), allowing one to generate a public ciphertext that can only be decrypted by someone in their (physical or virtual) neighborhood. Here proximity is defined as having significantly overlapping views, and security should be guaranteed as long as a non-overlapping view is sufficiently unpredictable. The above goal can be reduced to realizing a sublinear-time computable fuzzy extractor [24][3] for shift distance. Obtaining such fuzzy extractors from LPHS constructions requires an understanding of their behavior on entropic sources. It turns out that even for high-entropy sources, an LPHS provides no unpredictability guarantees. We get around this problem by defining a hash function that combines the output of an LPHS with a local function of the source. Using this approach, we obtain a sublinear-time LSE whose security holds for a broad class of mildly unpredictable sources. See Section 6.2 for the LSE application of LPHS.

Algorithmic applications. As discussed above, algorithmic applications of LPHS follow from the vast literature on sketching, locality-preserving and locality-sensitive hashing, and metric embeddings. Indeed, our different LPHS flavors can be roughly viewed as probabilistic isometric embeddings of certain shift metrics into a Euclidean space. Thus, for example, an LSH for the same shift metric can potentially follow by concatenating the LPHS with an LSH from the literature. However, some care should be taken in applying this high-level approach. One issue is the average-case nature of LPHS, which makes the failure probability input-dependent. We get around this via a worst-case to average-case reduction that restricts the input space to “non-pathological” inputs that are far from periodic. Another issue is that LPHS provides no explicit guarantees for inputs that are too far apart. We get around this by using the fact that an LPHS must have a well-spread output distribution on a random input. As representative examples, we demonstrate how LPHS can be applied in the contexts of communication complexity and LSH-based near-neighbor data structures for shifts. The algorithmic applications of LPHS are discussed in Section 6.3.

Open questions. Our work leaves several open questions. The main question, on which we make partial progress, is obtaining optimal parameters for k-dimensional LPHS. Other questions concern the optimality of the LPHS-based approach to sketching. A negative result from [23], which can be used to rule out sublinear-time LPHS with non-adaptive queries, in fact holds even for sketching. Do LPHS-based sketches also provide an optimal tradeoff between sketch size and error probability?

Organization. In Section 2 we introduce necessary preliminaries and notation, including the definition of Locality-Preserving Hash functions for Shifts (LPHS), and simple properties of and relations between LPHS variants. In Section 3, we present the general two-way connection between LPHS and algorithms for distributed discrete logarithm in the generic group model. In Section 4 we provide our results on multidimensional LPHS. Section 6 contains applications of LPHS. And,

[3] There are two differences from the standard notion of fuzzy extractors: the “distance” is not a strict metric, and the notion of unpredictability needs to ensure that the source is far from periodic with high probability.

for completeness, in Appendix A, we provide LPHS results based on the Iterative Random Walk algorithms of Dinur, Keller, and Klein [23].

2 Preliminaries

We denote by $\mathbb{Z}_n$ the additive group of integers modulo n. We will typically consider strings of length n over an alphabet $\Sigma_b = \{0,1\}^b$, indexing string entries by $i \in \mathbb{Z}_n$. We will use the notation $x^{(b)}$ when we want to make the alphabet size explicit. When the alphabet is binary or when b is clear from the context, we will typically omit the superscript and use the notation x. For $x^{(b)} \in \Sigma_b^n$ we denote by $x^{(b)}[i]$ the i'th symbol of $x^{(b)}$, for $i \in \mathbb{Z}_n$.

We denote by $x^{(b)} \lll r$ the cyclic rotation of $x^{(b)}$ by r symbols to the left, namely the string $y^{(b)}$ defined by $y^{(b)}[i] = x^{(b)}[i+r]$ with addition modulo n. We will also consider a *non-cyclic shift*, denoted by $x^{(b)} \ggg r$, where the r leftmost symbols of x are chopped and r *random* symbols are added on the right. Note that unlike the cyclic shift operator, which is deterministic, the non-cyclic version is randomized. We use $\Delta(x,y)$ to denote the Hamming distance between x and y, namely the number of symbols i in which $x[i]$ and $y[i]$ differ.

We use the notation $x^{(2,b)}$ to denote a 2-dimensional string (i.e., matrix) over alphabet Σ_b and denote by $x^{(2,b)}[i,j]$ its (i,j) entry. We denote by $x^{(2,b)} \ll (r_1, r_2)$ the cyclic rotation of $y^{(2,b)}$ by r_1 symbols to the left on the first axis and r_2 symbols to the left on the second axis. That is, $y = x^{(2,b)} \ll (r_1, r_2)$ is defined by $y[i_1, i_2] = x^{(2,b)}[i_1 + r_1, i_2 + r_2]$, where addition is modulo n. We will also consider the natural k-dimensional generalization $x^{(k,b)} \ll (r_1, r_2, \dots, r_k)$ and its non-cyclic variant $x^{(k,b)} \lll (r_1, r_2, \dots, r_k)$.

2.1 Locality-Preserving Hash Functions for Shifts

We now define our main notion of LPHS and some of its useful variants.

Definition 2.1 (LPHS: main variants). *Let $h\colon \Sigma_b^n \to \mathbb{Z}_n$ be a function. We say that h is a (cyclic)* (d,δ)-LPHS *if h can be computed by making d* adaptive *queries (of the form $x[i]$) to an input $x \in \Sigma_b^n$ and moreover $\Pr_{x \in_R \Sigma_b^n}[h(x) \neq h(x \ll 1) + 1] \leq \delta$.*

We will consider the following modifiers (that can be combined in a natural way):

- **Non-cyclic** *LPHS: replace $\mathbb{Z}_n$ by $\mathbb{Z}$ and $x \ll 1$ by $x \lll 1$;*
- **k-dimensional** *LPHS: let x be a random k-dimensional string $x \in \Sigma_b^{\mathbb{Z}_n^k}$ and $h\colon \Sigma_b^{\mathbb{Z}_n^k} \to \mathbb{Z}_n^k$. We require that $\Pr_x[h(x) \neq h(x \ll e_i) + e_i] \leq \delta$ for every* unit vector $e_i \in \mathbb{Z}_n^k$.

We will sometimes make more parameters explicit in the notation. For instance, an (n,b,d,δ)-LPHS is a (d,δ)-LPHS $h\colon \Sigma_b^n \to \mathbb{Z}_n$.

Remark (On computational complexity). A (d,δ)-LPHS $h\colon \Sigma_b^n \to \mathbb{Z}_n$ can be viewed as a *depth-d decision tree* over n input variables taking values from the alphabet Σ_b. In all of our positive results, h is *semi-explicit* in the sense that it can be realized by a *randomized* polynomial-time algorithm having oracle access to the input x. (In fact, our algorithms can be implemented in probabilistic $\tilde{O}(d)$ time.) Here the same randomness for h is used in the two invocations $h(x)$ and $h(x \ll 1)$. Alternatively, our positive results imply a deterministic h in a non-uniform setting. Our negative results apply to the existence of h with the given parameters, irrespective of the computational complexity of generating it.

Remark (Worst-case vs. average-case LPHS). Our default notions of LPHS assume a uniformly random input x. While this suffices for some applications, a worst-case notion of LPHS is more desirable for most applications. Since shift detection is impossible for highly periodic inputs (such as the all-0 string), or even for approximately periodic in the context of sublinear-time algorithms, the notion of worst-case LPHS is restricted to a set of "typical" inputs that are far from being periodic. Our notion of "typical" is very broad and arguably captures essentially all naturally occurring inputs in our motivating applications. In Section 5 we present a simple reduction of this worst-case flavor of LPHS to our default notion of LPHS for random inputs. This applies both to the cyclic and non-cyclic variants. The reduction only incurs a polylogarithmic loss in the parameters. Note that, unlike our main notion of LPHS, here it is inherent that the function h be randomized. A useful related byproduct of the worst-case variant is that the failure events of two independently chosen h_1 and h_2 are independent. This is useful for algorithmic applications of LPHS.

For some applications, we will be interested in the following additional LPHS variants.

Definition 2.2 (LPHS: additional variants)**.** *We consider the following additional variants of the main notion of LPHS from Definition 2.1.*

- **Shift-bounded** *LPHS with shift bound R: requires that for every $1 \leq r \leq R$, we have*

$$\Pr_{x \in_R \Sigma_b^n} [h(x) \neq h(x \lll r) + r] \leq \delta,$$

and similarly for the non-cyclic case.

- **Las Vegas** *LPHS: allow h to output $\perp$ with probability $\leq \delta$, and require that h never fail in the event that neither of its two invocations outputs $\perp$.*

A generic way of obtaining a Las Vegas LPHS h' from an LPHS h is to invoke h on both x and $x' = x \lll 1$ and output $\perp$ if $h(x) \neq h(x') + 1$. However, an extension of this to a *shift-bounded* LPHS is inefficient, since it requires invoking h on $x \lll r$ for every $0 \leq r \leq R$. In Appendix A.3.3 (Lemma A.5) we show that an optimal shift-bounded LPHS admits a Las Vegas variant with better parameters.

2.2 Simple LPHS Properties and Reductions

In this section we prove simple properties of LPHS and reductions between different LPHS variants that will be useful in what follows.

The following lemma is immediate.

Lemma 2.3 (Conversion between non-cyclic and cyclic LPHS)**.** *Any k-dimensional non-cyclic (n, b, d, δ)-LPHS that queries the last symbol $(n-1) \cdot e_i \in \mathbb{Z}_n^k$ in every dimension with probability 0 gives a cyclic (n, b, d, δ)-LPHS and vice versa.*

The following lemma shows the usefulness of LPHS for detecting arbitrary shift amounts.

Lemma 2.4 (Bigger shifts)**.** *For any positive integer r, an (n, b, d, δ)-LPHS h satisfies*

$$\Pr_{x^{(b)}} \left[h(x^{(b)}) \neq h(x^{(b)} \lll r) + r \right] \leq r \cdot \delta.$$

The same holds for non-cyclic LPHS, where $x^{(b)} \ll r$ is replaced by the string $x^{(b)} \lll r$ obtained from $x^{(b)}$ by chopping the r left-most symbols and adding r uniformly random symbols to the right. In the k-dimensional case, we have

$$\Pr_{x^{(k,b)}}\left[h(x^{(k,b)}) \neq h(x^{(k,b)} \ll (r_1, \ldots, r_k)) + (r_1, \ldots, r_k)\right] \leq \left(\sum_{i=1}^{k} r_i\right) \cdot \delta.$$

Proof. We give the proof for the cyclic 1-dimensional case for simplicity. The proof for the non-cyclic and k-dimensional cases is similar.

$$\Pr_{x^{(b)}}\left[h(x^{(b)}) \neq h(x^{(b)} \ll r) + r\right] \leq$$
$$\Pr_{x^{(b)}}\left[\exists i \in [0, r-1] : h(x^{(b)} \ll i) \neq h(x^{(b)} \ll i+1) + 1\right] \leq$$
$$\sum_{i=0}^{r-1} \Pr_{x^{(b)}}\left[h(x^{(b)}) \neq h(x^{(b)} \ll 1) + 1\right] \leq r \cdot \delta.$$

■

As a simple corollary, we get a lower bound on the error probability of non-cyclic LPHS.

Claim 2.5 (Error bound for non-cyclic LPHS)**.** *For any non-cyclic (n, b, d, δ)-LPHS we have $\delta \geq 1/(2n)$.*

Proof. Suppose towards contradiction that $\delta < 1/(2n)$. Applying Lemma 2.4 with $r = n$ we get that for two random and independent strings $x, x' \in_R \Sigma_b^n$ we have $\Pr_{x,x'}[h(x) = h(x') + n] > 1/2$, where addition is taken over the integers. By symmetry, we also have $\Pr_{x,x'}[h(x') = h(x)+n] > 1/2$. Since the two events cannot co-occur, we get a contradiction. ■

The above lower bound does *not* apply to cyclic LPHS. Indeed, when $d > n^{1/2}$ one can use discrete logarithm techniques to get cyclic LPHS in which δ is negligible in n. The proof of Claim 2.5 extends naturally to the k-dimensional case, where we have $\delta \geq 1/(2kn)$.

The following lemma shows a simple way to trade alphabet size for input length.

Lemma 2.6 (Bigger alphabet)**.** *For any positive integer r such that n/r is an integer, if there exists an (n, b, d, δ)-LPHS, then there exists an $(n/r, b \cdot r, d, r \cdot \delta)$-LPHS.*

Proof. On input $x^{(b \cdot r)}$, the $(n, b \cdot r, d, \delta \cdot k)$-LPHS (denoted by h_1) runs the (n, b, d, δ)-LPHS (denoted by h_2) on $x^{(b)}$, and outputs $\lfloor h_2(x^b)/r \rfloor$. Clearly, each query of h_2 to $x^{(b)}$ can be answered by a single query of h_2 to $x^{(b \cdot r)}$. Based on Lemma 2.4,

$$\Pr_{x^{(b \cdot r)}}\left[h_1(x^{(b \cdot r)}) \neq h_1(x^{(b \cdot r)} \ll 1) + 1\right] \leq \Pr_{x^{(b)}}\left[h_2(x^{(b)}) \neq h_2(x^{(b)} \ll r) + r\right] \leq r \cdot \delta.$$

■

The following lemma can be used to decrease the alphabet size of LPHS. In particular, it is useful for obtaining near-optimal LPHS over a *binary* alphabet from LPHS over a bigger alphabet Σ_b, such as the one derived directly from [23].

Lemma 2.7 (Smaller alphabet)**.** *For any positive integers r, k, if there exists an $(n, b \cdot r, d, \delta)$-LPHS, then there exists an $(n, b, d \cdot k, \delta + 4 \cdot d^2/2^{b \cdot k})$-LPHS.*

Proof. Given a function $f : \{0,1\}^{b\cdot k} \to \{0,1\}^{b\cdot r}$ selected uniformly at random, on input $x^{(b)}$, the $(n, b, d\cdot k, \delta + 4\cdot d^2/2^{b\cdot k})$-LPHS (denoted by h_1) runs the $(n, b\cdot r, d, \delta)$-LPHS (denoted by h_2) and answers query i by querying a k-tuple of elements and applying $f(x^{(b)}[i], \ldots, x^{(b)}[i+k-1])$. Finally, it outputs the same value as h_2.

We define the bad event $\mathcal{E}$, where two k-tuple queries of $h_1(x^{(b)})$ or $h_1(x^{(b)} \lll 1)$ for $i \neq j$ satisfy $(x^{(b)}[i], \ldots, x^{(b)}[i+k-1]) = (x^{(b)}[j], \ldots, x^{(b)}[j+k-1])$. Conditioned on $\neg\mathcal{E}$,

$$\Pr_{x^{(b)}}\left[h_1(x^{(b)}) \neq h_1(x^{(b)} \lll 1) + 1\right] = \Pr_{x^{(b\cdot r)}}\left[h_2(x^{(b\cdot r)}) \neq h_2(x^{(b\cdot r)} \lll 1) + 1\right] \leq \delta.$$

For any $i \neq j$, the probability that $(x^{(b)}[i], \ldots, x^{(b)}[i+k-1]) = (x^{(b)}[j], \ldots, x^{(b)}[j+k-1])$ is $2^{-b\cdot k}$, and taking a union bound on all $\binom{2d}{2}$ query pairs of $h_1(x^{(b)})$ and $h_1(x^{(b)} \lll 1)$ gives $\Pr[\mathcal{E}] < 4{\cdot}d^2/2^{b\cdot k}$.

Note that this requires both executions of $h_1(x^{(b)})$ and $h_1(x^{(b)} \lll 1)$ to have shared randomness. In a non-uniform setting, we can use an averaging argument to fix this randomness and obtain a deterministic h. To make h semi-explicit, pick f from an efficient family of n-wise independent hash functions. ■

Next, we observe that any cyclic LPHS implies a non-cyclic LPHS with a higher error probability δ. Concretely, the error probability grows by d/n. For $d = n^{1/3}$, this can yield a non-cyclic LPHS with (near-optimal) $\delta = \tilde{O}(n^{-2/3})$. But for $n^{1/3} < d < n^{1/2}$ this generic construction is subsumed by the direct construction from Section A.3 that can achieve the near-optimal parameters $d = n^{1/2}$ and $\delta = \tilde{O}(1/n)$.

Lemma 2.8 (From cyclic to non-cyclic)**.** *If there exists a* cyclic (n, b, d, δ)*-LPHS, then there exists a* non-cyclic $(n, b, d, \delta + d/n)$*-LPHS.*

Proof. Let h be a cyclic (n, b, d, δ)-LPHS. We define a randomized non-cyclic LPHS $h'_\rho(x) = h(x \lll \rho)$ where $\rho \in_R \mathbb{Z}_n$ is a uniformly random cyclic shift. Since the difference between $x \lll 1$ and the corresponding string x' for non-cyclic shift is restricted to the $(n-1)$-th position, after cyclically shifting by ρ the difference will be restricted to a single random position $\rho^* = (n - 1 - \rho) \mod n$. Since h (adaptively) queries d symbols of its input, the probability of querying the differing position ρ^* is bounded by d/n, from which the lemma follows. ■

Finally, several of our results will rely on the following "smoothness" feature of LPHS.

Lemma 2.9. *Let $m > 1$ be a natural number. Then any (n, b, d, δ)-LPHS satisfies*

$$\Pr_x[h(x) \bmod m = a] \leq m^{-1} + \delta + O(1/n)$$

for every $a \in \mathbb{Z}_m$.

Proof. We assume that $b = 1$. The proof for $b > 1$ is similar.

Consider the set S of inputs x for which the minimal r such that $x = x \lll r$ is n. The set S contains a fraction of $1 - o(1/n)$ of the inputs $x \in \{0,1\}^n$. Suppose first that m divides n. Partition the set S into sequences of length m of the form $x, x \lll 1, \ldots, x \lll (m-1)$ and consider the executions $h(x) \bmod m, \ldots, h(x \lll (m-1)) \bmod m$. Note that if a appears $t > 1$ times in an execution sequence, then h must err on at least $t - 1$ inputs in this sequence. Consequently,

$$\Pr_x[h(x) \bmod m = a] \leq m^{-1} + \delta + o(1/n).$$

If m does not divide n, then the set S cannot be accurately partitioned as above. Consequently, each set of cyclic shifts of any input x (of size n) in S may contain at most one additional input y such that $h(y) \bmod m = a$, contributing to the additional $O(1/n)$ factor. ■

3 LPHS and Distributed Discrete Log

In this section we introduce the Generic Group Model (GGM) and Distributed Discrete Logarithm (DDL) problem, and present a general two-way relation between (cyclic) LPHS and generic-group algorithms for DDL. We discuss and define the GGM and DDL problem in Section 3.1, and we provide the correspondence with LPHS in Section 3.2.

3.1 Generic Group Model and Distributed Discrete Log

Our notion of LPHS is closely related to variants of the discrete logarithm problem: given a group generator $g \in \mathrm{G}$ and a group element g^v, find v. More concretely, we will be interested in algorithms for problems related to discrete logarithm in the so-called *generic group model* (GGM). The GGM assigns random labels to group elements and treats the group operation as an oracle. We formalize this below.

Let n be a positive integer parameter (corresponding to the group order) and $b \geq 3 \log n$ an integer (representation length of group elements). The GGM setting can be described as a game, where at the beginning, a string $x^{(b)} \in \Sigma_b^n$ is chosen uniformly at random.[4] In the discrete log problem for $\mathbb{Z}_n$, a value $v \in \mathbb{Z}_n$ is chosen uniformly at random. A generic algorithm A for the discrete log problem in $\mathbb{Z}_n$ is a probabilistic algorithm that issues d (adaptive) queries of the form $(i,j) \in \mathbb{Z}_n \times \mathbb{Z}_n$. The answer to query (i,j) is $x^{(b)}[\ell_v(i,j)]$, where $\ell : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ is the affine query evaluation function defined by $\ell_v(i,j) = i \cdot v + j$ (where arithmetic operations are performed modulo n). Using the group notation, the query (i,j) corresponds to group element $(g^v)^i \cdot g^j$.

The algorithm A succeeds to solve the discrete log problem if $A^{\mathrm{G}}(x^{(b)}, v) = v$,[5] and its success probability is taken over the uniform choices of $x^{(b)}$ and v (and possibly additional randomness of its own coin-tosses).

The flavor of GGM we use in this paper is similar to the one of Shoup [41]. Besides differences in notations, there are two additional technical differences which are generally minor. First, in [41], strings are uniformly assigned to elements of $\mathbb{Z}_n$ without replacement, whereas in our model, we assign strings with replacement. However, a collision $x[i] = x[j]$ for some pair (i,j) is possible with probability $\leq 1/n$, which is negligible in our context. Second, in [41] queries of A are limited to linear combinations with coefficients of ± 1 to previously queried elements (where the initial queried elements consist of g and g^v). We note that any query (i,j) can be issued in Shoup's original GGM after at most $O(\log n)$ queries (using the double-and-add algorithm). Therefore, although our model is slightly stronger, any algorithm in our model can be simulated by an algorithm in the model of [41] by increasing the query complexity by a multiplicative factor of $O(\log n)$.

The following success probability upper bound was proved in [41].

Theorem 3.1 ([41], Theorem 1 (adapted))**.** *For a generic discrete log algorithm A with d queries and prime n, we have* $\Pr_{x^{(b)},v}[A^{\mathrm{G}}(x^{(b)}, v) = v] = O(d^2/n)$.

Although our model is slightly different than the one of [41], this result holds in our model as well (by a straightforward adaptation of the proof of [41]). The assumption that n is prime ensures that $\mathbb{Z}_n$ does not contain any non-trivial subgroup. It is necessary in general, since for composite

[4] We require $b \geq 3 \log n$ to ensure that for each $i \neq j$, $x[i] \neq x[j]$ (except with $\leq 1/n$ probability).

[5] We use the notation $A^{\mathrm{G}}(x^{(b)}, v)$ to indicate that A is a generic algorithm with no direct access to the parameters $x^{(b)}, v$.

n, the Pohlig-Hellman algorithm [38] breaks the discrete log problem into smaller problems in subgroups of $\mathbb{Z}_n$, beating the bound of Theorem 3.1.

We now define a restricted class of GGM algorithms that better correspond to LPHS.

Definition 3.2. *A GGM algorithm A is called query-restricted if it only issues queries of the form $(i, j) \in \mathbb{Z}_n \times \mathbb{Z}_n$ with $i = 1$.*

Thus, A is restricted to query group elements with a known shift j from v, analogously to the way an LPHS algorithm queries elements at a known offset. Query-restricted algorithms cannot exploit the subgroup structure of composite groups, and thus Theorem 3.1 holds for them regardless of whether n is prime. For a similar reason, the factorization of n will not play any role in our results on LPHS.

LPHS is closely related to query-restricted GGM algorithms for a variant of discrete log called *distributed discrete log* (DDL) [12, 23] that we describe next. The syntax is identical to that of discrete log. However, the goal here is different: rather than output v when the (implicit) input is v, the goal here is to maintain the *difference* between the outputs on v and $v+1$, except with error probability δ. More formally:

Definition 3.3. *A GGM algorithm A is an (n, b, d, δ)-DDL algorithm if it makes d (potentially adaptive) queries to $x^{(b)}$ and $\Pr_{x^{(b)},v}[A^{\mathrm{G}}(x^{(b)}, v) - A^{\mathrm{G}}(x^{(b)}, v+1) \neq 1] \leq \delta$.*

Remark. The original definition of DDL in [12] involves two parties A and B that may potentially run two different algorithms. The parties are placed within an unknown distance $r \in \{-1, 0, 1\}$ from each other and their goal is to minimize the error probability defined as $\Pr_{x^{(b)},v}[A^{\mathrm{G}}(x^{(b)}, v) - B^{\mathrm{G}}(x^{(b)}, v+r) \neq r]$. However, it was shown in [23, Lemma 9], that if both parties use A's algorithm, then the multiplicative loss in error probability is bounded by a constant. Hence, the above restricted definition of DDL is essentially equivalent to the original one of [12].

While LPHS is only directly related to query-restricted GGM algorithms for DDL, Lemma 3.7 asserts that when n is prime and d is sufficiently small compared to n, any unrestricted GGM algorithm for DDL can be converted to a query-restricted one at a negligible cost in error probability. This gives rise to Corollary 3.8 that establishes a reduction which converts any unrestricted GGM algorithm for DDL to an LPHS with a negligible cost in error probability.

3.2 Reductions Between LPHS Variants and DDL

In this section we show that a query-restricted DDL algorithm in the GGM is equivalent to our basic notion of (cyclic) LPHS, and then describe consequences of this equivalence. We then show that for most parameters, any DDL algorithm can be converted into a query-restricted one at a negligible cost.

We use this correspondence to derive asymptotically tight bounds on the parameters of LPHS, using the Iterative Random Walk algorithm from [23].

Equivalence of LPHS and query-restricted DDL. The equivalence in the query-restricted model is formally captured by the following two-way relation.

Lemma 3.4. *There exist reductions that convert an (n, b, d, δ)-LPHS to a query-restricted (n, b, d, δ)-DDLA and vice versa.*

Proof. Given access to an (n, b, d, δ)-LPHS denoted by h, we construct a query-restricted (n, b, d, δ)-DDLA, denoted by A as follows. We run h and translate query j into query $(1, j)$ for $A^{\mathrm{G}}(x^{(b)}, v)$. We then feed the answer $x^{(b)}[v+j]$ to h. Finally, we output the same value as h. Since $x^{(b)}[v+j] = (x^{(b)} \lll v)[j]$, we have $A^{\mathrm{G}}(x^{(b)}, v) = h(x^{(b)} \lll v)$, where $x^{(b)} \lll v$ is a uniform string. Therefore,

$$\Pr_{x^{(b)},v}[A^{\mathrm{G}}(x^{(b)}, v) - A^{\mathrm{G}}(x^{(b)}, v+1) \neq 1] =$$

$$\Pr_{x^{(b)},v}[h(x^{(b)} \lll v) - h(x^{(b)} \lll v+1) \neq 1] =$$

$$\Pr_{x^{(b)}}[h(x^{(b)}) - h(x^{(b)} \lll 1) \neq 1] = \delta.$$

In a similar way, a query-restricted (n, b, d, δ)-DDLA can be used to construct a (n, b, d, δ)-LPHS. ■

The DDL algorithm based on the Iterated Random Walk (IRW) from [23] is query-restricted. Therefore, combining Lemma 3.4 with the parameters of IRW, we get the positive result below for cyclic LPHS. The result for non-cyclic LPHS follows from the fact that the random walk makes queries within an interval of size bounded by $O(d^2)$, hence if $n = \Omega(d^2)$ is large enough, the LPHS gives both cyclic and non-cyclic LPHS with the same parameters.

Theorem 3.5 (LPHS Upper Bounds)**.** *For $n = \Omega(d^2)$ and $b \geq 3 \log n$ there is an (n, b, d, δ)-LPHS such that $\delta = O(1/d^2)$. Moreover, for $n = \Omega(d^2)$ and any $b \geq 1$ there is an (n, b, d, δ)-LPHS with $\delta = \tilde{O}(1/d^2)$. There are also non-cyclic LPHS with the same parameters.*

We can similarly convert the main negative result for DDLA from [23, Theorem 5] to a nearly tight lower bound on the error probability of LPHS.

Theorem 3.6 (LPHS Lower Bound)**.** *For $n = \Omega(d^2)$, any (cyclic or non-cyclic) $(n, 1, d, \delta)$-LPHS satisfies $\delta \geq \Omega(1/d^2)$.*

From GGM to query-restricted GGM. In this section we show that, when n is prime and d is sufficiently small compared to n, any DDL algorithm can be converted into a query-restricted one with similar parameters.

Lemma 3.7. *For $b \geq 3 \log b$, there exists a reduction that converts any (n, b, d, δ)-DDLA, for prime n, to a query-restricted $(n, b, d, \delta + O(d^2/n))$-DDLA.*

If $d = O(n^{1/4})$, then since $\delta = \Omega(d^{-2})$ by Theorem 3.6, we have $\delta + O(d^2/n) = \delta + O(d^{-2}) = O(\delta)$. Hence the reduction is almost without loss.

Proof. (sketch) Given black-box access to an (n, b, d, δ)-DDLA denoted by A, we construct a query-restricted $(n, b, d, \delta + O(d^2/n))$-DDLA, denoted by B. We will define a query mapping $\mathit{map} : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ below and run A, while translating query (i, j) into query $(1, \mathit{map}(i, j))$ for $B^{\mathrm{G}}(x^{(b)}, v)$ for which the answer $x^{(b)}[v + \mathit{map}(i, j)]$ is fed back into A. Finally, we output the same value as A.

Our goal in defining map is to simulate the joint distribution of query answers for $A^{\mathrm{G}}(y^{(b)}, v)$ and $A^{\mathrm{G}}(y^{(b)}, v+1)$ (for uniform $y^{(b)}, v$), while making only restricted queries. The simulation will be perfect unless some bad event (which happens with probability $O(d^2/n)$) occurs. This will assure that the error probability of h is bounded by $\delta + O(d^2/n)$.

The query answers of $A^{\mathrm{G}}(y^{(b)}, v)$ and $A^{\mathrm{G}}(y^{(b)}, v+1)$ are uniform in Σ_b, unless they query the same element, namely, $\ell_{v+a_0}(i,j) = \ell_{v+a_1}(i',j')$ for $i,j,i',j' \in \mathbb{Z}_n$ and $a_0, a_1 \in \{0,1\}$. We thus require that the mapping preserves equality, namely,

$$\ell_{v+a_0}(i,j) = \ell_{v+a_1}(i',j') \Leftrightarrow \ell_{v+a_0}(1, \mathit{map}(i,j)) = \ell_{v+a_1}(1, \mathit{map}(i',j')). \tag{1}$$

Considering the left-hand side, if $i \neq i'$, then the discrete log v can be computed (e.g., for $a_0 = a_1 = 0$, $v = (j'-j)\cdot(i-i')^{-1}$). We refer to this first bad event as a collision, and can bound its probability by $O(d^2/n)$ using Theorem 3.1. Hence, the analysis will assume that if $\ell_{v+a_0}(i,j) = \ell_{v+a_1}(i',j')$, then $i = i'$.

The second bad event is that $\ell_{v+a_0}(1, \mathit{map}(i,j)) = \ell_{v+a_1}(1, \mathit{map}(i',j'))$, but $\ell_{v+a_0}(i,j) \neq \ell_{v+a_1}(i',j')$. The probability of this event will be bounded by $O(d^2/n)$ below.

In order to define *map*,[6] assume that we fix elements $D_0, D_1, \ldots, D_{n-1}$, where $D_i \in \mathbb{Z}_n$. We set

$$\mathit{map}(i,j) = \begin{cases} D_i + j \cdot i^{-1}, & \text{if } i \neq 0 \\ D_i + j, & \text{otherwise,} \end{cases}$$

where i^{-1} is the multiplicative inverse of i modulo n.

Observe that for $i \neq 0$, if $\ell_v(i,j) = \ell_{v+1}(i,j')$, then $j = j' + i$ implying that

$$\ell_v(1, \mathit{map}(i,j)) = v + D_i + j \cdot i^{-1} = v + D_i + (j'+i)\cdot i^{-1} = v + 1 + D_i + j' \cdot i^{-1} = \ell_{v+1}(1, \mathit{map}(i,j')),$$

and the same equality holds for $i = 0$. Hence, assuming no collision, the first (right) part of Equation (1) holds for $a_0 \neq a_1$ (it also holds trivially for $a_0 = a_1$).

For the second part, observe that if $\ell_{v+a_0}(1, \mathit{map}(i,j)) = \ell_{v+a_1}(1, \mathit{map}(i',j')$ for $i, i' \neq 0$, then $v + a_0 + D_i + j \cdot i^{-1} = v + a_1 + D_{i'} + j' \cdot (i')^{-1}$. If $i = i'$, we get $i \cdot a_0 + j = i \cdot a_1 + j'$. Hence,

$$\ell_{v+a_0}(i,j) = i \cdot (v + a_0) + j = i \cdot (v + a_1) + j' = \ell_{v+a_1}(i,j'),$$

as required (a similar equality holds when $i = i' = 0$). When $i \neq i'$, we get a second bad event and we choose each $D_i \in \mathbb{Z}_n$ uniformly at random to bound its probability (for any specific a_0, i, j, a_1, i', j') by $1/n$. Summing over all $\binom{2d}{2}$ query pairs of $B^{\mathrm{G}}(x^{(b)}, v)$ and $B^{\mathrm{G}}(x^{(b)}, v+1)$, we bound the probability of the second bad event by $O(d^2/n)$.

Note that this requires both executions of $B^{\mathrm{G}}(x^{(b)}, v)$ and $B^{\mathrm{G}}(x^{(b)}, v+1)$ to have shared randomness. However, this requirement can be removed since our model is non-uniform, and $D_i \in \mathbb{Z}_n$ can be fixed by a standard averaging argument. ■

Combined with Lemma 3.4, we obtain the following corollary.

Corollary 3.8. *For $b \geq 3 \log b$, there exists a reduction that converts any (n, b, d, δ)-DDLA, for prime n, to an $(n, b, d, \delta + O(d^2/n))$-LPHS.*

4 Multidimensional LPHS

In this section we study the k–dimensional generalization of LPHS, focusing mainly on the case $k = 2$ (2D-LPHS). First, in Section 4.1, we consider the upper bound side. We start with simple constructions achieving error $\delta = \tilde{O}(d^{-1/2})$ (Section 4.1.1) and $\delta = \tilde{O}(d^{-4/5})$ (Section 4.1.2). The

[6] This mapping was defined in a slightly different context in [23].

latter makes a black-box use of the 1-dimensional IRW algorithm of [23]. More concretely, for $n = \Omega(d^{6/5})$, any $b \geq 3\log n$ and any $(r_1, r_2) \in \{0,1\} \times \{0,1\}$, we have

$$\Pr_{x \in_R \Sigma_b^{\mathbb{Z}_n \times \mathbb{Z}_n}} [h(x) - h(x \ll (r_1, r_2)) \neq (r_1, r_2)] = O(d^{-4/5}). \tag{2}$$

Since our construction only makes queries in a limited box of dimensions $O(d^{6/5}) \times O(d^{4/5})$ while $n = \Omega(d^{6/5})$, it gives both a cyclic and a non-cyclic LPHS with the same parameters.

This proves a weak version of Theorem 1.3. In Section 4.1.3 we obtain the improved upper bound of Theorem 1.3 by presenting a more intricate algorithm that achieves error rate of $\delta = \tilde{O}(d^{-7/8})$. Finally, in Section 4.1.4 we present a heuristic algorithm that we *conjecture* to achieve the near-optimal error probability of $\delta = \tilde{O}(d^{-1})$. This conjecture is supported by experimental evidence.

In Section 4.2 we study limitations of k-dimensional LPHS. We prove Theorem 1.4, which for $k = 2$ implies that the error probability of a 2D-LPHS must satisfy $\delta = \tilde{\Omega}(d^{-1})$.

4.1 Upper bounds on 2D-LPHS algorithms

We present and analyze 2D-LPHS algorithms achieving error $\delta = \tilde{O}(d^{-1/2}), \tilde{O}(d^{-4/5}), \tilde{O}(d^{-7/8})$, respectively, as well as a heuristic algorithm conjectured to achieve optimal error $\delta = \tilde{O}(d^{-1})$.

4.1.1 A simple 2D-LPHS with error rate $\delta = O(d^{-1/2})$

We begin by describing a very simple 2D-LPHS algorithm called Min-Hash.

Notation All integer operations in algorithms within this section are assumed to be floored. For example, we write x/y for $\lfloor x/y \rfloor$ and $\sqrt{n}$ for $\lfloor \sqrt{n} \rfloor$.

Algorithm 1: Min-Hash($x \in \Sigma_b^{\mathbb{Z}_n^2}, d \in \mathbb{N}$)

```
begin
    return $\arg\min_{i,j \in [0,\sqrt{d}]} \{x[i,j]\}$
end
```

The following lemma captures the performance of Min-Hash.

Lemma 4.1. *For $n = \Omega(d^{1/2})$ and any $b \geq 3\log n$ and $(r_1, r_2) \in \{0,1\} \times \{0,1\}$,*

$$\Pr[\textsc{Min-Hash}(x,d) - \textsc{Min-Hash}(x \ll (r_1, r_2), d) \neq (r_1, r_2)] = O(d^{-1/2}). \tag{3}$$

Proof. Notice that no matter what the values of $r_1, r_2 \in \{0,1\}$ are, both applications of Min-Hash on x and y query the values $G = \{x[i,j]\}_{i,j \in [1,\sqrt{d}]}$, together with some other $2\sqrt{d} - 1$ values. Hence Min-Hash(x,d) and Min-Hash(y,d) together read at most $4\sqrt{d}$ values outside of G. Under the uniformity assumption of x, the probability the minimum of all the symbols read by the two applications of Min-Hash is not in G, is bounded by $4\sqrt{d}/d = O(1/\sqrt{d})$. Hence, assuming that the minimum $x[i_0, j_0]$ is in G, and that this minimum is unique, we have

$$\textsc{Min-Hash}(x,d) = (i_0, j_0).$$
$$\textsc{Min-Hash}(y,d) = \arg\min_{i,j \in [0,\sqrt{d}]} \{x[i + r_1, j + r_2]\} = (i_0 - r_1, j_0 - r_2),$$

in which case, $\textsc{Min-Hash}(x, d) - \textsc{Min-Hash}(y, d) = (r_1, r_2)$.

The only thing left for the proof is showing that with a very high probability, the minimum of Algorithm 1 is uniquely attained. It can be easily verified (e.g., by induction on d) that the probability that the minimum is not unique, is upper bounded by $d/2^b$. Under the assumption $b \geq 3/2 \lg(d)$, this probability is dominated by the $O(1/\sqrt{d})$ error in (3). ■

4.1.2 An IRW-based 2D-LPHS with error rate $\delta = O(d^{-4/5})$

In this subsection we demonstrate how an 1D-LPHS may be used in order to construct a 2D-LPHS with $\delta = O(d^{-4/5})$. Let us recall the functionality of an optimal 1D-LPHS (see Theorem 3.5).

Optimal 1D-LPHS, rephrased. For $b \geq 3 \log n$, there exists an algorithm $\textsc{optimal1D}\colon \Sigma_b^{\mathbb{Z}_n} \times \mathbb{Z} \to \mathbb{Z}_n$ with the following properties. If $n = \Omega(d^2)$, then

$$\Pr_{x \sim \Sigma_b^{\mathbb{Z}_n}} [\textsc{optimal1D}(x, d) - \textsc{optimal1D}(x \ll 1, d) \neq 1] < O(1/d^2).$$

The 2D-LPHS is described in the Recursive-Hash algorithm (Algorithm 3). The algorithm works in two stages, first returning a column i_1 and then a row j_1. Namely, the parties first try to synchronize on a column via a random walk along their input rows, and then try to synchronize on a row via a random walk along this common column.

Specifically, the column i_1 is located by a walk along the fixed input row. The walk uses the optimal1D algorithm with $d/d' - 1$ queries (for a parameter d'). A query in this algorithm on any column i_0 is answered by the rec1D algorithm, which executes optimal1D with d' queries on column i_0.

After the column i_1 is returned, the Recursive-Hash algorithm runs the optimal1D algorithm along this column, starting from the input row to return the output row j_1.

Algorithm 2: $\textsc{rec1D}(z \in \Sigma_b^{\mathbb{Z}_n^2}, d' \in \mathbb{N}, i_0 \in \mathbb{Z}_n)$

```
begin
    Define $u \in \Sigma_b^{\mathbb{Z}_n}$ by $u[j] \leftarrow z[i_0, j]$
    $j_0 \leftarrow \textsc{optimal1D}(u, d')$
    return $z[i_0, j_0 + 10d'^2]$
end
```

Algorithm 3: $\textsc{Recursive-Hash}(z \in \Sigma_b^{\mathbb{Z}_n^2}, d \in \mathbb{N})$

```
begin
    $i_1 \leftarrow \textsc{optimal1D}(i \mapsto \textsc{rec1D}(z, d^{3/5} - 1, i), d^{2/5})$
    $j_1 \leftarrow \textsc{optimal1D}(j \mapsto z[i_1, j], d^{2/5})$
    return $(i_1, j_1)$
end
```

Lemma 4.2. *For $n = \Omega(d^{6/5})$ and any $b \geq 3\log n$ and $(r_1, r_2) \in \{0,1\} \times \{0,1\}$,*

$$\Pr\left[\textsc{Rec2D}(x,d) - \textsc{Rec2D}(x \lll (r_1,r_2), d) \neq (r_1,r_2)\right] = O(d^{-4/5}).$$

Proof. First, notice Recursive-Hash(x,d) makes at most d queries to x. This is because all but $d^{2/5}$ queries are made inside rec1D. This procedure is called at most $d^{2/5}$ times, and queries u at most $d^{3/5} - 1$ times, which in turn makes exactly one query to x.

Second, denoting $y = x \lll (r_1, r_2)$, notice that since $r_1 \in \{0,1\}$, we have from Theorem 3.5,

$$\Pr\left[\textsc{rec1D}(x, d^{3/5} - 1, i + r_1) \neq \textsc{rec1D}(y, d^{3/5} - 1, i)\right] \leq O(d^{-6/5}).$$

Since Recursive-Hash makes at most $d^{2/5}$ calls to rec1D, then except for probability $O(d^{-6/5} \cdot d^{2/5}) = O(d^{-4/5})$, we have $\textsc{rec1D}(x, d^{3/5}, i + r_1) = \textsc{rec1D}(y, d^{3/5}, i)$ for all i's such that both sides were computed in Recursive-Hash(x,d) and Recursive-Hash(y,d), respectively. If the output of $\textsc{rec1D}(x, d^{3/5}, i)$ would be uniformly distributed in Σ_b (and independent of all other queries), Theorem 3.5 would imply $i_1^x + r_1 = i_1^y$ except for probability $O(d^{-4/5})$, where i_1^z is the value of i_1 computed during Recursive-Hash(z,d), and $z \in \{x, y\}$. Similarly, j_1 is synchronized except for probability $O(d^{-4/5})$. In conclusion, the total error probability of the algorithm is $O(d^{-4/5})$. Finally, $x[i_0, j_0 + 10d'^2]$ is indeed uniformly distributed in Σ_b independently of all other queries, as optimal1D does not query that value. ■

4.1.3 3-Stage-Hash: a 2D-LPHS algorithm with $\delta = \tilde{O}(d^{-7/8})$

In this subsection we prove Theorem 1.3, by presenting the algorithm 3-Stage-Hash which achieves error rate of $\delta = \tilde{O}(d^{-7/8})$. 3-Stage-Hash is composed of 3 stages. The first is Min-Hash and we refer to the other two as stage2 and stage3.

Algorithm 4: 3-Stage-Hash$(z \in \Sigma_b^{\mathbb{Z}_n^2}, d \in \mathbb{N})$

```
begin
    $(i_0, j_0) \leftarrow$ Min-Hash$(z,\ d/3)$
    $(i_1, j_1) \leftarrow$ stage2$(z,\ d/3,\ i_0 + 2\sqrt{d},\ j_0 + 2\sqrt{d})$
    $(i_2, j_2) \leftarrow$ stage3$(z,\ d/3,\ i_1 + 2d^{3/4},\ j_1 + 2d^{3/4})$
    return $(i_2, j_2)$
end
```

Lemma 4.3. *Let $2^b \geq d^4$, $n \geq \Omega(d)$, and $r_1, r_2 \in \{0,1\}$. Then,*

$$\Pr\left[\text{3-}\textsc{Stage-Hash}(x,d) - \text{3-}\textsc{Stage-Hash}(y,d) \neq (r_1,r_2)\right] < O(d^{-7/8}), \tag{4}$$

In order to prove Lemma 4.3, we need the following facts, which we prove later.

Lemma 4.4. *Let $S_1, S_2, \ldots$ be a sequence of i.i.d. geometric random variables: $S_i \sim \mathrm{Geom}(p)$. If K is the minimal integer with $\sum_{k=1}^{K} S_k \geq r$, then $\mathbb{E}[K] \leq rp + 1$.*

Lemma 4.5. *Let $I_0, I_1, I_2, \ldots$ be a random walk with $I_{k+1} - I_k$ being i.i.d. variables distributed as the difference of two independent random variables uniformly distributed in $\{0, 1, \ldots, m\}$, with $m \in \mathbb{N}$. If T is the minimal time with $I_T = 0$, then $\mathbb{E}[\min(T, r)] \leq O\left((m + |I_0|/m)\sqrt{r}\right)$.*

Algorithm 5: stage2($z \in \Sigma_b^{\mathbb{Z}_n^2}, d' \in \mathbb{N}, i_0 \in \mathbb{Z}_n, j_0 \in \mathbb{Z}_n$)

```
begin
    $L \leftarrow \sqrt[4]{d'}$
    Let $\psi_1, \psi_2 \colon \Sigma_b \to \mathbb{Z}_L$ be $\psi_1(t) = t \bmod L$ and $\psi_2(t) = \lfloor t/L \rfloor \bmod L$
    $P, P' \leftarrow \emptyset, \emptyset$
    $i \leftarrow i_0$
    for $s = 1$ to $\sqrt{d'}$ do
        $j \leftarrow j_0$
        for $t = 1$ to $\sqrt{d'}$ do
            $P' \leftarrow P' \cup \{(i, j)\}$
            $j \leftarrow j + 1 + \psi_1(z[i, j])$
        end
        $i \leftarrow i + 1 + \psi_2(\min_{p' \in P'}\{z[p']\})$
        $P, P' \leftarrow P \cup P', \emptyset$
    end
    return $\arg\min_{p \in P}\{z[p]\}$
end
```

Given these facts, we turn to Lemma 4.3.

Proof (of Lemma 4.3, sketch). Let (i_k^z, j_k^z) be the values computed as (i_k, j_k) at 3-Stage-Hash(z, d), for $k \in \{0, 1, 2\}$ and $z \in \{x, y\}$. We say that (i_k^x, j_k^x) and (i_k^y, j_k^y) are synchronized if $(i_k^x, j_k^x) - (i_k^y, j_k^y) = (r_1, r_2)$. Observe that if (i_k^x, j_k^x) and (i_k^y, j_k^y) are synchronized, then so are (i_{k+1}^x, j_{k+1}^x) and (i_{k+1}^y, j_{k+1}^y). This is because each stage $k + 1$ of 3-Stage-Hash(z, d) deterministically depends on

Algorithm 6: stage3($z \in \Sigma_b^{\mathbb{Z}_n^2}, d' \in \mathbb{N}, i_0 \in \mathbb{Z}_n, j_0 \in \mathbb{Z}_n$)

```
begin
    Let $\psi \colon \Sigma_b \to \{-d'^{3/8}, \ldots, d'^{3/8}\}$ be $\psi(t) = (t \bmod (2d'^{3/8} + 1)) - d'^{3/8}$
    $P \leftarrow \emptyset$
    $(i, j) \leftarrow (i_0, j_0)$
    for $s = 1$ to $d'$ do
        $P \leftarrow P \cup \{(i, j)\}$
        $(i, j) \leftarrow (i + 1, j + \psi(z[i, j]))$
    end
    return $\arg\min_{p \in P}\{z[p]\}$
end
```

values queried from z with offset (i^z_k, j^z_k), so that evaluations keep being aligned. Overall,

$$\begin{aligned} \delta \doteqdot & \Pr_{x,y,r_1,r_2}[\text{3-}\textsc{Stage-Hash}(x,d) - \text{3-}\textsc{Stage-Hash}(y,d) \neq (r_1,r_2)] \\ & \leq \Pr[(i^x_0, j^x_0) - (i^y_0, j^y_0) \neq (r_1,r_2)] \cdot \\ & \cdot \prod_{k=1}^{2} \Pr\left[(i^x_k, j^x_k) - (i^y_k, j^y_k) \neq (r_1,r_2) \,\middle|\, (i^x_{k-1}, j^x_{k-1}) - (i^y_{k-1}, j^y_{k-1}) \neq (r_1,r_2)\right] . \end{aligned} \tag{5}$$

Hence, to bound δ it is sufficient to verify the following three claims:

1. $\Pr[(i^x_0, j^x_0) - (i^y_0, j^y_0) \neq (r_1,r_2)] \leq O(1/\sqrt{d})$.
2. $\Pr\left[(i^x_1, j^x_1) - (i^y_1, j^y_1) \neq (r_1,r_2) \,\middle|\, (i^x_0, j^x_0) - (i^y_0, j^y_0) \neq (r_1,r_2)\right] \leq O(1/\sqrt[4]{d})$.
3. $\Pr\left[(i^x_2, j^x_2) - (i^y_2, j^y_2) \neq (r_1,r_2) \,\middle|\, (i^x_1, j^x_1) - (i^y_1, j^y_1) \neq (r_1,r_2)\right] \leq O(1/\sqrt[8]{d})$.

Claim 1) Follows from Lemma 4.1.

Claim 2) Since Min-Hash(z,d) scans a $\sqrt{d} \times \sqrt{d}$ area and outputs a point inside it, we are guaranteed that $|i^x_0 - i^y_0| \leq \sqrt{d}+1$ and $|j^x_0 - j^y_0| \leq \sqrt{d}+1$. Because stage2 is fed with the output point of Min-Hash shifted by $2\sqrt{d}$ in each axis, its queries do not overlap these of Min-Hash, and its performance is independent of the Min-Hash phase. Moreover, stage2 can be modeled as a random walk on the i axis, whose steps are integers uniformly distributed in $[1, \sqrt{d'}]$ (with $d' = d/3$, as in stage2), which are determined by some random walk on the j axis. Denote by $I^x_1, \dots, I^x_{\sqrt{d'}}$ and $I^y_1, \dots, I^y_{\sqrt{d'}}$ the sequences of i's observed by stage2 applied on x and on y.

In order to compute the probability that the outputs of stage2(x,d') and stage2(y,d') are not synchronized, it is sufficient (following the proof of Lemma 4.1) to count the number of queries that the two processes make, which are not shared. These queries can be classified into two categories: queries with non-shared i, and queries with shared i and non-shared j. Our goal is to show each class contains on average $O(d'^{3/4})$ such queries, implying that the probability of the outputs not being synchronized is $O(d'^{3/4}/d')$ (similarly to Lemma 4.1). We start by reasoning about the first class of queries, and then proceed to the second.

Let U_1 denote the total number of i-steps until i^x and i^y are synchronized (i.e. $U_1 = k + k'$ when k, k' are minimal with $I^x_k - I^y_{k'} = r_1$). Up to this point, the two stage2 applications act independently, as their queries do not overlap. Using [23, Lemmas 3,5] with $b \leq \sqrt{d}+1$ and $L = \sqrt[4]{d'}$ we see $\mathbb{E}[U_1] \leq O(d^{1/4})$. Next, we note that once $I^x_k - I^y_{k'} = r_1$, it is likely that $I^x_{k+1} - I^y_{k'+1} = r_1$. Specifically, we will show

$$\Pr\left[I^x_{k+1} - I^y_{k'+1} \neq r_1 \,\middle|\, I^x_k - I^y_{k'} = r_1\right] \leq O(1/\sqrt{d}).$$

Assuming this, the two walks make U_1 unsynchronized steps, then S_1 synchronized steps with $S_1 \sim \text{Geom}(O(1/\sqrt{d}))$ distributed geometrically. The walks then make another U_2 unsynchronized steps, with [23, Lemma 5] yielding $\mathbb{E}[U_2] \leq O(d^{1/4})$, followed by S_2 synchronized steps with $S_2 \sim \text{Geom}(O(1/\sqrt{d}))$. The process continues this way until one of the walks has completed its $\sqrt{d'}$ steps. Using Lemma 4.4, the expected number of such phases of synchronization-unsychronization is $\leq O(\sqrt{d'} \cdot \sqrt{1/d} + 1) = O(1)$. Combining this with the fact that $\mathbb{E}[U_k] = O(d^{1/4})$, we deduce that the expected number of unsynchronized i-steps is $O(d^{1/4})$. Each such step involves $\sqrt{d'}$ j-steps, so the total number of queries with non-shared i is $O(d^{3/4})$.

Regarding the steps with shared i and non-shared j, random-walk arguments similar to the above argument imply that since on each shared i, the two j-walks start with distance $O(\sqrt{d})$, and have steps of size $\Theta(d^{1/4})$, they are expected to meet after $O(d^{1/4})$ queries (follows from [23, Lemmas 3,5]). Since there are $O(\sqrt{d})$ i-steps, the total number of non-shared such queries is $O(d^{3/4})$ as well.

Claim 3) Similarly to the previous claim, the queries made by stages before STAGE3 are confined to a square area of size $(d^{3/4} + 3\sqrt{d}) \times (d^{3/4} + 3\sqrt{d})$, and since the input is shifted by $2d^{3/4}$, the queries of STAGE3 do not overlap previous stages. Note that the queries made by STAGE3 are confined to a $2d \times 2d$ square except for a negligibly small error prbobability ($\exp(-\Omega(d^{1/4}))$) due to Hoeffding's inequality, and since $n \geq \Omega(d)$ (recall $x, y \in \Sigma_b^{\mathbb{Z}_n^2}$), the queries of the different stages do not overlap (with high probability) even though the index space of x, y is cyclic.

It is clear that once the two walks of STAGE3 on x and on y are synchronized, they remain synchronized. Let T be the total number of steps until the two walks share a point. There are at most $\min(2T, d')$ steps which are not shared, and the failure probability is $\leq \min(2T, d')/d'$, similarly to the proof of Lemma 4.1. Thus, it is sufficient to verify $\mathbb{E}[\min(T, d)] \leq d^{7/8}$. Clearly, after $|i_1^x - i_1^y|$ steps, the walks are being synchronized with respect to the i-axis. Let J denote the random variable measuring their distance on the j-axis, once the walks first share this same i. Since each of the advances of j are independent of the other steps,

$$\mathbb{E}[J^2] = |j_1^x - j_1^y|^2 + \sum_{t=0}^{|i_1^x - i_1^y|} \mathbb{E}[S_t^2],$$

where S_t is the jump on the j-axis on the t-step of the runner-up walk. In particular $\mathbb{E}[S_t^2] \leq ((d/3)^{3/8})^2 \leq d^{3/4}$. Since $|i_1^x - i_1^y| \leq 2d^{3/4}$, we overall deduce $\mathbb{E}[J^2] \leq O(d^{3/2})$.

From this point on, the walks of STAGE3(x, d') and STAGE3(y, d') keep being aligned with respect to the i-axis. Once they meet on the j-axis, they will remain synchronized. The distance on the j-axis between the walks can be modeled as a one dimensional random walk, starting at J, and having independent steps whose lengths are a difference of two independent variables uniformly distributed in $\{0, 1, \ldots, 2d'^{3/8}\}$. Once this difference walk hits 0, the walks keep being synchronized. Lemma 4.5 then immediately yields

$$\mathbb{E}[\min(T, d')] \leq |i_1^x - i_1^y| + O\left((d'^{3/8} + \mathbb{E}\,|J|/d'^{3/8})\sqrt{d'}\right).$$

Substituting $\mathbb{E}[J]^2 \leq \mathbb{E}[J^2] = O(d^{3/2})$, we obtain $\mathbb{E}[\min(T, d')] \leq d^{7/8}$, as required. ■

We now fill in the proofs of the above-stated facts.

Proof (of Lemma 4.4). Since each S_i counts the number of Bernoulli-p variables until success, K distributes as 1+ the number of successful Bernoulli-p variables, out of r. This interpretation immediately gives $\mathbb{E}[K] = p(r-1) + 1$. ■

Proof (of Lemma 4.5). Let $T_0, T_1, T_2, \ldots$ be the sequence of times $t \geq 0$ with $|I_t| \leq \frac{m+1}{2}$ (in increasing order). Observe that for all i, the event $I_{T_i+1} = 0$ happens with probability $\geq \frac{1}{2(m+1)}$, even when conditioning on the trajectory $\{I_t\}_{t \leq T_i}$ up to time T_i. Let K be minimal with $I_{T_K+1} = 0$. Using the (probabilistic) chain rule this observation means that

$$\Pr[K \geq k] \leq (1 - 1/(2m+2))^k.$$

When combined with

$$\mathbb{E}[\min(T_K, r)] \leq \mathbb{E}[\min(T_0, r)] + \sum_{k=1}^{\infty} \mathbb{E}\left[\mathbb{1}_{\{K \geq k-1\}} \cdot \min(T_k - T_{k-1}, r)\right],$$

we deduce

$$\mathbb{E}[\min(T_K, r)] \leq \mathbb{E}[\min(T_0, r)] + \sum_{k=1}^{\infty} \left(\frac{2m+1}{2m+2}\right)^{k-1} \mathbb{E}\left[\min(T_k - T_{k-1}, r) \,|\, K \geq k-1\right]. \tag{6}$$

Clearly, upper bounding $\mathbb{E}[\min(T_K, r)]$ is relevant, as if T is the minimal time with $I_T = 0$, then $T \leq T_K + 1$, and in particular, $\min(T, r) \leq \min(T_K, r) + 1$. We claim the following:

1. $\mathbb{E}[\min(T_0, r)] \leq O(1 + |I_0|\sqrt{r}/m)$.
2. For all i, and all values of $\{I_t\}_{t \leq T_i}$, $\mathbb{E}\left[\min(T_{i+1} - T_i, r) \,|\, I_0, I_1, \ldots, I_{T_i}\right] \leq O(\sqrt{r})$.

These claims together with (6) and $\min(T, r) \leq \min(T_K, r) + 1$ give

$$\mathbb{E}[\min(T, r)] \leq O(1 + |I_0|\sqrt{r}/m) + \sum_{k=1}^{\infty} \left(\frac{2m+1}{2m+2}\right)^{k-1} O(\sqrt{r}) \leq O(|I_0|\sqrt{r}/m + m\sqrt{r}),$$

as required. Let us verify the above claims.

Claim 2) This is a specialization of Claim 1 to the walk $I_{T_i+1}, I_{T_i+2}, \ldots$, satisfying $|I_{T_i+1}| \leq \frac{3m+1}{2}$.

Claim 1) Without loss of generality assume $I_0 \geq 0$ (due to symmetry). Write $L = m\sqrt{r}$. Instead of the stopping time $\min(T_0, r)$, consider the stopping time T' which is the minimal (time) $t \geq 0$ with $I_t \leq (m+1)/2$ or $I_t > L$. It is standard to show that $\Pr[T' > k]$ decreases exponentially fast with k (albeit with deficient constants), and so all quantities presented in the proof will turn out to be finite (in particular, $\mathbb{E}[T']$).

Since the definition of T_0 is similar to that of T', except that the latter allows to stop also when $I_t > L$, we may upper bound $\mathbb{E}[\min(T_0, r)]$ by $\mathbb{E}[T'] + r\Pr[I_{T'} > L]$, i.e., we compensate by r in all cases when T' is not identical to T_0.

Since $I_{k+1} - I_k$ is a symmetric random variable, and is independent of $I_0, \ldots, I_k$, the sequence $I_0, I_1, \ldots$ constitutes a martingale. In particular, by the *Optional stopping theorem*, $\mathbb{E}[I_{T'}] = I_0$. This implies, together with that $|I_{k+1} - I_k| \leq m$

$$I_0 = \mathbb{E}[I_{T'}] \geq -(m+1)/2 \cdot \Pr[I_{T'} \leq (m+1)/2] + L\Pr[I_{T'} > L] \geq L\Pr[I_{T'} > L] - m,$$

and in particular, $\Pr[I_{T'} > L] \leq (m + I_0)/L$. Recall that our goal of proving *claim 1* is non-trivial only when $I_0 > (m+1)/2$, and so we may assume $\Pr[I_{T'} > L] \leq O(I_0/L)$. In particular,

$$\mathbb{E}[\min(T_0, r)] \leq \mathbb{E}[T'] + r \cdot O(I_0/L) \leq \mathbb{E}[T'] + O(I_0\sqrt{r}/m).$$

Thus the claim is implied from $\mathbb{E}[T'] \leq O(I_0\sqrt{r}/m)$ which we now prove.

Write $s = \mathbb{E}[(I_{k+1} - I_k)^2] = (m^2 + 2m)/6$, and consider the sequence of random variables

$$\left(I_k^2 - sk\right)_{k=0}^{\infty}.$$

We claim it is a martingale. Indeed:

$$\begin{aligned}\mathbb{E}\left[I_{k+1}^2 - s(k+1) \,\middle|\, I_0, \dots, I_k\right] &= \mathbb{E}\left[I_k^2 + 2I_k(I_{k+1} - I_k) + ((I_{k+1} - I_k)^2 - s) - sk \,\middle|\, I_0, \dots, I_k\right] \\ &= I_k^2 - sk\end{aligned}$$

The first equality uses $(a + b)^2 = a^2 + 2ab + b^2$, and the second uses the fact that $I_{k+1} - I_k$ is a symmetric random variable independent of $I_0, \dots, I_k$, having variance s. The *Optional stopping theorem* thus implies $I_0^2 = \mathbb{E}[I_{T'}^2 - sT']$, yielding $\mathbb{E}[T'] \le \mathbb{E}[I_{T'}^2]/s$. To bound $\mathbb{E}[I_{T'}^2]$, we recall that $I_{T'}$ has absolute value $\le (m + 1)/2$ with probability ≤ 1 (trivially), and is between L and $L + m$ with probability $\le O(I_0/L)$. Hence $\mathbb{E}[I_{T'}^2] \le m^2 + (L + m)^2 O(I_0/L)$. Since $L \ge m$, we deduce $\mathbb{E}[I_{T'}^2] \le O(s + I_0 L)$. Overall,

$$\mathbb{E}[T'] \le O(1 + I_0 L/s) \le O(1 + I_0\sqrt{r}/m),$$

and the proof is complete. ■

4.1.4 Conjectured optimal algorithm

The 2D-LPHS algorithms presented in the previous sections have the property of not treating both axes symmetrically. For example, Recursive-Hash iterates over several i_0's, and for each of them it makes many queries of the form $x[i_0, j]$ for different j's. Except for not being aesthetic, this asymmetry has other disadvantages. For example, it is not obvious how to generalize these algorithms to higher dimensions. More importantly, these algorithms (that we considered) do not have optimal dependence of δ on d.

We conjecture that the following symmetric algorithm (Random-Walk-Hash) has the optimal performance of $\delta = \widetilde{O}(1/d)$. However, we were not able to rigorously analyze it.

Heuristic performance. Here we heuristically describe why we expect the algorithm Random-Walk-Hash to achieve $\delta = \widetilde{O}(1/d)$.

The main heuristic assumption we make is that each rw-stage(x, d, L, i, j) can be modeled by a random walk on $\mathbb{Z}^2$, starting at (i, j) and having independent steps which are uniformly distributed on each axis as $\sim U(-L, L)$. We further assume that once the two walks of rw-stage(x) and rw-stage(y) are synchronized (collided), they remain synchronized. Moreover, we recall that the output location of a rw-stage(x) is the point visited in this walk having the minimal x-value.

Remark. These assumptions are not precise mainly because we need the steps to be both deterministic and independent (with respect to the input's randomness) of the previous steps. In practice we cannot guarantee independence, since the random walk occasionally runs into loops. We try to break these in a canonical way, which complicates the analysis. If the algorithm would make monotone queries along (at least) one axis (as the one-dimensional algorithm), then it would avoid loops and its analysis would be much simpler. Unfortunately, we do not know how to design such an algorithm with similar performance.

Based on the heuristic assumptions above, an analysis of Random-Walk-Hash would follow from the following two claims:

- Let T be (a random variable measuring) the meeting time of two random walks on $\mathbb{Z}^2$, starting at distance D (in L_1 norm), and making steps uniformly distributed in $\{-L, -L+1, \dots, L\}$

Algorithm 7: RW-STAGE($z \in \Sigma_b^{\mathbb{Z}_n^2}, d \in \mathbb{N}, L \in \mathbb{N}, i \in \mathbb{Z}_n, j \in \mathbb{Z}_n$)

```
Let $\psi_1, \psi_2 \colon \Sigma_b \to \{-L, \ldots, L\}$ be independent random functions
begin
    $P \leftarrow \text{list}()$
    for $s \leftarrow 0 \ldots d-1$ do
        $P[s] \leftarrow (i, j)$
        $v \leftarrow z[i, j]$
        $(i, j) \leftarrow (i + \psi_1(v), j + \psi_2(v))$
        if $(i, j) \in P$ then
            Let $t$ be the only index satisfying $P[t] = (i, j)$
            $k \leftarrow \arg\min_{u \in [t,s]} \{z[P[u]]\}$
            $(i, j) \leftarrow P[k]$
            while $(i, j) \in P$ do
                $j \leftarrow j + 1$
            end
        end
    end
    return $\arg\min_{(i',j') \in P} \{z[i', j']\}$
end
```

Algorithm 8: RANDOM-WALK-HASH($z \in \Sigma_b^{\mathbb{Z}_n^2}, d \in \mathbb{N}$)

```
begin
    $I \leftarrow \lg \lg(d)$
    $d' \leftarrow d/I$
    $(i_0, j_0) \leftarrow$ MIN-HASH($z,\ d'$)
    $(i_1, j_1) \leftarrow$ RW-STAGE($z,\ d',\ d'^{1/4},\ i_0,\ j_0$)
    $(i_2, j_2) \leftarrow$ RW-STAGE($z,\ d',\ d'^{3/8},\ i_1,\ j_1$)
    $(i_3, j_3) \leftarrow$ RW-STAGE($z,\ d',\ d'^{7/16},\ i_2,\ j_2$)
    $\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$
    $(i_I, j_I) \leftarrow$ RW-STAGE($z,\ d',\ \sqrt{d'/2},\ i_{I-1},\ j_{I-1}$)
    return $(i_I, j_I)$
end
```

on both axis. Then $\mathbb{E}[\min(d', T)] \leq O(\sqrt{d'}(L + D/L))$.[7] We say that walks A, B 'meet' in time t if t is minimal so that $\exists i, j \leq t$ with $\text{location}_i(A) = \text{location}_j(B)$.

- The expected (Manhattan) distance between the start and final point of a 2D-walk with d' steps uniformly distributed in $\{-L, -L+1, \ldots, L\}$ on both axis, is $O(L \cdot \sqrt{d'})$.

Let us analyze the first few stages of Random-Walk-Hash using these claims (which we do not prove here).

Just after the first stage, which is Min-Hash, the walks of Random-Walk-Hash(x) and Random- Walk-Hash(y) are synchronized except for probability $O(1/\sqrt{d'})$ (Lemma 4.1), and in case of this failure event, the distance of the two walks has expected value $O(\sqrt{d'})$.

At the second stage, rw-stage$(x, d', d'^{1/4})$, the initial distance between the walks is $D = O(d'^{1/2})$ and $L = d'^{1/4}$. Using the above claims, and a Markov inequality, the random walks would synchronize except for probability $O(D/L + L)\sqrt{d'}/d' = O(d'^{-1/4})$, and in case of failure, the expected distance is $O(d'^{3/4})$.

At the third stage, rw-stage$(x, d', d'^{3/8})$, $D = O(d'^{3/4}), L = d'^{3/8}$ and the failure probability becomes $O(d'^{3/8})/\sqrt{d'} = O(d'^{-1/8})$, and the distance upon failure is $= O(d'^{7/8})$.

Continuing this heuristic to later stages we get that the total failure probability, which is the product of failure probabilities of all the stages, is $2^{O(I)}/d' = \widetilde{O}(d^{-1})$.

4.2 Lower bounds on 2D-LPHS algorithms

In this section we prove Theorem 1.4 for $k = 2$ (i.e., 2D-LPHS). The proof for any other value of $k > 1$ is similar. In particular, we show that any 2D-LPHS algorithm satisfies $\delta \geq \Omega(1/d)$, given $n \geq 2d$.

Lemma 4.6 (Bigger shifts). *Let h be a 2D (n, b, d, δ)-LPHS. Let $r'_1, r'_2 \in \mathbb{N}$, and $y' = x' \ll (r'_1, r'_2)$ where $x' \in \Sigma_b^{\mathbb{Z}_n^2}$ is a uniformly random string. Then,*

$$\Pr\left[h(x', d) - h(y', d) \neq (r'_1, r'_2)\right] \leq \max\{r'_1, r'_2\}\delta.$$

Proof. Write $I = \max\{r'_1, r'_2\}$ and set $r_1'^{(i)} = \min(r'_1, i), r_2'^{(i)} = \min(r'_2, i)$. Note $\forall i$: $(r_1'^{(i+1)} - r_1'^{(i)}), (r_2'^{(i+1)} - r_2'^{(i)}) \in \{0, 1\}$. Define the strings $x'_i \in \Sigma_b^{\mathbb{Z}_n^2}$ by $x'_i = x' \ll (r_1'^{(i)}, r_2'^{(i)})$ for $i = 0, \ldots, I$. Since each x'_i is a uniformly random function (because x' is), we may use (2) to deduce

$$\delta_i \doteqdot \Pr\left(A(x'_i, d) - A(x'_{i+1}, d) \neq (r_1'^{(i+1)} - r_1^{(i)}, r_2'^{(i+1)} - r_2'^{(i)})\right) \leq \delta.$$

Notice $x' = x'_0$ and $y' = x'_I$. Using a union-bound argument, we conclude

$$\Pr\left[A(x', d) - A(y', d) \neq (r'_1, r'_2)\right] \leq \sum_{i=0}^{I-1} \delta_i \leq I\delta$$

as required. ■

The following lemma implies Theorem 1.4 for $k = 2$.

[7] The parameters are chosen so that the parties meet within an expected number of $O(\sqrt{d'}(L + D/L))$ steps, while they do not meet within d' steps with probability $O((L + D/L)/\sqrt{d'})$.

Lemma 4.7. *For $n > 2d$ and any $b > 0$, every 2D (cyclic or non-cyclic) (n, b, d, δ)-LPHS satisfies $\delta \geq 1/(3d)$.*

Proof. Consider the set of queries P made to a uniformly random string $x \in \Sigma_b^{\mathbb{Z}_n^2}$ by an (n, b, d, δ)-LPHS $h(x, d)$. That is, P is a random variable whose values are sets of sizes $\leq d$ of (i, j) pairs.

Let $x', y' \in \Sigma_b^{\mathbb{Z}_n^2}$ be two uniformly random strings related by $y' = x' \ll (r_1', r_2')$ for independent uniform variables $r_1', r_2' \sim \{0, 1, \ldots, 2d\}$. Using Lemma 4.6,

$$\begin{aligned}\Pr[h(x', d) - h(y', d) \neq (r_1', r_2')] &= \mathop{\mathbb{E}}_{r_1', r_2'} \left[\Pr\left[h(y', d) - h(x', d) \neq (r_1', r_2') \,\middle|\, r_1', r_2'\right]\right] \\ &\leq \mathop{\mathbb{E}}_{r_1', r_2'} \left[\max(r_1', r_2')\delta\right] \leq 2d\delta.\end{aligned} \tag{7}$$

Let $P_{x'}, P_{y'}$ be copies of P which are the sets of queries issued by $h(x', d), h(y', d)$, respectively. Generally, the random variables $P_{x'}, P_{y'}$ are dependent. However, we are going to see they are only slightly dependent. Indeed, suppose P_1, P_2 are two values of P. We claim that given specific values of r_1', r_2' (call these r_1'', r_2'') so that P_1 and (the Minkowski sum) $P_2 + \{(r_1'', r_2'')\}$ are disjoint, we have

$$\Pr\left[P_{x'} = P_1 \wedge P_{y'} = P_2 \,\middle|\, (r_1', r_2') = (r_1'', r_2'')\right] = \Pr\left[P = P_1\right] \cdot \Pr\left[P = P_2\right]. \tag{8}$$

This is because the event $P_{x'} = P_1$ depends only on $x'\big|_{P_1}$ (i.e. lies inside the σ-algebra generated by $x'\big|_{P_1}$)[8], which is independent of $y'\big|_{P_2}$, given that P_1 and (the Minkowski sum) $P_2 + \{(r_1'', r_2'')\}$ are disjoint, as different entries of x' are independent. Consider the 'conditional' random variable

$$X_{r_1'', r_2'', P_1, P_2} \div \left[h(x', d) - h(y', d) \,\middle|\, r_1' = r_1'', r_2' = r_2'', P_{x'} = P_1, P_{y'} = P_2\right],$$

which is defined on the part of the probability space in which $r_1' = r_1'', r_2' = r_2'', P_{x'} = P_1, P_{y'} = P_2$. We use the law of total probability to compute

$$\begin{aligned}&\Pr[h(x', d) - h(y', d) = (r_1', r_2')] \\ &\quad= \mathop{\mathbb{E}}_{r_1', r_2', P_{x'}, P_{y'}} \left[\Pr\left[h(x', d) - h(y', d) = (r_1', r_2') \,\middle|\, r_1', r_2', P_{x'}, P_{y'}\right]\right] \\ &\quad= \mathop{\mathbb{E}}_{r_1', r_2'} \left[\sum_{P_1, P_2} \Pr\left[X_{r_1', r_2', P_1, P_2} = (r_1', r_2')\right] \cdot \Pr\left[P_{x'} = P_1, P_{y'} = P_2 \,\middle|\, r_1', r_2'\right]\right] \\ &\quad\overset{(a)}{\leq} \underbrace{\mathop{\mathbb{E}}_{r_1', r_2'} \left[\sum_{\substack{P_1, P_2 : \\ ((r_1', r_2') \in P_1 - P_2)}} \Pr\left[P_{x'} = P_1, P_{y'} = P_2 \,\middle|\, r_1', r_2'\right]\right]}_{Q_1} + \\ &\quad\quad+ \underbrace{\mathop{\mathbb{E}}_{r_1', r_2'} \left[\sum_{\substack{P_1, P_2 : \\ ((r_1', r_2') \notin P_1 - P_2)}} \Pr\left[X_{r_1', r_2', P_1, P_2} = (r_1', r_2')\right] \cdot \Pr[P = P_1] \Pr[P = P_2]\right]}_{Q_2} \\ &\quad\overset{(b)}{\leq} \frac{d^2}{(2d+1)^2} + \frac{1}{(2d+1)^2} = \frac{d^2 + 1}{(2d+1)^2},\end{aligned}$$

[8] If the LPHS is probabilistic, then we should add the algorithm's randomness into the σ-algebra. Since this randomness is independent of all other random variables, the proof applies verbatim.

where $P_1 - P_2$ is a Minkowski difference. Inequality (a) follows from (8) and the fact that probabilities are upper bounded by 1. Inequality (b) is the key argument. To bound Q_1 (by $d^2/(2d+1)^2$) we use

$$Q_1 + \underbrace{\mathop{\mathbb{E}}_{r_1',r_2'}\left[\sum_{\substack{P_1,P_2:\\((r_1',r_2')\notin P_1-P_2)}} \Pr[P=P_1]\Pr[P=P_2]\right]}_{Q_3} = 1.$$

Exchanging summation order and using $|P_1 - P_2| \leq |P_1| \cdot |P_2| \leq d^2$, which holds since $h(x,d)$ makes at most d queries, we see that $Q_3 \geq 1 - d^2/(2d+1)^2$. Notice we use here $n > 2d$. This proves $Q_1 \leq d^2/(2d+1)^2$.

To bound Q_2, we note that every (P_1, P_2) contributes at most

$$\Pr[P=P_1]\Pr[P=P_2]/(2d+1)^2$$

to Q_2. To see this, observe that the distribution of the random variable X_{r_1'',r_2'',P_1,P_2} does not depend on the particular value of r_1'', r_2'' (given that $P_1 + (r_1'', r_2'')$ is disjoint from P_2), and this random variable always attains a single value (that is, a random variable X and a set E always satisfy $\sum_{e\in E}\Pr[X=e] \leq 1$). Hence

$$Q_2 \leq \sum_{P_1,P_2} \frac{\Pr[P=P_1]\Pr[P=P_2]}{(2d+1)^2} = \frac{1}{(2d+1)^2}.$$

Overall, we deduce $2d\delta + (d^2+1)/(2d+1)^2 \geq 1$, implying $\delta \geq 3/(8d)$. ■

Remark (Extending Lemma 4.7 to higher dimensions). The proof of Lemma 4.7 for $k = 2$ readily extends to a lower bound on the error probability of any k-dimensional LPHS with $k > 2$ (the case $k = 1$ follows from the lower bound in [23] and our generic model equivalence with LPHS). Concretely, for a k-dimensional LPHS we have $\delta \geq 1/(3d^{2/k})$ whenever $n > (2d)^{2/k}$, implying Theorem 1.4.

The extension to a general dimension k requires the following modifications. First, we use a distance-extension lemma, analogous to Lemma 4.6, in a way similar to (7). This step reduces our task to showing that no algorithm can synchronize on random inputs $x, y \in \Sigma_b^{\mathbb{Z}_n^k}$ with probability higher than, say 1/2, where y is a random k-dimensional shift of x by about $(2d)^{2/k}$ in every axis.

Then, we observe that in the event that the LPHS applied on x and y queries disjoint input cells (we think of x and y as inlaid in a common landscape), synchronization is unlikely, as expressed by the bound on Q_2 in the proof of Lemma 4.7. Hence, the synchronization probability is dominated by the probability that the LPHS queries a shared input cell. To bound this latter probability we use a birthday-paradox argument similar to the bound on Q_1 in the proof of Lemma 4.7: there are at least $((2d)^{2/k})^k = 4d^2$ possible shifts, while there are only $d \times d$ pairs of queries that may collide – any of the d queries made to x may collide with any of the d queries made to y. It follows that there is a probability of at most 1/4 to have a shared query, concluding the argument.

5 LPHS for Worst-Case Inputs

The basic definition of LPHS provides guarantees for *random* input strings. This directly aligns with some applications, where the strings to synchronize can be chosen in such manner (e.g., when broadcasting a random synchronization string). In many other application settings, however, we may wish to perform locality-sensitive hashing on inputs whose structure is not uniform, e.g. location coordination given substrings of an existing DNA string or image. Here, it is not sufficient to provide small error probability over a random input; rather, the input will be fixed, and we will wish to obtain small error over a random choice of hash function. In this section, we address this notion of locality-preserving hashing for shifts (LPHS) for *worst-case inputs.*

The main result in this section is showing how to use an underlying LPHS for average-case inputs to obtain LPHS for worst-case inputs that do not exhibit too much regularity (in which case one cannot hope to synchronize shifts effectively). Our approach first embeds the input space endowed with shift metric into an intermediate space over a larger alphabet also endowed with shift metric, with the promise of *all symbols being distinct.* This can be viewed as an analog of approaches taken, e.g. by [19], for the case of edit distance and so-called *Ulam distance* over permutations; however, for our relaxation to shift metric as opposed to edit distance, we can accommodate weaker restrictions on input strings.

In the following subsections, we address LPHS for worst-case inputs in the settings of cyclic and non-cyclic shifts, respectively. For simplicity, we restrict attention to the case of alphabet $\{0,1\}$; that is, inputs $x \in \{0,1\}^n$.

5.1 Cyclic LPHS for Worst-Case Inputs

We begin with the somewhat simpler setting of cyclic shifts. To ensure that our worst-case definition extends from a single shift to multiple shifts, it will be convenient to define the following closure operator.

Definition 5.1. *[Closure under cyclic shifts] For a set of inputs $X \subseteq \{0,1\}^n$, we let*

$$\hat{X} = \{x \lll i \,:\, x \in X,\, 0 \le i < n\}.$$

Definition 5.2. *[Worst-case cyclic LPHS] Let $X \subseteq \{0,1\}^n$ be a set of inputs. A family $\mathcal{H}$ of hash functions $h : \{0,1\}^n \to \mathbb{Z}_n$ is an (n,d,δ)-*cyclic LPHS for worst-case inputs in X *if each h makes d (adaptive) queries (of the form $x[i]$), for every $x \in \hat{X}$, it holds that*

$$\Pr_{h \in_R \mathcal{H}}[h(x) \neq h(x \lll 1) + 1] \le \delta.$$

Unlike the case of LPHS for random inputs, in the worst-case setting the choice of hash function must necessarily be randomized in order to achieve low per-input error.

A first observation is that one cannot hope to synchronize shifts of inputs x that exhibit too much regularity: for example, the all-0 string. We formalize this requirement as α*-goodness*, such that any cyclic shift of x differs in at least an α-fraction of positions from x. In what follows, $\Delta(x,y)$ denotes the Hamming distance of strings $x, y \in \{0,1\}^n$.

Definition 5.3 (α-good inputs)**.** *An input $x \in \{0,1\}^n$ is said to be* α-good *if for every $i \in [n]$ it holds that $\Delta(x \lll i, x) \ge \alpha n$. We denote the set of all α-good inputs in $\{0,1\}^n$ by Good_n^α.*

Note that Good_n^α is closed under shifts. The above goodness requirement is necessary, in the sense that even if it is only violated by a single shift $0 < i < n$, it requires a notable deterioration of the LPHS parameters. For example, if x is a random string with period $n/2$ (i.e., $x = (x \lll n/2)$), then for any choice of hash function h necessarily $h(x) = h(x \lll n/2)$, meaning one cannot achieve δ better than $2/n$.

Remark (Biased random inputs). As an example application, inputs that occur as the result of *biased* random sampling satisfy the above α-goodness with high probability, for α that is a function of the bias.

Namely, consider the distribution $\mathcal{D}_\beta$ over $\{0,1\}^n$ where each bit x_i is selected as biased i.i.d. bits, equalling 1 with (constant) probability $0 < \beta < 1$. Then if n is prime (or, more generally, without too many factors), then for every constant $\alpha < \min\{\beta, 1-\beta\}$, it holds that

$$\Pr_{x \in_R \mathcal{D}_\beta}[x \in \mathsf{Good}_n^\alpha] \geq 1 - ne^{-\Omega(n)}.$$

To see this, observe that since n is prime, then for any given nonzero shift $0 < i < n$, it holds that $(1, 1+i, 1+2i, \ldots, 1+ni)$ forms a permutation of $[n]$. Consider the string $y := (x_1, x_{1+i}, x_{1+2i}, \ldots, x_{1+ni})$, formed by appropriately permuting $x \in \{0,1\}^n$. Then $\Delta(x, x \lll i)$ is equal to the number of positions $j \in [n]$ for which $y_j \neq y_{j+1}$ (taking $y_{n+1} := y_1$). For each $j \in [n]$, conditioned on the values of $y_1, \ldots, y_j$, the value of y_{j+1} is randomly sampled with bias β. In particular, for any such prefix, the probability that $y_{j+1} \neq y_j$ is at least $\min\{\beta, 1-\beta\}$. The claim thus follows by a Chernoff bound, together with a union bound over choices of $i \in [n]$.

Our main result is a black-box construction of worst-case LPHS for the set of inputs Good_n^α, from any LPHS for random inputs, with small overhead (poly-logarithmic in the query complexity).

At a high level, the construction reduces to the case of random inputs, by (1) considering an intermediate input value $x' \in (\{0,1\}^b)^n$ over a *larger* alphabet $\{0,1\}^b$, whose symbols are formed from a random tiling of $b \in \omega(\log n)$ bits of x, and then (2) converting x' to a new input $y \in \{0,1\}^n$ using an n-wise independent hash $\gamma : \{0,1\}^b \to \{0,1\}$. This approach is reminiscent of the generic alphabet-enlarging procedure used in Property 2.7 (and other prior works) via shingling, except with randomly chosen tile windows to accommodate worst-case inputs. As we will show, the α-goodness of the original input x will ensure with high probability over the choice of tiling that all tile-symbols of x' are distinct elements of $\{0,1\}^b$. Then by n-wise independence of γ, the resulting input $y \in \{0,1\}^n$ will be *uniform.* Note that if we begin with an LPHS which makes only d queries to the (random) input, then we require only d-wise independence of γ.

Proposition 5.4 (Worst-case cyclic LPHS for α-good inputs)**.** *Assume there exists an (n,d,δ)-cyclic LPHS for random inputs. Then, for every $0 < \alpha(n) \leq 1$ and $b \geq \omega(\log n)$, there exists a (n,d',δ')-cyclic LPHS for worst-case inputs in Good_n^α, with $d' = O(d \cdot b)$ and $\delta' = \delta + n^2(1-\alpha)^b$.*

Plugging in the LPHS construction from Theorem 3.5 (quoted from [23]), together with $b \in \omega(\log n) \cap \log^{O(1)}(n)$ yields the following corollary.

Corollary 5.5. *For every constant $0 < \alpha \leq 1$, there exists a worst-case* cyclic *(n,d,δ)-LPHS for Good_n^α with $d = \tilde{O}(\sqrt{n})$ and $\delta = O(1/n)$.*

We now proceed to prove Proposition 5.4.

Proof. Let $b = b(n)$. Consider the following hash function family $\mathcal{H} = \{h_{S,\gamma}\}$, indexed by a subset $S \subset [n]$ of size b, and a hash function $\gamma : \{0,1\}^b \to \{0,1\}$ from an n-wise independent hash family. Sampling a hash function $h_{S,\gamma}$ from $\mathcal{H}$ will consist of randomly selecting $S \subset [n]$ and sampling γ from the n-wise independent hash family.

The worst-case LPHS will use as a black box an underlying (n,d,δ)-LPHS $h^* : \{0,1\}^n \to \mathbb{Z}$ on *random* inputs.

Evaluation of $h_{S,\gamma}(x)$, for $x \in \{0,1\}$:

1. For $i \in [n]$, denote $i + S = \{i + s \mod n \mid s \in S\}$ and $x_{i+S} = (x_j)_{j \in i+S} \in \{0,1\}^b$.
2. Execute the algorithm for average-case LPHS h^*. For each index $i \in [n]$ that h^* wishes to query, perform the following:
   (a) Query b indices of x, corresponding to the set $(i + S) \subset [n]$. Denote the corresponding bit string by $x_{i+S} \in \{0,1\}^b$.
   (b) Let $y_i := \gamma(x_{i+S}) \in \{0,1\}$. Submit y_i to h^*, as the answer to query $i \in [n]$.
3. Let $z \in \mathbb{Z}$ denote the output of h^*. Output z.

Note that the query complexity of $h_{S,\gamma}$ is precisely b times the query complexity of h^*. We now analyze the correctness of the resulting worst-case LPHS.

Claim 5.6. *Let $x \in \mathsf{Good}_n^\alpha$. With overwhelming probability over a random choice of subset $S \subset [n]$ of size $b(n) \leq n$, all S-tiles x_{i+S} of x are distinct. Namely,*

$$\Pr_{h_{S,\gamma} \in_R \mathcal{H}}[\exists i \neq i' \in [n], x_{i+S} = x_{i'+S}] \leq n^2 \cdot (1 - \alpha)^b.$$

Proof (of Claim 5.6). Fix $i < i' \in [n]$. By α-goodness of the input x, it holds that $\Delta(x \ll i, x \ll i') = \Delta(x, x \ll (i' - i)) \geq \alpha n$; that is, there exists a subset $T \subseteq [n]$ of size $|T| \geq \alpha n$ for which $x_{i+j} \neq x_{i'+j}$ for every $j \in T$. Over the choice of S, $\Pr_S[x_{i+S} = x_{i'+S}] \leq \Pr_S[S \cap T = \emptyset] \leq (1 - \alpha)^b$. The claim thus holds by a union bound over pairs $i, i' \in [n]$. ■

Conditioned on distinctness of all S-tiles x_{i+S} queried by the algorithm h^*, then by n-wise independence of the hash family γ, it holds that the computed output bits $y_i := \gamma(x_{i+S})$ will be independently random. That is, conditioned on the above event, the hash function $h_{S,\gamma}$ will err with probability identical to that of h^* on a random input. The proposition follows.

■

5.2 Non-Cyclic LPHS for Worst-Case Inputs

We next present and achieve a notion of *non-cyclic* LPHS for worst-case inputs, up to a maximum shift R. In this case, the alphabet-tiling procedure must be adjusted, as symbols outside the shift window will be lost. Instead, we consider a modified approach, in which the tiles for each index i are chosen as a random subset from within a W-size window beginning at index i, for $W < n$. This makes for a slightly more complex "goodness" condition for worst-case input strings.

Definition 5.7 (Worst-Case LPHS). *Let $X \subseteq \{0,1\}^n$ be a set of inputs. A family $\mathcal{H}$ of hash functions $h : \{0,1\}^n \to \mathbb{Z}_n$ is a (non-cyclic) (n, d, δ)-LPHS for worst-case inputs in X, up to shift bound R, if each h makes d (adaptive) queries (of the form $x[i]$), and for every $x \in X$, and every $1 \leq r \leq R$ it holds that*

$$\Pr_{h \in_R \mathcal{H}}[h(x) \neq h(x \lll r) + r] \leq \delta.$$

We formalize the desired input non-regularity requirement via (α, W)-*goodness*, parameterized by a "window size" $W \in [n]$ and difference parameter $0 < \alpha \leq 1$, such that each of the length-W substrings x differ pairwise in at least α fraction of their symbols. In what follows, we denote by $x_{i+[W]}$ the W-substring of x beginning at index i (which may have length less than W if $(i+W) < n$), and by $\Delta(x', y')$ the Hamming distance of strings $x', y' \in \{0,1\}^\ell$ (where $1 \leq \ell \leq W$).

Definition 5.8 ((α, W)-good inputs)**.** *An input $x \in \{0,1\}^n$ is said to be* (α, W)-good *if for every $i < i' \in \{1, \ldots, n - W/2\}$ it holds that $\Delta(x_{i+[W]}, x_{i'+[W]}) \geq \alpha |x_{i'+[W]}|$.*[9] *We denote the set of all (α, W)-good inputs in $\{0,1\}^n$ by $\mathsf{Good}_n^{\alpha,W}$.*

Remark (Biased random inputs)*.* As with the cyclic case, we similarly have that biased per-bit random inputs satisfy (α, W)-goodness with high probability for sufficiently large W and α that is a function of the bias.

Namely, consider the same distribution $\mathcal{D}_\beta$ over $\{0,1\}^n$ where each bit x_i is selected as biased i.i.d. bits, equalling 1 with (constant) probability $0 < \beta < 1$. Then for every constant $\alpha < \min\{\beta, 1 - \beta\}$, it holds that

$$\Pr_{x \in_R \mathcal{D}_\beta}[x \in \mathsf{Good}_n^{\alpha,W}] \geq 1 - n^2 e^{-\Omega(W)}.$$

(Note here that n need not be prime.) The argument here follows similarly to the cyclic case, except without the complication of cyclic wraparound.

Consider a fixed pair $i < j \in [n - W/2]$. We analyze $\Delta(x_{i+[W]}, x_{j+[W]}) = |\{\ell \in [W] : x_{i+\ell} \neq x_{j+\ell}\}|$. We may again reorder the elements of x, this time into a collection of $i' := j - i$ sequences, the corresponding shifted cosets of $(x_i, x_{i+i'} x_{i+2i'}, \cdots)$ starting with the corresponding $(j - i)$th value x_ℓ, $\ell \in \{i, \ldots, j - 1\}$, and containing all i'-multiple instances, up to $j + W$. For any such sequence, and any fixed choice of values in the prefix of the sequence, the probability that the following term is equal to the previous is at least $\min\{\beta, 1-\beta\}$. Altogether, all indices in the range $\{i, \ldots, i+W\} \cup \{j, \ldots, j+W\}$ thus contribute a fresh term, aside from the initial $(j-i)$ that served as start points of the sequences. The claim thus follows by a Chernoff bound, together with union bound over pairs i, j. Note that for $W \in \omega(\log n)$, the resulting error probability is negligible.

As with the cyclic case, we demonstrate an analogous black-box construction of worst-case *non-cyclic* LPHS for the set of inputs $\mathsf{Good}_n^{\alpha,W}$, from any *non-cyclic* LPHS for random inputs, with small overhead. We state the following proposition in terms of an arbitrary window-size parameter W, and then discuss relevant settings afterward.

Proposition 5.9 (Worst-Case LPHS For (α, W)-Good Inputs)**.** *Assume there exists an (n, d, δ)-non-cyclic LPHS with shift bound R for* random *inputs. Then, for every $0 < \alpha(n) \leq 1$, $1 < W(n) \leq n$, and $b \geq \omega(\log n)$, there exists an (n', d', δ')-non-cyclic LPHS with shift bound R for* worst-case *inputs in $\mathsf{Good}_{n'}^{\alpha,W}$, with $n' = n + W$, $d' = O(d \cdot b)$ and $\delta' = \delta + e^{-\Omega(b)}$.*

Proof. Let $b = b(n)$, $W = W(n)$. Consider the hash function family $\mathcal{H} = \{h_{S,\gamma}\}$ as defined within the proof of Proposition 5.4, indexed by a subset $S \subset [W]$ of size b, and a hash function $\gamma : \{0,1\}^b \to \{0,1\}$ from an n-wise independent hash family. Here, we have the following two differences from the cyclic case:

- Sampling a hash function $h_{S,\gamma}$ from $\mathcal{H}$ will consist of randomly sampling γ from the n-wise independent hash family (as before), but now sampling the subset $S \subset [W]$ instead of $[n]$.
- The worst-case LPHS will use as a black box an underlying (n, d, δ)-*non-cyclic* LPHS $h^* : \{0,1\}^n \to \mathbb{Z}$ on random inputs (as opposed to cyclic LPHS).

[9] Note for $i \leq n - W$, the length $|x_{i+[W]}| = W$, but for $(n - W) < i' \leq (n - W/2)$, then $W/2 \leq |x_{i'+[W]}| = (n - i') < W$. That is, we consider also substring windows that "hang over" the edge of the string x up to $W/2$.

Note that (as before) the query complexity of $h_{S,\gamma}$ is precisely b times the query complexity of h^*. We now analyze the correctness of the resulting worst-case LPHS.

First, note that for any $x \in \mathsf{Good}_{n'}^{\alpha,W}$ and $1 \leq r \leq R$, then with high probability over $y \in_R \{0,1\}^r$, it holds that the concatenated string $z := x||y \in \mathsf{Good}_{n'+r}^{\alpha/2,W}$. Indeed, consider $\Delta(z_{i+[W]}, z_{j+[W]})$ for various choices of $i, j \in [n'+r-W/2]$. From the definition of $\mathsf{Good}_{n'}^{\alpha,W}$, the required relative distance $\alpha/2$ holds for pairs with $i, j \in [n'-W/2]$. For any $i \in [n'+r-W/2]$ and $j \in \{(n'-W/2+1), \ldots, (n'+r-W/2)\}$, then the string z_j is composed of at least $W/2$ *uniform* bits (coming from y); the required relative distance $\alpha/2$ thus holds except with probability upper bounded by $e^{-\Omega(W)}$, by the (biased) random inputs Remark above.

The proposition then follows from the following claim, which collectively implies that our transformation reduces (with small error) directly to the underlying average-case LPHS: (1) First, that aside from probability $n^2(1-\alpha)^b$ over $h_{S,\gamma}$, queries into a "good" input in $\{0,1\}^{n+W}$ get mapped to queries into a *uniform* input in $\{0,1\}^n$; (2) Second, that the transformation preserves the (non-cyclic) shift metric.

Claim 5.10. *For given S, γ, define $f_{S,\gamma} : \{0,1\}^{n+W} \to \{0,1\}^n$ via $f_{S,\gamma}(x) = (\gamma(x_{i+S}))_{i\in[n]}$. For every $x \in \mathsf{Good}_{n'}^{\alpha,W}$, the following hold:*

1. *With high probability over a random choice of subset $S \subset [W]$ of size $b(n) \leq n$, all "S-tiles" x_{i+S} of x are distinct. Namely,*

$$\Pr_{S\in_R\binom{W}{b}}[\exists i \neq i' \in [n], x_{i+S} = x_{i'+S}] \leq n^2 \cdot (1-\alpha)^b.$$

   As a consequence, then with probability $1 - n^2(1-\alpha)^b$ over the choice of S, it holds that $\{f_{S,\gamma}(x)\}_\gamma \equiv U_{\{0,1\}^n}$, where the distribution $\{f_{S,\gamma}(x)\}_\gamma$ is over the choice of γ, and $U_{\{0,1\}^n}$ denotes the uniform distribution over $\{0,1\}^n$.

2. *The map $f_{S,\gamma}$ preserves the (non-cyclic) shift metric. That is, for every $1 \leq r \leq R$,*

   - *Agreement window: For every S, γ, and every index $(1+r) \leq i \leq (n-r)$, it holds that $f_{S,\gamma}(x)_i = f_{S,\gamma}(x \lll r)_{i+r}$ with probability 1 (over randomness of $\lll$).*
   - *Outside agreement window:*

$$\{f_{S,\gamma}(x \lll r)\}_\gamma \equiv \left\{ \left(f_{S,\gamma}(x)_{[1+r,\ldots,n]}, U_{\{0,1\}^r} \right) \right\}_\gamma.$$

Proof. We address each part of the claim.

1. Fix $i < i' \in [n]$. By (α, W)-goodness of the input x, it holds that $\Delta(x_{i+[W]}, x_{i'+[W]}) \geq \alpha n$; that is, there exists a subset $T \subseteq [W]$ of size $|T| \geq \alpha n$ for which $x_{i+j} \neq x_{i'+j}$ for every $j \in T$. Over the choice of S, $\Pr_S[x_{i+S} = x_{i'+S}] \leq \Pr_S[S \cap T = \emptyset] \leq (1-\alpha)^b$. The claim thus holds by a union bound over pairs $i, i' \in [n]$.

2. Note that $x \in \{0,1\}^{n+W}$ and $x \lll r$ satisfy the property $x_i = (x \lll r)_{i+r}$ for all $i \in [n+W-r]$. Since $S \subset [W]$, then in particular $x_{i+S} = (x \lll r)_{i+r+S}$ for every $i \in [n-r]$. This implies the desired equality within the agreement window.

   Outside the agreement window, the required property holds directly by applying the Claim part (1) to the appended string $z := x||y$ for $y \in_R \{0,1\}^r$, which can be done since (as argued above), this string satisfies $z \in \mathsf{Good}_{n'+r}^{\alpha/2,W}$.

■

■

Plugging in the non-cyclic LPHS construction from Theorem 3.5, together with $b \in \omega(\log n) \cap \log^{O(1)}(n)$ yields the following corollary.

Corollary 5.11. *For every constant $0 < \alpha \leq 1$ and $W \in \omega(\log n)$, there exists a worst-case* non-cyclic *(n, d, δ)-LPHS for $\mathsf{Good}_n^{\alpha,W}$ with $d = \tilde{O}(\sqrt{n})$ and $\delta = O(1/n)$.*

Let $d_S(x, y)$ denote non-cyclic shift distance, defined as the distance between x and y on the De Bruijn Graph. Then the above results imply the following (probabilistic, bounded-distance) isometric embedding of d_S to the line.

Corollary 5.12. *Let $0 < \alpha \leq 1$, $R \in [n]$ be a shift distance bound, $W \in \omega(\log n)$, and $d < n^{1/2}$. Then, there exists a family of hash functions $\mathcal{H}$ such that*

1. *Each $h \in \mathcal{H}$ makes d queries to the input;*

2. *For all $x, y \in \mathsf{Good}_n^{\alpha,W}$ with $d_S(x, y) = r \leq R$, we have $\Pr[|h(x) - h(y)| \neq r] \leq \tilde{O}(r/d^2)$.*

6 Applications

In this section we present several cryptographic and algorithmic applications that motivate the different LPHS flavors studied in this work. In all applications, there are two or more parties that have partially overlapping views of a large object, and the goal is to measure in *sublinear time* the relative misalignment between the views, with low failure probability and without direct interaction between the parties. Before discussing the applications in detail, we give a taxonomy of the kinds of LPHS instances on which they depend.

- **1-dimensional vs. k-dimensional.** The first application, to packed homomorphic secret sharing, requires k-dimensional LPHS for $k \geq 2$. The other applications can apply in any dimension, but the 2-dimensional variant seems most useful in the context of natural use cases that involve 2-dimensional objects (such as digital images).

- **Cyclic vs. non-cyclic.** While some of the applications can also meaningful in the cyclic case, the non-cyclic one is needed to capture the "partially overlapping views" scenario.

- **Random vs. worst-case.** In the application to packed homomorphic secret sharing from a generic group, the object is a huge and locally random mathematical object. Hence the default random-input variant of LPHS suffices. In other applications, which may involve physical objects or digital documents, the worst-case variant is needed. While the latter must inevitably exclude objects that are close to being highly periodic, this is not an issue for most natural use cases.

- **Small shift vs. big shift.** When the misalignment is small, one could apply a MinHash-based $(d, \tilde{O}(1/d))$-LPHS and reduce its failure probability via repetition. This solution is not suitable for applications that depend on the simple metric structure of the LPHS output, and in any case leads to inferior communication rate (for the same failure probability and running time) compared to applying a single instance of a $(d, \tilde{O}(1/d^2))$-LPHS. In applications that require detecting an arbitrary misalignment in sublinear time, a $(d, \tilde{O}(1/d))$-LPHS does not suffice at all since the failure probability is multiplied by the shift amount.

We proceed with the details of the applications, starting with cryptographic applications.

6.1 Packed Homomorphic Secret Sharing

In this section, we present a cryptographic application of k*-dimensional* LPHS for constructing a "packed" version of homomorphic secret sharing (HSS) from cryptographically hard groups. Roughly speaking, k-packed HSS can share a vector of k values at the same communication cost as sharing a single value. This improved communication comes at the expense of a computational overhead that we optimize using two distinct LPHS-based approaches. Our main approach relies on k-dimensional LPHS. Since even for $k = 2$ we do not have a provable optimal construction of k-dimensional LPHS, we also present an alternative approach that relies on 1-dimensional LPHS and performs better in some parameter regimes.

The remainder of this section is composed as follows. In Section 6.1.1 we introduce a k-dimensional variant of the DDL problem and in Section 6.1.2 we show how to realize it from k-dimensional non-cyclic LPHS. Then, in Section 6.1.3, we present the application of k-dimensional DDL to reducing the communication complexity of group-based HSS, as well as an alternative construction based on 1-dimensional embedding.

6.1.1 Multidimensional DDL

In this section we describe a k-dimensional generalization of DDL (kD-DDL for short) that we will use as an intermediate step towards constructing packed HSS. Since DDL has already found applications beyond the context of HSS [26, 29, 25], one may expect the same for the k-dimensional variant.

While perhaps the most direct extrapolation of DDL to k dimensions would correspond to a DDL challenge within k *independent* copies of the group (i.e., $\mathbb{Z}_n^k \hookrightarrow \bigoplus_{i=1}^k G$), for our purposes, we will consider a notion where the k dimensions are embedded within a *single* group G of size $N \gg n$. More explicitly, we will embed $\mathbb{Z}_n^k \hookrightarrow G$ via $(j_1, \ldots, j_k) \mapsto \prod_{i=1}^k g_i^{j_i}$ for k randomly selected generators g_i of G. As this embedding is not injective, its use within a construction of kD-DDL from kD-LPHS will introduce an extra collision error probability; however, when N is much bigger than the number of queries d made by the kD-LPHS algorithm (as will be the case for the HSS application), this collision error probability will be negligible.

As in the case of (1-dimemsional) DDL algorithms defined in Section 3.1, we consider generic algorithms, i.e. operating within the generic group model. We present a version of this model that captures the k-dimensional case below.

Recall that in the 1-dimensional case, the LPHS parameter n was the same as the group size and the LPHS string $x^{(b)}$ represented the sequence of all group elements. Since this will no longer be the case in the k-dimensional variant, from here on we denote the group size by N, and the sequence of labels of group elements by $X^{(b)}$. In the 1-dimensional case, we considered an experiment where a string $X^{(b)} \in \Sigma_b^N$ (representing the group) and a value $v \in \mathbb{Z}_N$ are sampled uniformly; the generic DDL algorithm makes adaptive queries of the form $(\alpha, \beta) \in \mathbb{Z}_N \times \mathbb{Z}_N$, which are answered by $X^{(b)}[\ell_v(\alpha, \beta)] \in \Sigma_b$, where $\ell_v(\alpha, \beta) := \alpha \cdot v + \beta \in \mathbb{Z}_N$, and outputs a value $\gamma \in \mathbb{Z}_N$. The goal of the 1D-DDL algorithm was that the outputs γ, γ' on implicit inputs v and $v' = v + 1$ satisfied $\gamma - \gamma' = 1$.

In the k-dimensional experiment that we define, again, a string $X^{(b)} \in \Sigma_b^{\mathbb{Z}_N}$, representing a single group of order N with generator g, and a value $v \in \mathbb{Z}_N$. are sampled uniformly. In addition, k public "basis elements" $w_1, \ldots, w_k \leftarrow \mathbb{Z}_N$ are randomly selected, representing the (discrete logarithm of) k group generators $g_i = g^{w_i}$. We assume N to be prime, in which case the g_i are distinct group generators with high probability. Given this setup, the generic algorithm A^{G} makes adaptive queries of the form $(\alpha, (\beta_1, \ldots, \beta_k)) \in \mathbb{Z}_N \times \mathbb{Z}_N^k$, which are answered by $X^{(b)}[\alpha v + \sum_{i=1}^k \beta_i w_i]$, namely the string handle of the group element $(g^v)^\alpha \cdot \prod_{i=1}^k g_i^{\beta_i}$.

The output of the algorithm is an integer vector $\gamma \in \mathbb{Z}^k$. Here we use the notation $A^{\mathrm{G,w}}(X^{(b)}, v)$ to indicate that A is a generic algorithm that does not have direct access to $X^{(b)}$ and v, but can have full knowledge of the public basis $w = (w_1, \ldots, w_k)$.

The goal of the kD-DDL algorithm is to similarly detect a shift of 1 on the input in any one of the k dimensions. In this case, such a shift in dimension $i \in [k]$ corresponds to an additive offset of w_i in the discrete logarithm. Concretely, we would like that the outputs γ, γ' on implicit inputs v and $v' = v + w_i$ to satisfy $\gamma - \gamma' = e_i$, the ith unit vector in $\mathbb{Z}^k$.

Note that the error probability of a kD-DDL algorithm can come from multiple sources: its internal coins, the choice of the implicit input v, and the "setup" process of choosing group labels $X^{(b)}$ and the basis w. While for the purpose of driving down the error via repetition it is useful to separate the first source of error from the others (see more below), we will consider the probability space that combines all these random choices for simplicity.

Definition 6.1 (kD-DDL)**.** *A generic kD-DDL algorithm A is an (N, b, d, δ) kD-DDLA if A makes d (adaptive) queries and for every $i \in [k]$,*

$$\Pr[A^{\mathrm{G,w}}(X^{(b)}, v) - A^{\mathrm{G,w}}(X^{(b)}, v + w_i) \neq e_i] \leq \delta,$$

where A^{G} denotes generic access as described above and the probability is over the choice of $X^{(b)}$ from $\Sigma_b^{\mathbb{Z}_N}$ and $v, w_1, \ldots, w_k$ from $\mathbb{Z}_N$.

Remark (Bigger k-dimensional shifts)*.* Analogously to Lemma 2.4, we can use a union bound to deduce a bound on the error probability for bigger k-dimensional shifts. Concretely, for any (N, b, d, δ) kD-DDLA and k-dimensional shift $u = (u_1, \ldots, u_k) \in \mathbb{Z}^k$ we have

$$\Pr\left[A^{\mathrm{G,w}}(X^{(b)}, v) - A^{\mathrm{G,w}}\Big(X^{(b)}, v + \sum_{i=1}^k u_i w_i\Big) \neq u\right] \leq \delta \cdot |u|_1,$$

where $|u|_1$ denotes the ℓ_1-norm of u and the probability space is as in Definition 6.1. In the context of the packed HSS application, this will imply an error that scales with the ℓ_1-norm of the output vector.

As DDL algorithms have a small but non-negligible error, it may be useful to drive the error probability down using independent repetitions, where the internal coins of the algorithm are picked independently but all other sources of randomness are fixed. (This repetition comes at a price of additional communication, since each instance produces different outputs.) For this purpose, one can use a more refined version of Definition 6.1 in which the probability space is only over the internal randomness of $A^{\mathrm{G,w}}$ and the probability bound δ should hold except with negligible probability over all other random choices. The construction we present next indeed satisfies this property.

6.1.2 kD-DDL from kD-LPHS

Our main solution to the kD-DDL problem is based on the *non-cyclic* variant of k-dimensional LPHS from Definition 2.1 (kD-LPHS for short) that we recall below.

Let e_i denote the ith unit vector of length k. For a k-dimensional string $x = x^{(k,b)}$, we denote by $x \lll e_i$ a non-cyclic shift of x by 1 in the ith dimension (e.g., for $k = 2$ and $i = 1$, this corresponds to chopping the top row and adding a random row on the bottom). In the following we will use $\mathbb{Z}_n$ to denote the set of integers $\{0, 1, \ldots, n-1\}$ and will not use its group structure.

Definition 6.2 (Non-cyclic kD-LPHS)**.** *Let $h : \Sigma_b^{\mathbb{Z}_n^k} \to \mathbb{Z}^k$ be a function. We say that h is a non-cyclic (n, b, d, δ) kD-LPHS if h can be computed by making d* adaptive *queries (of the form $x[\beta_1, \ldots, \beta_k]$ for $\beta_i \in \mathbb{Z}_n$) to a k-dimensional input $x = x^{(k,b)} \in \Sigma_b^{\mathbb{Z}_n^k}$ and for every $i \in [k]$:*

$$\Pr_{x \in_R \Sigma_b^{\mathbb{Z}_n^k}} [h(x) \neq h(x \lll e_i) + e_i] \leq \delta.$$

This naturally extends to a probabilistic $h \in_R \mathcal{H}$.

Given any non-cyclic kD-LPHS, we can construct a kD-DDL algorithm with similar parameters. The transformation will not be perfect, but will induce collision probability error that for sufficiently large group sizes N will be negligible.

Theorem 6.3 (From non-cyclic kD-LPHS to kD-DDL)**.** *Let k, n be positive integers and N be a prime such that $N \geq n$. There exists a black-box reduction that converts any (n, b, d, δ)* non-cyclic *kD-LPHS to an $(N, b, d, \delta + d^2/N)$ kD-DDL algorithm. The DDLA is* query-restricted *in the sense that the values $\alpha, \beta_1, \ldots, \beta_k$ for each generic query satisfy $\alpha = 1$ and $\beta_i \in \mathbb{Z}_n$.*

Proof. Given black-box access to an (n, b, d, δ) k-dimensional non-cyclic LPHS denoted by h, we construct the $(N, b, d, \delta + d^2/N)$ k-dimensional DDLA, denoted by A, as follows. The algorithm A simulates h by responding to each query $\beta = (\beta_1, \ldots, \beta_k) \in \mathbb{Z}_n^k$ that h makes into $x = x^{(k,b)}$ with the group symbol read by $A^{\mathrm{G,w}}[X^{(b)}, v]$ on the corresponding generic group query $(\alpha = 1, \beta)$. Once the d queries are completed, A outputs the output of h.

We now bound the error probability of A over the choice of the generators basis $w = (w_1, \ldots, w_k) \in_R \mathbb{Z}_N^k$, challenge value $v \in_R \mathbb{Z}_n$, and the randomness of h and $X^{(b)}$. For $\beta \in \mathbb{Z}_n^k$ (viewed as a vector of integers), let $\langle w, \beta \rangle$ denote the mod-N inner product of w and β. Consider the executions of A on v and $v + w_i$. The key observation is that conditioned on the good event that neither of the two executions contains a colliding set of queries β, β' such that $\langle w, \beta \rangle = \langle w, \beta' \rangle$, the joint distribution of the symbols read by $A^{\mathrm{G,w}}[X^{(b)}, v]$ and $A^{\mathrm{G,w}}[X^{(b)}, v + w_i]$ is identical to that of the symbols read by $h(x)$ and $h(x \lll e_i)$. It follows that conditioned on this good event, the error probability of A is bounded by δ. It thus suffices to show that the probability of the bad collision event is bounded by d^2/N.

For any fixed pair of distinct query vectors $\beta, \beta' \in \mathbb{Z}_n^k$, we have

$$\Pr_{w \in \mathbb{Z}_N^k} [\langle w, \beta \rangle = \langle w, \beta' \rangle] = 1/N$$

(here we use the assumptions that N is prime and $N \geq n$). For each non-colliding query made by A, the value it receives back is a freshly sampled uniform value $X^{(b)}[\ell] \in \Sigma_b$ (for some non-previously-queried index $\ell \in \mathbb{Z}_N$); in particular, this value is independent of the choice of w. Thus, taking a union a bound over the $\binom{d}{2}$ pairs of distinct queries made in an execution of A, we get a collision probability bound of $\binom{d}{2}/N$ in a single execution, and at most $2 \cdot \binom{d}{2}/N < d^2/N$ collision probability on either the execution on v or on $v + w_i$. ■

For example, plugging in the results from Section 4, we get a provable 2D-DDL algorithm that makes d queries (that can be implemented using $\tilde{O}(d)$ group multiplications) and has $\delta = \tilde{O}(d^{-7/8} + d^2/N)$ error probability. Using the conjectured optimal algorithm, the $d^{-7/8}$ term can be replaced by d^{-1}.

The additive error term of d^2/N can be further reduced by applying the worst-case notion of LPHS from Appendix 5 to get robustness against collisions. However, this term is already negligible for a typical choice of parameters. For instance, practical cryptographically hard groups have order $N > 2^{256}$, and so for this term to be significant the running time d should be close to 2^{128}.

6.1.3 From kD-DDL to Packed HSS

We now demonstrate applications of kD-DDL to packed HSS. More formally, we consider homomorphic secret sharing (HSS) with non-negligible error δ, as introduced in [12]:

Definition 6.4 (δ-Homomorphic Secret Sharing). *A (2-party)* δ-Homomorphic Secret Sharing (δ-HSS) *scheme for a class of programs $\mathcal{P}$ over $\mathbb{Z}$ with input space $\mathcal{I} \subseteq \mathbb{Z}$ consists of PPT algorithms* ($\mathsf{HSS.Share}$, $\mathsf{HSS.Eval}$) *with the following syntax:*

- $\mathsf{HSS.Share}(1^\lambda, x)$*: Given security parameter 1^λ and secret input value $\alpha \in \mathcal{I}$, the sharing algorithm outputs secret shares* $(\mathsf{share}_0, \mathsf{share}_1)$*.*

- $\mathsf{HSS.Eval}(i, (\mathsf{share}_i^{(1)}, \ldots, \mathsf{share}_i^{(\rho)}), P, m)$*: Given party index $i \in \{0, 1\}$, the ith secret share for ρ inputs, program $P \in \mathcal{P}$ with ρ input values, and integer $m \geq 2$, homomorphic evaluation outputs $y_i \in \mathbb{Z}_m$, constituting party i's share of an output $y \in \mathbb{Z}_m$.*

The algorithms $(\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ must satisfy the expected homomorphic evaluation correctness $y_0 + y_1 = P(\alpha^{(1)}, \ldots, \alpha^{(\rho)}) \in \mathbb{Z}_m$ for any set of inputs and program $P \in \mathcal{P}$, with probability at least $(1 - \delta)$ over the execution of $\mathsf{HSS.Share}$. In addition, $\mathsf{HSS.Share}$ should satisfy semantic security (i.e., given only the ith share of a sequence of inputs, a polynomial-time bounded adversary cannot distinguish which of two input sequences they were derived from).

We will be interested in δ-HSS for vector-scalar multiplications. That is, let $\mathcal{P}_{\mathsf{blin}\text{-}k}$ denote the class of programs which perform bilinear operations over two variable types: *vectors* over $[M]^k$, and *scalars* over $[M]$. We will maintain notation of scalars denoted by Greek letters and vectors as lowercase roman letters: e.g., $\alpha \in [M], u = (u_1, \ldots, u_k) \in [M]^k$.

More formally, we consider $\mathcal{P}_{\mathsf{blin}\text{-}k}$ the class of programs which act on inputs each of type vector or scalar, and which makes polynomially many of the following operations. Note that all data types are additionally either "Input" or "Non-Input"; to capture the bilinear limitation, multiplication is only allowed between Input scalars and vectors.

- Add Input (or Non-Input) Vectors: $u'' \leftarrow u + u'$ (or $z'' \leftarrow z + z'$).

- Add (Input or Non-Input) Scalars: $\alpha'' \leftarrow \alpha + \alpha'$.

- Multiply Input Scalar and Input Vector: $z \leftarrow \alpha \cdot u$.

- Parse Non-Input Vector as Non-Input Scalars: $(z_1, \ldots, z_k) \leftarrow z$

For example, this class of programs includes many useful statistical computations, such as correlations.

Overview of existing group-based HSS. At a very high level, the group-based HSS constructions of [12] and successors work as follows. Let G be a (cyclic) cryptographically hard group[10] of prime order N, and randomly selected generator g. Let $c \in_R \mathbb{Z}_N$ be a random secret key for ElGamal encryption (as described below). Consider a message space $\mathcal{I} \subset \mathbb{Z}^+$ (where correctness error will scale with the magnitude of the inputs and partial computation values). We will maintain these notations for the remainder of the section.

Secret data is encoded by the HSS in one of two types:

- Encryptions: $\alpha \in \mathbb{Z}$ encoded by an ElGamal ciphertext $[\alpha] := (g^r, g^{cr+\alpha}) \in G^2$, for random $r \in_R \mathbb{Z}_N$. Both parties receive the ciphertext $[\alpha]$.
- Additive Shares: $\alpha' \in \mathbb{Z}$ encoded as two sets of additive secret shares $\langle \alpha' \rangle$ and $\langle c\alpha' \rangle$ over $\mathbb{Z}_N$, where $c \in \mathbb{Z}_N$ is the ElGamal secret key. (Notationally, $\langle \alpha \rangle$ denotes that each party receives a random share in $\mathbb{Z}_N$ subject to sum (over $\mathbb{Z}_N$) equaling α.)

Homomorphic evaluation takes place via a sequence of *addition* steps, performable directly on data items encoded in the same type, and *restricted multiplications*, in which a value in Encrypted type can be multiplied by a value in Additive Share type, as follows:

- Pairing: Via linear operation in the exponent using $\langle \alpha' \rangle$ and $\langle c\alpha' \rangle$ together with $[\alpha]$, the parties locally compute $g^{\langle \alpha\alpha' \rangle}$: that is, the parties hold group elements g^{β} and $g^{\beta+\alpha\alpha'}$, for some exponent $\beta \in \mathbb{Z}_N$.

  This can be viewed as a form of "distributed decryption," leveraging that decryption of a ciphertext (g^{γ}, g^{ζ}) via $(g^{\zeta}) \cdot (g^{\gamma})^{-c}$ induces an operation $\zeta - c\gamma$ in the exponent space that is linear in c. Performing the operation on an identical ciphertext (i.e., fixed γ, ζ) and additive shares of c thus yields the desired result.
- Share Conversion: Each party executes the (1D) DDL on his resulting group element.

If the DDL algorithm succeeds, then the parties result in additive shares $\langle \alpha\alpha' \rangle$ of the difference in the discrete logarithms of the two input elements $g^{\beta}, g^{\beta+\alpha\alpha'}$. In this case, namely, additive shares of the product $\alpha\alpha'$ over $\mathbb{Z}_N$.

Ultimately, the existing 1-dimensional HSS scheme performs multiplication of scalars $\zeta = \alpha\alpha' \in [M]$ and, given runtime T, succeeds except with error probability ζ/T^2 (inherited from the 1D-DDL [23]).

We next proceed to describe three solution approaches for supporting HSS for bilinear functions $\mathcal{P}_{\mathsf{blin}\text{-}k}$: (1) A non-packed baseline application of the HSS described above, where a k-dimensional plaintext vector u is simply encoded as k scalars; (2) Our new packed HSS solution *from kD-DDL*, which packs a vector u into a single ciphertext via a product of randomly selected generators, and

[10] For which finding discrete logarithms is computationally hard. More specifically, we require groups for which the "Decisional Diffie-Hellman" (DDH) assumption [22] holds: i.e., (g, g^a, g^b, g^{ab}) is computationally indistinguishable from (g, g^a, g^b, g^c) for random generator g and random exponents a, b, c.

leverages k-dimensional DDL; and (3) An alternative *1-dimensional embedding* solution approach, which encodes u into a scalar value and executes standard 1D-DDL.

Given an optimal kD-DDL (with error $d^{-2/k}$) then the kD-DDL packed HSS solution would dominate the 1-dimensional embedding approach; however, given the current gap, the two solutions are presently incomparable. In particular, the kD-DDL solution wins out when the payload magnitude and desired error probability are not fully known *in advance*.

Baseline solution: Non-packed. For baseline comparison, we consider the existing solution for obtaining group-based HSS for $\mathcal{P}_{\mathsf{blin}\text{-}k}$: *Direct (non-packed) application* of [12, 13, 23], which encodes each component of a vector $u \in [M]^k$ as a separate element

In the direct application, each vector $u = (u_1, \ldots, u_k) \in [M]^k$ is simply encoded as a collection of k ElGamal ciphertexts, and scalar-vector multiplications $\alpha \cdot u$ is homomorphically evaluated via k independent multiplications $\alpha \cdot v_i$, for $i = [k]$. The share size to encode a vector thus increases to k ElGamal ciphertexts. Each vector-scalar multiplication corresponds to k independent instances of a standard scalar-scalar multiplication; in particular, for output value $z = \alpha \cdot u$ and runtime T for the share conversion procedure (namely, 1D-DDL), the error grows as $\sum_{i=1}^{k}(z_i/T^2)$.

New solution: From kD-DDL. In our new solution *from kD-DDL*, we embed the vector u into a single ElGamal ciphertext as a corresponding *product of generators*, and run the kD-DDL algorithm to extract the corresponding additive shares of the product.

More concretely, let G be a cyclic DDH-hard group of prime order N, and let $g, g_i = g^{w_i}$, be $k+1$ randomly selected group generators. (Recall we assume N to be prime, in which case g^{w_i} for randomly chosen w_i will be a generator with high probability.) Given a secret vector $u = (u_1, \ldots, u_k) \in [M]^k$, we embed u into a single ciphertext as follows. Recall a standard ElGamal ciphertext with secret key $c \in \mathbb{Z}_N$ encodes scalar plaintext $\alpha \in [M]$ as a pair $(g^r, g^{rc} \cdot g^\alpha) \in G^2$ (where $g \in G$ is a generator and $r \in_R \mathbb{Z}_N$ is encryption randomness). We now encode a *vector* $u \in [M]^k$ into a pair of group elements using the generators g_i by sampling random $r \in_R \mathbb{Z}_N$ and outputting:

$$\mathsf{HSS.ShareVector}((u_1, \ldots, u_k); r) = \left(g^r, g^{rc} \cdot \prod_{i=1}^{k} g_i^{u_i}\right).$$

Scalar values $\alpha \in [M]$ will be encoded as Additive Secret Shares, as before.

Consider now homomorphic multiplication between an encoded *vector* and scalar.

- Pairing: Perform the standard HSS pairing procedure between the additively shared $\langle \alpha \rangle$ and $\langle c\alpha \rangle$ together with the above ciphertext. This enables the parties to obtain group elements $g^\beta \in G$ and $(g^\beta \cdot \prod_{i=1}^{k} g_i^{\alpha u_i}) \in G$ for some $\beta \in \mathbb{Z}_N$.
- Share Rerandomization: In order to rerandomize the exponent β, both parties multiply their local share by the *same* random group element g', computed as e.g. pseudorandom function of the unique instruction identifier. Note that this can be achieved with minimal additional share size (a single key to a pseudorandom function, included in each party's share) and computation. (Further, note that existence of pseudorandom functions is already implied by the DDH computational assumption.) We will thus roughly ignore this step in terms of analysis, and assume the value β is distributed (pseudo-)uniformly, conditioned on the entire execution up to this point.

- Share Conversion: At this point, the parties will now attempt to extract shares of the exponent vector $(u_1, \ldots, u_k)$ via execution of the kD-DDL algorithm. Let A be a generic (N, b, d, δ)-kD-DDLA, as per Definition 6.1. Recall for $x \in \Sigma_b^N$ and $\beta \in \mathbb{Z}_N$ we denote by $A^G(x, \beta)$ the execution of the algorithm A with oracle access to the generic group represented by x on the input challenge string $x[\beta] \in \Sigma_b$ representing the generic group string handle for the element g^β. (Note that the role of β was notated by v in the previous sections.) Then directly applying the kD-DDL property together with a union bound (see Remark below Definition 6.1), we directly have that for $\alpha \cdot u \in \mathbb{Z}_n^k$,

  $$\Pr_{x \in_R \Sigma_b^N, \beta \in_R \mathbb{Z}_N} \left[A^{\mathrm{G}}(x, \beta) - A^{\mathrm{G}}\Big(x, \beta + \sum_{i=1}^{k} \alpha u_i\Big) \neq \alpha u \right] \leq \delta \alpha |u|_1,$$

  where $|u|_1$ denotes the ℓ_1 norm of u.

  That is, aside from error probability bounded by $\delta\alpha|u|_1$, executing the algorithm A with respect to the two values g^β and $(g^\beta \cdot \prod_{i=1}^k g_i^{\alpha u_i}) \in G$ will result in precisely the desired additive output shares of the multiplied vector αu.

We now present a more detailed description of the Packed HSS construction from kD-DDL, based on the Decisional Diffie-Hellman assumption that underlies the construction.

Definition 6.5 (DDH). *Let $G = \{\mathbb{G}_\rho\}$ be a set of finite cyclic groups, where $|\mathbb{G}_\rho| = q$ and ρ ranges over an infinite index set. We use multiplicative notation for the group operation and use $g \in \mathbb{G}_\rho$ to denote a generator of $\mathbb{G}_\rho$. Assume that there exists an algorithm running in polynomial time in* $\log q$ *that computes the group operation of $\mathbb{G}_\rho$. Assume further that there exists a PPT instance generator algorithm $\mathcal{IG}$ that on input 1^λ outputs an index ρ which determines the group $\mathbb{G}_\rho$ and a generator $g \in \mathbb{G}_\rho$. We say that the Decisional Diffie-Hellman assumption (DDH) is satisfied on G if $\mathcal{IG}(1^\lambda) = (\rho, g)$ and for every non-uniform PPT algorithm $\mathcal{A}$ and every three random $a, b, c \in \{0, \ldots, q-1\}$ we have*

$$|Pr[\mathcal{A}(\rho, g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{A}(\rho, g^a, g^b, g^c) = 1]| < \varepsilon(\lambda),$$

for a negligible function ε. We will sometimes write $(\mathbb{G}, g, q) \leftarrow \mathcal{IG}(1^\lambda)$.

Notation 6.6. In this section: For $a \in \mathbb{Z}_q$, we denote by $\langle a \rangle$ additive secret shares $(a_0, a_1) \in \mathbb{Z}_q$. For selected generator g_i, we denote by $[a]_i$ the group element $(g_i)^a$. To maintain closer consistency to the notation of prior work, we will denote the (prime) group order by q (as opposed to N).

Construction 6.7 (k-Packed HSS for Vector-Scalar Mult). Let $(G, g, q) \in_R \mathcal{IG}(1^\lambda)$. Let $g_1, \ldots, g_k$ denote additional random generators of $\mathbb{G}$, selected as $g_i := g^{w_i}$ for random $w_i \in_R \mathbb{Z}_q$.

Let $A_{k\text{-}\mathsf{DDL}}$ be a query-restricted (N, b, d, δ)-kD-DDL algorithm, and $\mathsf{PRF} : \mathbb{G} \to \{0,1\}^b$ a pseudorandom function.

- $\mathsf{HSS.Share}(1^\lambda)$: Sample a random ElGamal secret key $c \in_R \mathbb{Z}_q$. To share each secret vector/scalar input, execute the corresponding algorithm:
  - $\mathsf{HSS.ShareVector}(\vec{x})$: ("Encryption") Given input vector $\vec{x} \in \{0,1\}^k$, output a *packed* ElGamal ciphertext, $\mathsf{ct} = ([r]_1, [rc]_1 \cdot \prod_{i=1}^k [x_i]_i) \in \mathbb{G} \times \mathbb{G}$.
  - $\mathsf{HSS.ShareScalar}(y)$: ("Additive shares") Given input scalar $y \in \{0,1\}$, output additive secret shares $\langle y \rangle, \langle cy \rangle$ over $\mathbb{Z}_q$. (Denote party i's part of these values by share_i.)

- $\mathsf{HSS.Eval}(i, ((\mathsf{ct}^{(1)}, \dots, \mathsf{ct}^{(\rho)}), (\mathsf{share}_i^{(1)}, \dots, \mathsf{share}_i^{(\rho')})), P, r)$: Let P denote a bilinear function

  $$P\left((\vec{x}^{(1)}, \dots, \vec{x}^{(\rho)}), (y^{(1)}, \dots, y^{(\rho')})\right) = \sum_{j \in [\rho], \ell \in [\rho']} \alpha_{j,\ell} \cdot y^{(\ell)} \cdot \vec{x}^{(j)} \in \mathbb{Z}_m^k,$$

  where $\alpha_{j,\ell} \in \mathbb{Z}_m$. Then homomorphic evaluation of P takes place as in Algorithm 9.

Algorithm 9: $\mathsf{HSS.Eval}(i, (\mathsf{ct}^{(j)}, \mathsf{share}_i^{(\ell)}), P, m)$ for bilinear functions, given $A_{k\text{-}\mathsf{DDL}}$

```
begin
    Let bilinear function $P$ be defined by coefficients $\alpha_{j,\ell} \in \mathbb{Z}_m$
    Initialize $z \in_R 1 \in \mathbb{G}$;
    for $j = 1$ to $\rho$, $\ell = 1$ to $\rho'$ do
        if $\alpha_{j,\ell} \neq 0$ then
            $z \in_R z \cdot \mathsf{Pair}(\mathsf{ct}^{(j)}, \mathsf{share}_i^{(\ell)})$;
        end
    end
    Output $A_{k\text{-}\mathsf{DDL}}(z)$;
end
Subroutine Pair(ct, share$_i$):
begin
    Parse $\mathsf{ct} = ([a], [b]) \in \mathbb{G} \times \mathbb{G}$ and $\mathsf{share}_i = (y_i, (cy)_i) \in \mathbb{Z}_q \times \mathbb{Z}_q$ ;
    Output $[b]^{y_i} \cdot [a]^{-(cy)_i} \in \mathbb{G}$;
end
Subroutine Convert($z$):
begin
    Begin executing the DDL algorithm $A_{k\text{-}\mathsf{DDL}}$
    while $A_{k\text{-}\mathsf{DDL}}$ makes query $(1, \vec{j}) \in \mathbb{Z}_n \times \mathbb{Z}_n^k$ do
        Compute $z_{\vec{j}} = z \cdot \prod_{i=1}^{k} [j_i]_i \in \mathbb{G}$;
        Respond to $A_{k\text{-}\mathsf{DDL}}$ with value $\mathsf{PRF}(z_{\vec{j}}) \in \{0,1\}^b$;
    end
    Output the value output by $A_{k\text{-}\mathsf{DDL}}$;
end
```

New solution: 1-dimensional embedding. Our second solution performs a *1-dimensional embedding*, encoding a vector $u = (u_1, \dots, u_k)$ into a single scalar element $\sum_{i=1}^{k} u_i M^{i-1}$ in an enlarged payload space $[M^k]$ for chosen parameter M (see below).

More concretely, in the 1-dimensional embedding approach, the existing (1-dimensional) HSS scheme is used in a black-box manner, embedding a vector value $u \in [M]^k$ as a single *integer* $\tilde{u} := \sum_{i=1}^{k} M^{i-1} u_i \in [M^k]$ within a larger input space, where M is a parameter chosen at the time of HSS encoding. (A larger choice of M will ultimately result in smaller multiplication error, but will require greater runtime.) The corresponding share size of the vector u is thus a *single* ElGamal ciphertext. Homomorphic scalar-vector multiplication of $z = \alpha \cdot u$ will take place via two phases:

- Applying the 1D HSS multiplication procedure on the encoded secret *integer* $\tilde{u} \in [M^k]$ (encoded via Encryption) together with secret scalar $\alpha \in [M]$ (encoded via Additive Shares), then with error probability $\tilde{z}/T^2$, one can homomorphically obtain additive secret shares of the *integer* product $\tilde{z} := \alpha\tilde{u} = \sum_{i=1}^{k} M^{i-1}\alpha u_i$ over $\mathbb{Z}_N$.

  However, this is not the required output: HSS demands additive shares of the target vector $z = \alpha u$, over the corresponding *vector space* $\mathbb{Z}_N^k$. This distinction is a crucial requirement for many HSS application settings, where additive shares are later combined or manipulated over the respective output space.[11]

- To satisfy this requirement, the 1-dimensional embedding approach must thus add a step to revert shares of $\tilde{z} = \sum_{i=1}^{k} M^{i-1}z_i$ over $\mathbb{Z}_N$ to shares of $z = (z_1, \ldots, z_k)$ over $\mathbb{Z}_N^k$.

  Denote the original shares by $a, a' \in \mathbb{Z}_N$, and express as integers in base M: i.e., $a_0, a_1, \ldots, a_m$ and $a'_0, a'_1, \ldots, a'_m$ where $a_i, a'_i \in \{0, \ldots, M-1\}$ and $m = \lceil \log_M(N) \rceil$. Conditioned on correct shares of z over $\mathbb{Z}_N$, we have $a' = a + \sum_{i=1}^{k} M^{i-1}z_i$, which in turn implies each coordinate $a'_i = a_i + z_i$ as long as for each i it holds that $(a_i + z_i) < M$ (i.e., as long as there is no "carry" to the next power of M). Since $z_i \in [M]$, then by rerandomizing shares (i.e., adjusting both shares a, a' by the same random additive offset in $\mathbb{Z}_N$), this bad event $(a_i + z_i) \geq M$ occurs only if $a_i \geq M - 1 - z_i$, i.e. with probability z_i/M.

Combining the error of the HSS together with this error from share conversion (union bounding over the k dimensions), the overall error of the 1-dimensional embedding approach becomes $(\sum M^{i-1}z_i/T^2) + (\sum z_i/M)$, where $z = \alpha u \in [M]^k$ and summations are over $i = 1$ to k.

Remark (1D embedding parameter M). Observe that the additive term $(\sum z_i/M)$ in the 1D embedding multiplication error expression does not decrease with runtime T. The choice of the parameter M must thus be set sufficiently large to enable a desired error $(\sum z_i/M) < \epsilon$. However, M must be selected at the time of HSS *encoding*—to translate the plaintext vector u to an integer $\sum M^{i-1}u_i$—at which point the eventual payload magnitude $\sum z_i$ and target error ϵ may not yet be known.

The 1D embedding solution thus requires one to *predict* at encode time what the final payload magnitude and target error will eventually be. If the prediction mis-estimates the final expression $(\sum z_i/\epsilon)$ by a multiplicative factor ξ, then one of two things will take place. If $\xi < 1$, i.e. M was chosen too small, then the execution will have failed: error ϵ will be unachievable. If $\xi > 1$, i.e. M was overestimated, then this will inflict a runtime overhead to appropriately shrink the second error term $(\sum M^{i-1}z_i/T^2)$. Specifically, to obtain a given error with this inflated M, one will now need to increase T by a factor of $\xi^{(k-1)/2}$ to account for the extra leading M^{k-1} term in the numerator.

Comparison of approaches. Ultimately, the resulting parameters of the three approaches are summarized in the following theorem statement.

Theorem 6.8 (Packed HSS from kD-DDL)**.** *Suppose there exists a query-restricted (N, b, d, δ)-k-dimensional-DDL algorithm. Then, based on the DDH assumption, there exists HSS for the class $\mathcal{P}_{\mathsf{blin}\text{-}k}$ for vector-scalar operations with the following parameters. Multiplication error is given for vector-scalar multiplication $z = \alpha u$; summations are $\sum_{i=1}^{k}$.*

[11] For example, HSS for program class $\mathcal{P}$ yields succinct 2-server Private Information Retrieval (PIR) [20] for private database queries of related class $\mathcal{P}'$ [30, 11], crucially depending on the *additive* reconstruction of the HSS scheme over the output space of $\mathcal{P}$.

	Non-packed	*1D Embedding* (M)	*From kD-DDL*
Share size, vector $u \in [M]^k$*:*	k *ElGamal CT*	*1 ElGamal CT*	*1 ElGamal CT*
Share size, scalar $\alpha \in [M]$*:*	*2* $\mathbb{Z}_N$*-elmts*	*2* $\mathbb{Z}_N$*-elmts*	*2* $\mathbb{Z}_N$*-elmts*
Mult error, $z = \alpha u$*, time* T*:*	$\sum z_i/T^2$	$\sum(M^{i-1}z_i)/T^2 + (\sum z_i)/M$	$\sum z_i \cdot \delta(T)$

Note that the *non-packed* application of [12, 13, 23] has large vector share size. As discussed above, if the desired final error probability and the magnitude of the final payload z are known, then the error expression for the *1D embedding solution* can be minimized to $\sim \sum z_i/T^{2/k}$, by setting the encoding parameter M to $M = T^{2/k}$.

The *solution from kD-DDL* does not require this a priori knowledge. Plugging in the results from Section 4, for example, we have a provable 2D-DDL algorithm that makes T queries (that can be implemented using $\tilde{O}(T)$ group multiplications) and results in 2-dimensional packed HSS with vector-scalar multiplication error $\sum z_i\delta(T) \sim \sum z_i T^{-7/8}$. Using the conjectured optimal algorithm, the $T^{-7/8}$ term can be replaced by T^{-1}.

6.2 Location-Sensitive Encryption

In this section, we present an application of LPHS techniques to a form of *location-sensitive encryption* (LSE). At a high level, an LSE scheme enables a user to encrypt messages with respect to his location (e.g., within a virtual world), as captured by a substring $x \in \{0,1\}^n$ representing the user's view within a global environment (modeled as a much larger binary string). The LSE scheme should support two properties: (1) that any other user within close proximity to the location of encryption can decrypt, and (2) that any user who is far from this location does not learn any information about the hidden plaintext. We focus on the case of a 1-dimensional such string for simplicity, although an analogous approach can be take for 2 dimensions. Proximity in this setting corresponds to shifted view strings $x \in \{0,1\}^n$, $(x \lll i) \in \{0,1\}^n$.

Importantly, we seek LSE schemes whose efficiency requirements are *sublinear* in the (potentially large) view size n. In particular, this rules out approaches based on existing constructions of "fuzzy extractors" [24] (loosely, extractors robust to small input noise) with respect to edit distance. While such object would imply LSE, the constructions operate via a combination of tiling and hashing for set similarity, which require linear time.

However, the constructions in this section take inspiration from [24], and can informally be viewed as constructing a *sublinear-time* fuzzy extractor for shift distance.

We remark that the above-described goal of location-sensitive encryption is inherently different from "position-based encryption" as in [18], where parties rely on geographic location at the time of communication and leverage the speed of light to ensure a single recipient.

Conditions on global environment. In order to provide the desired LSE guarantees, it must of course hold that the global environment string is neither too regular nor predictable.

Regularity: We follow the terminology as introduced in the worst-case LPHS section, in Section 5. Recall that $\mathsf{Good}_n^{\alpha,W}$, parameterized by "window size" $W \in [n]$ and difference parameter $0 < \alpha \leq 1$, denotes the set of strings x in $\{0,1\}^n$ such that each of the length-W substrings of x differ pairwise in at least α fraction of their symbols (see Definition 5.8). For purposes of concreteness and streamlined notation, we will focus on inputs in $\mathsf{Good}_n^{\alpha,W}$ for specific fixed parameters:

Notation 6.9. Within this section, we will denote $\mathsf{Good}_n^{\alpha,W}$ for $\alpha = 0.1$ and $W = 0.1n$ by the abbreviated notation Good_n.

Predictability: In order to provide the desired secrecy property for parties whose view strings are not overlapping the encryptor's, it must necessarily be that un-viewed portions of the global environment string are unpredictable. A natural approach would be to place some entropy requirement on the global string; however, leveraging such worst-case entropy while maintaining sublinear complexity would necessitate heavy requirements on the amount of such entropy available. We instead consider the following notion of *local unpredictability*, which in particular holds for sources with constant-fraction entropy, but further holds for most distributions whose entropy is only polylogarithmic, including many occurring naturally within applications. Roughly, a source X over $\{0,1\}^n$ is locally unpredictable if one cannot predict the symbols x_{i+S} of a sample $x \in_R X$ at any i-shift of random challenge index set $S \subset [n]$, except with negligible probability.

Definition 6.10 (Local Unpredictability)**.** *We will say a distribution X on $\{0,1\}^n$ is* (m,ϵ)-locally unpredictable *with respect to window size W if for every algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ winning in the following challenge is $\epsilon(n)$.*

1. *The challenger samples $x \in_R X$ and a random $m(n)$-size subset of coordinates $S \in_R \binom{[W]}{m}$. It sends S to $\mathcal{A}$.*

2. *The algorithm $\mathcal{A}$ must output a pair $(i, x'_S) \in [n] \times \{0,1\}^m$. It wins if $x'_{i+S} = x_{i+S}$, where $i + S := \{i + s \mod n : s \in S\}$.*

For purposes of this section, if we say X is simply "locally unpredictable," this will implicitly refer to a convenient case where $m(n) = \log^4 n$ and $\epsilon(n) = 2^{-\log^3 n}$, for window size $W = 0.1n$.

For a string $x \in \{0,1\}^n$ and $\ell \in [n]$, recall the notation $x \lll \ell$ denotes a randomized process which samples a random suffix $x'_2 \in_R \{0,1\}^\ell$ and outputs the n-bit string formed by the last $(n-\ell)$ bits of x appended with the ℓ bits x'_2. We will use the notation $x' \in x \lll \ell$ to denote that $\Pr[x' = x \lll \ell] > 0$; equivalently, $x' \in \mathsf{Supp}(x \lll \ell)$.

We now define the Locality-Sensitive Encryption notion that is the focus of this section.

Definition 6.11 (Location-Sensitive Encryption)**.** *An α-*location-sensitive encryption *(LSE) scheme for message space $\mathcal{M}$ is a pair of PPT algorithms* $(\mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- $\mathsf{Enc}(m, x)$ *takes as input message $m \in \mathcal{M}$ and location string $x \in \{0,1\}^n$, and outputs a ciphertext c.*

- $\mathsf{Dec}(c, x')$ *takes as input ciphertext c and location string $x' \in \{0,1\}^n$, and outputs a plaintext value $m' \in \mathcal{M}$.*

The scheme is said to be a sublinear LSE *if* Gen *and* Enc *make oracle access into the bits of their respective inputs x, x', and the number of such queries made is $o(n)$.*

An LSE scheme must satisfy the following correctness and security properties:

- **Nearby decryption.** *For every message $m \in \mathcal{M}$, location string $x \in \mathsf{Good}_n$, and $x' \in x \lll i$ for some $i \leq \alpha n$, it holds that*

$$\Pr_{c \in_R \mathsf{Enc}(m,x)} \left[\mathsf{Dec}(c, x') = m\right] = 1 - \mathsf{negl}(n).$$

- **Far-distance security (from local unpredictability).** *Let X be any distribution on Good_n which is locally unpredictable (Definition 6.10). Then it holds for X and for any $m, m' \in \mathcal{M}$ that encryptions of m and m' are computationally indistinguishable:*

$$\{\mathsf{Enc}(m, X)\} \overset{c}{\cong} \{\mathsf{Enc}(m', X)\}.$$

6.2.1 Building Sublinear LSE

In what follows, we demonstrate how to achieve a *sublinear* location-sensitive encryption scheme, taking inspiration from notions of fuzzy extractors and biometric embeddings of Dodis *et al.* [24]. The high-level idea will be to build a tool comparable to a fuzzy extractor for *shifts*. This is done by constructing a form of metric embedding from shift into Hamming distance, which then enables us to directly appeal to fuzzy extractor results for Hamming metric.[12] The construction of such embedding is the focus of this subsection.

We begin by constructing a weaker tool—a Binary LPHS—and then amplify. Loosely, a Binary LPHS family maps inputs to a single bit, such that close inputs with respect to shift distance are mapped to the same bit with good probability, whereas inputs with sufficient unpredictability hash unpredictably.

The exposition is organized as follows. We first present the definition of shift-to-Hamming embedding. We present and construct a notion of Binary LPHS. We then provide an amplification procedure which attains a shift-to-Hamming embedding given access to Binary LPHS. Finally, we demonstrate how to pair this embedding together with techniques from [24] to reach the desired LSE primitive.

For $x, x' \in \{0,1\}^n$, we denote Hamming distance of x and x' by $\Delta(x, x')$.

Definition 6.12 (Shift-to-Hamming Embedding)**.** *We define a (α, α')-randomized* shift-to-Hamming embedding *with output length $\ell = \ell(n)$ as a pair of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Embed})$ with the following syntax.*

- $\mathsf{Gen}(1^n)$ *is a randomized procedure that takes input length 1^n in unary and outputs an index value v.*
- $\mathsf{Embed}(v, x)$ *is a deterministic procedure that takes as input an index v and input string $x \in \{0,1\}^n$, and outputs a value $y \in \{0,1\}^\ell$.*

  We will consider sublinear embeddings, wherein Embed makes oracle access into the bits of x, and the number of such queries made is $o(n)$.

The algorithms satisfy the following properties.

- **Preserves closeness.** *For any $x \in \mathsf{Good}_n$ and $x' \in \{0,1\}^n$ for which $x' \in x \lll i$ for some $i \leq \alpha n$, it holds that*

$$\Pr_{v \in_R \mathsf{Gen}(1^n)}[\Delta\big(\mathsf{Embed}(v, x), \mathsf{Embed}(v, x')\big) \leq \alpha' \ell] \geq 1 - \mathsf{negl}(n).$$

- **Preserves unpredictability.** *Let X be any distribution on Good_n which is locally unpredictable (Definition 6.10). Then for every algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ winning in the following challenge is negligible in n.*

  1. *The challenger samples $v \in_R \mathsf{Gen}(1^n)$ and $x \in_R X$. It sends v to $\mathcal{A}$.*
  2. *The algorithm $\mathcal{A}$ must output $y' \in \{0,1\}^\ell$. It wins if $y' = \mathsf{Embed}(v, x)$.*

[12] We remark that a sublinear method for embedding edit distance into Hamming distance was shown in a recent independent work [34], also using a random walk technique.

Note that the "preserves unpredictability" property guarantees that the output string (on a locally unpredictable output) has super-logarithmic minimum entropy. Once such embedding is reached, this can be combined with constructions of (computational) fuzzy extractors for Hamming distance, to yield the desired LSE encryption scheme. While this entropy level is not sufficient for the existence of information theoretically secure fuzzy extractors for Hamming distance [28], computational constructions exist based on plausible computational hardness assumptions (see e.g. [27, 32, 16, 2, 42]). The resulting LSE security inherits the corresponding agreement/entropy parameters and computational assumption of the underlying fuzzy extractor.

We will achieve the desired shift-to-Hamming embedding by means of the following tool, a form of Binary LPHS.[13] As described earlier, a Binary LPHS family maps inputs to a single bit, such that close inputs map to the same bit with good probability, whereas inputs with sufficient unpredictability hash in an unpredictable manner.

Definition 6.13 (Binary LPHS)**.** *A family of hash functions $\mathcal{H} = \{h : \{0,1\}^* \to \{0,1\}\}$ is said to be a (α, β)-*binary LPHS *if the following properties hold.*

- **Close inputs agree.** *For any $x \in \mathsf{Good}_n$ and $x' \in \{0,1\}^n$ for which $x' \in x \lll i$ for some $i \leq \alpha n$, it holds*
$$\Pr_{h \in_R \mathcal{H}}[h(x) = h(x')] \geq 1 - \beta.$$
- **Output unpredictability for locally unpredictable inputs.** *Let X be any distribution on Good_n which is locally unpredictable (Definition 6.10). Then for every algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ winning in the following challenge is bounded by $1/2 + \mathsf{negl}(n)$.*
    1. *The challenger samples $h \in_R \mathcal{H}$ and $x \in_R X$. It sends h to $\mathcal{A}$.*
    2. *The algorithm $\mathcal{A}$ must output $y \in \{0,1\}$. It wins if $y = h(x)$.*

We next proceed to build the above notion of Binary LPHS. The approach borrows techniques from worst-case LPHS: first reducing to the random-input case via a "random subset tiling" of the input x (to obtain a related string over a larger alphabet with all distinct symbols) followed by a t-wise independent hash applied to individual symbols. As a second step (similar to worst-case LPHS) we can now apply an average-case LPHS on the resulting string y. However, instead of simply outputting the resulting LPHS-output integer z (or compressed version), which has limited unpredictability, we instead use this value to *select a symbol* from the intermediate randomized string y, and extract out a single bit output.

Loosely, given inputs x, x' differing by small shift i, the resulting strings y, y' will differ by shift of i, and the LPHS will (aside from some error) provide outputs z and $z' = z - i$ differing by i. Thus the two outputs will select the *same* symbol, $y_z = y'_{z'}$. On the other hand, for any x with sufficient unpredictability, then no single symbol of the corresponding string y will be predictable, thus the resulting bit will be close to uniform.

Proposition 6.14 (Constructing Binary LPHS)**.** *There exist $0 < \alpha < \beta < 1$ for which there exists (α, β)-binary LPHS making $\tilde{O}(\sqrt{n})$ queries to the input x.*

Proof. We provide the desired binary LPHS construction. Consider the following tools (each with $\tilde{O}(\sqrt{n})$ query complexity into the input x, when relevant):

[13] Note that this notion is highly related but not equivalent to the Binary LSH primitive defined and constructed in Theorem 6.19.

- A hash function family $\mathcal{H} = \{h_{S,\gamma}\}$, indexed by a subset $S \subset [n]$ of size $\log^2 n$, and a hash function $\gamma : \{0,1\}^{\log^2 n} \to \{0,1\}^{\log^2 n}$ from an n-wise independent hash family. Sampling a hash function $h_{S,\gamma}$ from $\mathcal{H}$ will consist of randomly selecting $S \subset [n]$ and sampling γ from the n-wise independent hash family. The output of $h_{S,\gamma}(x)$ is the string $(y_i)_{i\in[n]} \in (\{0,1\}^{\log^2 n})^n$ for which $y_i = \gamma(x_{i+S})$.
- An (n, d, δ)-non-cyclic *average-case* LPHS h^* with shift bound αn, for random inputs in $(\{0,1\}^{\log^2 n})^n$, for $d \in \tilde{O}(\sqrt{n})$ and $\delta \in \tilde{\Theta}(1/n)$.
- A seeded extractor $\mathsf{Ext} : \{0,1\}^{\log^2 n} \times \{0,1\}^{\log^2 n} \to \{0,1\}$ for entropy sources X over $\{0,1\}^{\log^2 n}$ with $H_\infty(X) \geq \log^{3/2} n$. Namely, for any such X, it holds $(U, \mathsf{Ext}(U, X)) \overset{s}{\cong} (U, U_{\{0,1\}})$.

We observe that the following combinations of these tools are proved to satisfy various properties in other sections of this work:

- $h_{S,\gamma} : \{0,1\}^n \to (\{0,1\}^{\log^2 n})^n$ for randomly sampled S, γ is proved in Appendix 5 to convert a worst-case input $x \in \mathsf{Good}_n$ to a *random* input $h_{S,\gamma}(x) \in (\{0,1\}^{\log^2 n})^n$, while preserving shift distance between close input pairs.
- The composition $h_{\mathsf{wc}} := h^* \circ h_{S,\gamma} : \{0,1\}^n \to \mathbb{Z}$ for randomly sampled S, γ is proved in Appendix 5 to be a *worst-case LPHS* for inputs $x \in \mathsf{Good}_n$.

We construct the desired Binary LPHS hash function family $\mathcal{H}$ as follows.

- **Sampling.** A hash function $h \in_R \mathcal{H}$ is sampled from the family by sampling S, γ as above, and sampling a random extractor seed $r \in_R \{0,1\}^{\log^2 n}$. The hash function is indexed by (S, γ, r)
- **Evaluating.** To evaluate $h = h_{(S,\gamma,r)}(x)$:
  1. Compute $y = h_{S,\gamma}(x) \in (\{0,1\}^{\log^2 n})^n$.
  2. Compute $z = h^*(y) \in [n]$. (This corresponds to $z = h_{\mathsf{wc}}(x)$.)
  3. Output $\mathsf{Ext}(r, y_z) \in \{0,1\}$.

We now prove that $\mathcal{H}$ satisfies the necessary properties.

Close inputs agree. Let $x \in \mathsf{Good}_n$ and $x' \in \{0,1\}^n$ for which $x' \in x \lll i$ for $i \leq \alpha n$. Consider the probability space defined by sampling (S, γ, r) via $h \in_R \mathcal{H}$. Denote $y = h_{S,\gamma}(x), y' = h_{S,\gamma}(x'), z = h^*(y)$, and $z' = h^*(y')$. Define the event E to occur when the following two conditions hold:

1. $y' \in y \lll i$: that is, the tiling of x and x' properly preserves their shift.
2. $z = z' + i$: that is, correctness holds for the worst-case LPHS h_{wc} applied to x, x'.

It is proved in Section 5 (as part of worst-case LPHS Proposition 5.4), together with a union bound on i, that event E fails to occur with probability no greater than $i\delta$. Conditioned on event E, then (since z is sufficiently small) it holds that $y'_{z'} = (y \lll i)_{z-i} = y_z$; in particular, $\mathsf{Ext}(r, y_z) = \mathsf{Ext}(r, y'_{z'})$.

Output unpredictability for locally unpredictable inputs. Let X be a locally unpredictable distribution on $\{0,1\}^n$. Consider the probability space defined by sampling (S,γ,r) via $h \in_R \mathcal{H}$ and $x \in_R X$. Consider a hash $h(x)$-predictor algorithm $\mathcal{A}$ who is given both the hash index (S,γ,r) (as is the case in the security game), as well as the value $z = h_{\mathsf{wc}}(x) \in [n]$ as additional leakage on x. By the local unpredictability of X, given just S, no algorithm can predict x_{i+S} for any shift $i \in [n]$ with better than negligible probability $\epsilon = 2^{-\log^3 n}$. The values γ, r are independent and thus do not affect this probability. The leakage $z = h_{\mathsf{wc}}(x) \in [n]$ provides at most $\log n$ bits of information and hence cannot improve the prediction ability beyond a factor of $2^{\log n}$. In particular, $\mathcal{A}$ cannot predict $y_z = x_{z+S}$ with probability better than negligible $2^{\log^2 n}$; that is, $H_\infty(y_z|S,\gamma,r,z) \in \Omega(\log^2 n)$. Thus, it holds that $((S,\gamma,r),z,\mathsf{Ext}(r,y_z)) \overset{s}{\cong} ((S,\gamma,r),z,U_{\{0,1\}})$. The claim follows.

$\blacksquare$

We next demonstrate how to attain a randomized shift-to-Hamming embedding, making use of the Binary LPHS. Simply, the embedding map will be generated by sampling polylogarithmically many independent Binary LPHS hash function descriptions, and the embedding of an input $x \in \mathsf{Good}_n$ is performed by evaluating each hash function on x. At a high level, for any two inputs x, x' close in shift distance, a large fraction of the Binary LPHS evaluations will agree, resulting in close Hamming distance of the respective outputs; in contrast, any input distribution X with local unpredictability will introduce entropy into several of the hash output values.

The formal unpredictability argument is slightly more subtle, as the hash functions (while independent) are each applied to the *same* input sample. Thus hash outputs $h_1(x),...,h_j(x)$ can be viewed as side information on x that can aid in predicting $h_{j+1}(x)$. However, as we prove, this leaked information still leaves sufficient local unpredictability in the sample x, thus allowing us to appeal to the existing entropy argument.

Proposition 6.15. *Let $\mathcal{H}$ be a (α,β)-binary LPHS, as per Definition 6.13. Then the following procedure is a (α,β') random shift-to-Hamming embedding, as per Definition 6.12, for any $\beta' > \beta$, with output length $\ell(n) = \log^2 n$.*

- $\mathsf{Gen}(1^n)$*: Sample $\ell = \log^2 n$ independent hash functions $h_1,\dots,h_\ell \in_R \mathcal{H}$.*
  Output $v = (h_1,\dots,h_\ell)$.
- $\mathsf{Embed}(v,x)$*: Parse $v = (h_1,\dots,h_\ell)$. Output $(h_1(x),\dots,h_\ell(x)) \in \{0,1\}^\ell$.*

Proof. Consider the necessary properties.

Preserves closeness: Let $x \in \mathsf{Good}_n$ and $x' \in x \lll i$ for $i \le \alpha n$. Then it holds that

$$\Pr_{v \in_R \mathsf{Gen}(1^n)}[\Delta(\mathsf{Embed}(v,x),\mathsf{Embed}(v,x')) \le \beta'\ell] = \Pr_{h_j \in_R \mathcal{H}, j\in[\ell]}\left[\sum_{j\in[\ell]} \left|h_j(x) - h_j(x')\right| \le \beta'\ell\right]$$

Now, recall for any $x \in \mathsf{Good}_n$ that each h_j *independently* satisfies $h_j(x) = h_j(x')$ with probability at least $1-\beta$. Each $|h_j(x) - h_j(x')|$ can then be analyzed as an independent Bernoulli boolean variable equal to 1 with probability no greater than β. The probability expression above is thus bounded by $e^{-\Omega(\ell^2)}$ by a Chernoff bound. Since $\ell = \log^2 n$ this is negligible in n, as desired.

Preserves unpredictability: Let X be a locally unpredictable distribution over $\{0,1\}^n$. We wish to show that no algorithm $\mathcal{A}$ can predict the output $\mathsf{Embed}(v,x)$ (for random $v \in_R \mathsf{Gen}(1^n), x \in_R X$) given the hash index v, except with negligible advantage. Fix a strategy $\mathcal{A}$ for X. Let

$$p_j := \Pr_{\substack{h_1,\dots,h_j \in_R \mathcal{H},\\ x \in_R X}}\left[\mathcal{A}_\ell\big(v, h_1(x),\dots,h_{j-1}(x)\big) = h_j(x)\right]$$

Then

$$\Pr_{\substack{v \in_R \mathsf{Gen}(1^n),\\ x \in_R X}}[\mathcal{A}(v) = \mathsf{Embed}(v, x)] = \Pr_{\substack{h_j \in_R \mathcal{H},\\ x \in_R X}}[\forall j \in [\ell], \mathcal{A}_j(v) = h_j(x)] \leq \prod_{j \in [\ell]} p_j.$$

Consider a single p_j. We know the input distribution X is locally unpredictable, where no algorithm can win the game of Definition 6.10 with probability better than $2^{-\log^3 n}$. In the expression for p_j, an algorithm is given leakage on x, in the form of the previous hash function evaluations. In the worst case, there are $\ell = \log^2 n$ such bits of leakage on x. However, such leakage can at best improve the local unpredictability success probability to $(2^{-\log^3 n})(2^{\log^2 n}) < 2^{-\Omega(\log^3 n)}$. That is, even the distribution of X conditioned on the leakage satisfies local unpredictability, the output unpredictability property of the Binary LPHS will apply, implying that p_j is bounded above by $1/2 + \mathsf{negl}(n)$. Thus, the desired probability above is bounded by $\prod_{j \in [\ell]}(1/2 + \mathsf{negl}) \in 2^{-\Omega(\log^2 n)}$, which is negligible, as required.

■

6.3 Algorithmic Applications

In this section we discuss two representative algorithmic applications of LPHS. These applications are generic in nature and could apply to any kind of LPH (see, e.g., [33] for other examples). However, we have tried to identify the simplest settings and parameter regimes that benefit from the advantages of LPHS over alternative approaches. In all of these applications the goal is to identify in *sublinear time*, and with low failure probability, either *small* or *arbitrary* misalignments of two or more strings.

The first application only takes advantage of the short output length of the LPHS, whereas the second take advantage of the metric property of being "locality preserving." Finally, while we describe the applications in the 1-dimensional, random-input case, they can naturally benefit from the k-dimensional, worst-case-input LPHS variants considered in this paper. In fact, 2-dimensional LPHS seems like the most useful variant in these contexts.

6.3.1 Succinct sublinear-time sketching for shifts

Consider the following sketching scenario, described as a simultaneous messages (SM) communication complexity problem. Two parties $\mathcal{A}$ and $\mathcal{B}$, with shared randomness ρ, hold n-bit substrings x_A and x_B of a big string $X \in \{0,1\}^N$. The two inputs are within (non-cyclic) shift offset $s \in [-R, R]$ of each other, for some shift bound $R < n$. Each party can send c bits to Carol, where the message is computed by (adaptively) reading d bits from the input. Carol should reconstruct s from the two messages with error probability bound γ, assuming that X is picked uniformly at random and independently of ρ. What are the achievable tradeoffs between the parameters?

Claim 6.16. *There exists a non-cyclic shift-finding SM protocol with $c = \log R + O(1)$ bits of communication, $d = O(\sqrt{n})$ input queries, and $\gamma = \tilde{O}(R/n)$ error probability.*

Proof. Let $h_\rho : \{0,1\}^n \to \mathbb{Z}$ be a (near-optimal) non-cyclic (d, δ)-LPHS with $d = O(\sqrt{n})$ and $\delta = \tilde{O}(1/n)$, as guaranteed by Theorem 3.5. Each party sends to Carol the output of h on its input, reduced modulo $2R+1$. Carol computes the difference δ between the two messages modulo $2R+1$, and outputs either $s = \delta$ if $0 \leq \delta \leq R$ or $s = \delta - (2R+1)$ otherwise. Using Lemma 2.4, the error probability is at most $|s| \cdot \delta \leq \tilde{O}(R/n)$ as required. ■

In particular, if $R = \mathrm{polylog}(n)$, we get $\tilde{O}(1/n)$ error using $\log R + O(1)$ bits of communication (and with only $O(\sqrt{n})$ queries). Note that within the tight communication budget of $c = \log R + O(1)$, we cannot afford to amplify the success probability of a protocol based on the simple $(d, O(1/d))$-LPHS via repetition. Finally, the repetition-based approach cannot yield sublinear-time protocols with low error probability δ when R is big, as required by the extension to unbounded shifts discussed below.

One can get a Las Vegas variant of the above sketching protocol, where Carol can *detect* whenever an error may occur (except with negligible probability), using the Las Vegas flavor of LPHS (see Definition 2.2 and following remark). Other LPHS variants can also be motivated in this setting. The assumption that X is random can be relaxed to "far from periodic" by using the notion of worst-case LPHS from Appendix 5. A 2D-LPHS can be used to capture a 2-dimensional terrain X.

Sketching for unbounded shifts. The above protocol can be extended to apply to an *arbitrary* shift amount, with $\mathrm{polylog}(n)$ communication, $d = \tilde{O}(n^{1/2})$ input queries, and $n^{-\omega(1)}$ failure probability, by running multiple instances of the protocol with $R = n/\mathrm{polylog}(n)$, where in each instance $\mathcal{B}$ shifts its inputs by a different multiple of R. Using a Las Vegas variant of LPHS, Carol can identify the correct instance. Alternatively, one can avoid using a Las Vegas variant and rely instead on Lemma 2.9 for identifying the distractors. Note that, unlike the case of small R discussed above, here we cannot use at all the simple $(d, O(1/d))$-LPHS. Indeed, with $d = \tilde{O}(n^{1/2})$, the failure probability is too big to handle large shifts.

6.3.2 Locality-sensitive hashing and near-neighbor data structures for shifts

A near-neighbor data structure represents m points in a metric space and enables efficient, e.g. sublinear time, near-neighbor queries on any point on the space. A near-neighbor query on point x with distance R returns a point y in the data structure that is within distance R from x or returns an indication of failure if no such point exists. Approximate near-neighbor queries relax the requirement so that with good probability the output y is within distance cR from x for some constant $c > 1$.

In this section we design an approximate near-neighbor data structure for strings with distance measured by a shift metric, with applications to matching shifted pictures, or determining the location of an agent in some terrain given only a local view of its surroundings.

Intuitively, the term "shift distance" refers to the minimal shift amount required to obtain one string from the other. In the non-cyclic case this should correspond to directed distance in the De Bruijn graph, whereas in the cyclic case the distance is infinite if there is no such shift. Our notion of shift distance should not be confused with an alternative notion (cf. [4]) referring to the smallest Hamming distance between one string and some cyclic shift of the other.

However, defining a shift distance correctly requires some care for non-cyclic shifts. Measuring the number of shifts required to obtain an n-bit string y from an n-bit string x in the directed De Bruijn graph is not symmetric. Using the graph metric on the undirected De Bruijn graph leads to cases in which strings that should be distant in our proposed applications are near in the graph. In our proposed applications of this metric, the shifted strings are part of a larger "universe". We therefore adopt the following metric, defined over a sub-graph of the De Bruijn graph.

Definition 6.17 (Shift metric). *Let n, N be two integers $n \leq N$ and let C be a cycle of length N in the undirected De Bruijn graph over strings of length n. The* shift metric *over C, denoted by*

d_C, is the standard graph metric on $V(C)$, the N nodes in C.

An equivalent way to view his definition is to regard C as a circular string of length N such that any substring of length n appears exactly once in C. The set of all N substrings of length n in C is denoted $V(C)$. The distance $d_S(x, y)$ between two substrings $x, y \in V(C)$ is the minimum of two values: the number of shifts on C required to move from x to y and the number of shifts on C required to move from y to x.

A useful tool in the design of near-neighbor data structures is *Locality Sensitive Hashing* (LSH) which assigns to any two points x, y the same value with high probability if they are close and two different values if they are distant. More precisely,

Definition 6.18 (LSH)**.** *Let M be a set with a metric d. A family of hash functions $\mathcal{H}$ is a (R, cR, p_1, p_2)* Locality-Sensitive Hash (LSH) *for (M, d) if for any two points $x, y \in M$*

- *If $d(x, y) \leq R$ then $\Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] \geq p_1$.*
- *If $d(x, y) \geq cR$ then $\Pr_{h \in_R \mathcal{H}}[h(x) = h(y)] \leq p_2$.*

Constructing a near-neighbor data structure from an LSH (see [6] and references therein) uses a preprocessing step in which L hash functions $h_1, \ldots, h_L$ are randomly chosen from $\mathcal{H}$ and L hash tables are constructed. Each point y is then placed in L buckets $h_1(y), \ldots, h_L(y)$. Given a query point x the distances between the points in all the buckets $h_1(x), \ldots, h_L(x)$ and x are computed. If there exists some point y in one of these buckets such that $d(x, y) \leq cR$ then y is returned and otherwise failure is announced.

We construct an LSH family for the shift metric d_C on a large cycle C in the De Bruijn graph on n-bit strings. Similarly to the case of Location Sensitive Encryption the input string cannot be too regular. We begin by showing an LSH family for random input and then discuss how to achieve similar results to the case of worst-case inputs which are α-Good in the sense of Appendix 5.1.

Theorem 6.19. *Let Σ be an alphabet, $|\Sigma| \geq n^3$, let n, N be integers, $N > n$, and let C be a random N-bit binary string. For any constant $c > 1$ there exist a constant a and a hash family $\mathcal{H}$ of functions $h : \Sigma^n \to \{0, 1\}$ that is (R, cR, p_1, p_2)-LSH for the cyclic shift metric over C, with parameters $0 \leq R \leq \frac{n}{2ac}$, $p_1 = 1 - \frac{R(2a+1)+1}{n}$, and $p_2 = p_1 - \frac{R((2c-2)a-c-1)}{n})$. In addition, any $h \in \mathcal{H}$ can be computed with $\sqrt{n}$ queries $x[i]$ to the input string x, and the same result holds for any alphabet Σ with a that is polylogarithmic in n.*

Proof. Let $\mathcal{H}'$ denote the LPHS family constructed in Theorem 3.5 that with $\sqrt{n}$ queries of an input string of length n achieves error probability a/n, such that a is a constant for $|\Sigma| \geq n^3$, and is polylogarithmic in n for general alphabet. Let $\mathcal{H}$ be the family of hash functions $h_u : V(C) \to \{0, 1\}$, for $u = (h', z)$, $h' \in \mathcal{H}'$, $z \in \{0, \ldots, n-1\}$ defined by

$$h_u(x) = \begin{cases} 0 & (h'(x) \bmod n) < z \\ 1 & \text{otherwise} \end{cases}$$

If $x, y \in V(C)$ and $d_C(x, y) = 1$ then $\Pr_{h' \in \mathcal{H}'}[h'(x) = h'(y) + 1] = 1 - a/n$. Therefore, by union bound, if the distance between x and y is $r \leq R$ then $\Pr_{h' \in \mathcal{H}'}[h'(x) = h'(y) + r] = 1 - ar/n$. If $h'(x) = h'(y) + r$ then $h_u(x) \neq h_u(y)$ if z is chosen so that it is between $h'(x) \bmod n$ and $h'(y) \bmod n$. There are two possible cases for $h'(x) = h'(y) + r$: either $h'(x) \bmod n > h'(y) \bmod n$

over the integers or $h'(x) > n-r$. In the first case we have that $\Pr_{h_u \in \mathcal{H}}[h_u(x) = h_u(y)] = 1 - r/n$. While for the second case it follows from the proof to Lemma 2.9 (plugging in $m = n$ and $\delta = a/n$) that $\Pr_{x \in \Sigma^n}[(h'(x) \bmod n > n - r] \leq 1/n + ar/n$. It follows that if the shift distance between x and y is at most R then

$$\Pr_{h_u \in \mathcal{H}}[h_u(x) = h_u(y)] \geq \left(1 - \frac{ar}{n}\right)\left(1 - \frac{r+1+ar}{n}\right) \geq 1 - \frac{R(2a+1)+1}{n}.$$

If $x, y \in V(C)$ and $d_C(x,y) \geq cR$ then there are two cases: $(h'(x) - h'(y)) \bmod n \leq cR$ and $(h'(x) - h'(y)) \bmod n > cR$. We show an upper bound for the probability of the first case, and use the fact that in the second case, the probability that h_u assigns different values to x and y is at least $\frac{cR}{n}$. To bound the first case we divide into two sub-cases: $d_C(x,y) \geq n$ and $d_C(x,y) < n$. In the first sub-case, y is a random string, independent of x. By the proof to Lemma 2.9 and by union bound it holds that $\Pr_{y \in \Sigma^n}[(h'(x) - h'(y)) \bmod n \leq cR] \leq \frac{1+acR)}{n}$. The probability that the second sub-case occurs is bounded by $\Pr[(h'(x) - h'(y)) \bmod n \leq cR] \leq acR/n$, by the definition of LPHS and union bound. Therefore,

$$\Pr[h_u(x) = h_u(y)] \leq \frac{1+2acR}{n} + 1 - \frac{cR}{n} = 1 - \frac{cR(2a-1)+1}{n}.$$

■

For worst-case results consider input that is α-Good for the whole cycle C and windows of size n. That is, assume that for every every two strings $x, y \in V(C)$ it holds that a fraction α of the strings is different. If α is a constant then the result of Theorem 6.19 holds with a degradation in the probabilities p_1 and p_2 of at most $O(a/n)$.

A data structure for approximate near-neighbor searches can be constructed directly from binary-LSH for the shift metric. In the interest of brevity we describe a slightly different construction based on previous work on near-neighbor data structures for the L_1 metric.

Corollary 6.20. *Let Σ be an alphabet, let n, N be integers, $N > n$, and let $C \in \Sigma^N$ be α-Good. There exists a c-approximate R-near-neighbor data structure for the shift metric on the strings in C, for a constant c, $R = o(n)$, with size $\tilde{O}(mn \log|\Sigma| + m^{1+1/c})$ and search time $\tilde{O}(m^{1/c})$.*

Proof. Let the strings in the data structure be $x_1, \ldots, x_m$ and let $\mathcal{H}'$ denote the LPHS family constructed in Theorem 3.5. Choose k random functions $h_1, \ldots, h_k \in \mathcal{H}'$, $k = \omega(\log nm)$. For each h_j, construct the c-approximate near-neighbor data structure for the L_1 metric given by Andoni and Indyk in [5, 3] on inputs $h_j(x_1), \ldots, h_j(x_m)$. In each cell that stores $h_j(x_i)$ store a pointer to x_i.

Search for a near-neighbor to a string y by running an independent search on each structure for h_j. The result is a list of possible neighbors $x_{i_1}, \ldots, x_{i_t}$, such that for each x_{i_ℓ} for at least one h_j it holds that the $|h_j(x_{i_\ell}) - h_j(y)| \leq cR$. To test whether h_j correctly measures the shift distance between y and x_{i_ℓ} choose at random $\omega(\log n)$ locations in y and check that the shifted locations in x_{i_ℓ} are identical.

The near-neighbor data structure for the L_1 metric is of size $\tilde{O}(m^{1+1/c})$, which together with the m strings of length n that the structure must store implies the size of the structure in the statement of the corollary. The search time in the L_1 structure is dominated by computing the distance of $O(m^{1/c})$ points to the query point. In the current structure that requires $\tilde{O}(m^{1/c})$ with the $\tilde{O}$ notation used for polylogarithmic factors in nm.

Failing to find a near neighbor can have one of two causes. The first is failure in the L_1 search structure, which has negligible probability [5, 3]. The second is that the shift distance between y and some x_i is small, but the difference $|h_j(y) - h_j(x_i)|$ is large. The probability of that event for a specific h_j is $\tilde{O}(1/n)$ by Theorem 3.5 and the probability that it will occur for all functions h_j is negligible. Reporting a distant string as a neighbor can occur only if $\omega(\log n)$ locations in y are identical to the same locations plus a small shift (less than R) in x_i, which has negligible probability for substrings of an α-Good C. ■

Acknowledgements. We thank Piotr Indyk, Leo Reyzin, David Woodruff, and anonymous reviewers for helpful pointers and suggestions.

E. Boyle was supported by AFOSR Award FA9550-21-1-0046, ERC Project HSS (852952), and a Google Research Scholar Award. I. Dinur was supported by ISF grant 1903/20 and ERC starting grant 757731 (LightCrypt). N. Gilboa was supported by ISF grant 2951/20, ERC grant 876110, and a grant by the BGU Cyber Center. Y. Ishai was supported by ERC Project NTSC (742754), ISF grant 2774/20, and BSF grant 2018393. N. Keller was supported by ERC starting grant 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. O. Klein was supported by the Clore Scholarship Programme.

References

[1] Adi Akavia, Hayim Shaul, Mor Weiss, and Zohar Yakhini. Linear-regression on packed encrypted data in the two-server model. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019*, pages 21–32. ACM, 2019.

[2] Quentin Alamélou, Paul-Edmond Berthier, Chloé Cachet, Stéphane Cauchie, Benjamin Fuller, Philippe Gaborit, and Sailesh Simhadri. Pseudoentropic isometries: A new framework for fuzzy extractor reusability. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 673–684. ACM, 2018.

[3] Alexandr Andoni. Approximate nearest neighbor problem in high dimensions. Master of Engineering Thesis, Massachusetts Institute of Technology, 2005.

[4] Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: sketching edit and shift distances. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 931–940. ACM, 2013.

[5] Alexandr Andoni and Piotr Indyk. Efficient algorithms for substring near neighbor problem. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1203–1212. Society for Industrial and Applied Mathematics, 2006.

[6] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE, 2006.

[7] Alexandr Andoni, Piotr Indyk, Dina Katabi, and Haitham Hassanieh. Shift finding in sublinear time. In *SODA 2013*, pages 457–465, 2013.

[8] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *STOC 2003*, pages 316–324, 2003.

[9] Elette Boyle, Itai Dinur, Niv Gilboa, Yuval Ishai, Nathan Keller, and Ohad Klein. Locality-preserving hashing for shifts with connections to cryptography. In *ITCS 2022*.

[10] Elette Boyle, Itai Dinur, Niv Gilboa, Yuval Ishai, Nathan Keller, and Ohad Klein. On the noise sensitivity of locality-preserving hashing for shifts. Manuscript in preparation, 2021.

[11] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology - EUROCRYPT*, pages 337–367, 2015.

[12] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, pages 509–539, 2016. Full version: IACR Cryptology ePrint Archive 2016: 585 (2016).

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT 2017, Part II*, pages 163–193, 2017.

[14] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In *CRYPTO 2020, Part III*, pages 738–767, 2020.

[15] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[16] Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam D. Smith. Reusable fuzzy extractors for low-entropy distributions. *J. Cryptol.*, 34(1):2, 2021.

[17] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *STOC 2016*, pages 712–725, 2016.

[18] Nishanth Chandran, Vipul Goyal, Ryan Moriarty, and Rafail Ostrovsky. Position-based cryptography. *SIAM J. Comput.*, 43(4):1291–1341, 2014.

[19] Moses Charikar and Robert Krauthgamer. Embedding the Ulam metric into l_1. *Theory of Computing*, 2(11):207–224, 2006.

[20] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[21] Thomas M. Cover and B. Gopinath. *Open Problems in Communication and Computation.* Springer-Verlag, 1987.

[22] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[23] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In *CRYPTO 2018, Part III*, pages 213–242, 2018.

[24] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[25] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Kevin Liu, and Giulio Malavolta. Rate-1 trapdoor functions from the Diffie-Hellman problem. In *ASIACRYPT 2019, Proceedings, Part III*, pages 585–606, 2019.

[26] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In *CRYPTO 2019*, pages 3–32, 2019.

[27] Benjamin Fuller, Xianrui Meng, and Leonid Reyzin. Computational fuzzy extractors. *Inf. Comput.*, 275:104602, 2020.

[28] Benjamin Fuller, Leonid Reyzin, and Adam D. Smith. When are fuzzy extractors possible? *IEEE Trans. Inf. Theory*, 66(8):5282–5298, 2020.

[29] Sanjam Garg, Mohammad Hajiabadi, and Rafail Ostrovsky. Efficient range-trapdoor functions and applications: Rate-1 OT and more. In *TCC 2020, Proceedings, Part I*, pages 88–116, 2020.

[30] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Advances in Cryptology - EUROCRYPT*, pages 640–658, 2014.

[31] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In David Zuckerman, editor, *FOCS 2019*, pages 1101–1120.

[32] Charles Herder, Ling Ren, Marten van Dijk, Meng-Day (Mandel) Yu, and Srinivas Devadas. Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Trans. Dependable Secur. Comput.*, 14(1):65–82, 2017.

[33] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh S. Vempala. Locality-preserving hashing in multidimensional spaces. In *STOC 1997*, pages 618–625, 1997.

[34] Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1168–1179. IEEE, 2020.

[35] Nathan Linial and Ori Sasson. Non-expansive hashing. In *STOC 1996*, pages 509–518, 1996.

[36] Henrik Ohlsson, Yonina C Eldar, Allen Y Yang, and S Shankar Sastry. Compressive shift retrieval. *IEEE Transactions on Signal Processing*, 62(16):4105–4113, 2014.

[37] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, pages 678–708, 2021.

[38] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). *IEEE Trans. Information Theory*, 24(1):106–110, 1978.

[39] John M Pollard. Monte carlo methods for index computation mod p. *Mathematics of computation*, 32(143):918–924, 1978.

[40] Barna Saha. The Dyck language edit distance problem in near-linear time. In *FOCS 2014*, pages 611–620, 2014.

[41] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT 97*, pages 256–266, 1997.

[42] Yunhua Wen, Shengli Liu, and Shuai Han. Reusable fuzzy extractor from the decisional diffie-hellman assumption. *Des. Codes Cryptogr.*, 86(11):2495–2512, 2018.

Appendix

A LPHS Results Based on Iterative Random Walks [23]

In the rest of this section, we summarize the IRW algorithm used to derive Theorem 3.5. Note that it is sufficient to derive the theorem for $b \geq 3 \log n$, while the theorem for $b < 3 \log n$ follows from Lemma 2.7.

A.1 The Basic LPHS

We begin by describing the min-based LPHS [13] in Algorithm 10 and refer to it as the basic LPHS. It scans the d values $x[0], x[1], \ldots, x[d-1]$ and chooses the index i_{min} for which $x[i]$ is minimal. The output of the LPHS is $\mathrm{Basic}_d(x) = i_{min}$.

The motivation behind the algorithm is apparent: $\mathrm{Basic}_d(x)$ and $\mathrm{Basic}_d(x \lll 1)$ scan the two lists $x[0], x[1], \ldots, x[d-1]$ and $x[1], x[2], \ldots, x[d]$, respectively. These lists have $d-1$ common symbols, and with high probability, the minimal value among $x[0], x[1], \ldots, x[d-1], x[d]$ is unique and obtained on a common symbol, implying $\mathrm{Basic}_d(x) = \mathrm{Basic}_d(x \lll 1) + 1$, as desired.

Error probability. The following lemma calculates the error probability of the basic DDL algorithm as a function of d.

Lemma A.1. *The error probability of the basic LPHS is*

$$\Pr_x[\mathrm{Basic}_d(x) - \mathrm{Basic}_d(x \lll 1) \neq 1] \leq \frac{2}{1+d} + 1/n.$$

Proof. Executions $\mathrm{Basic}_d(x)$ and $\mathrm{Basic}_d(x \lll 1)$ scan the two lists $x[0], x[1], \ldots, x[d-1]$ and $x[1], x[2], \ldots, x[d]$, respectively. If the minimal value is obtained on a symbol $x_{min} = x[i_{min}]$ which is queried by both, then we have $\mathrm{Basic}_d(x) = i_{min}$ and $\mathrm{Basic}_d(x) = i_{min} - 1$, implying that $\mathrm{Basic}_d(x) - \mathrm{Basic}_d(x \lll 1) = 1$ and the executions are successful. Similarly, an error occurs when the minimal value on the symbols $x[0], \ldots, x[d]$ is obtained on a symbol computed only by one party, namely on one of the 2 symbols $x[0]$ or $x[d]$. Since the symbols are uniform (and since $x[0], \ldots, x[d]$

are distinct, except with probability at most $1/n$), this occurs with probability $2/(1+d)$ and the lemma follows. ■

An important quantity that plays a role in the more advanced LPHS constructions is the output difference of the executions in case they fail to synchronize on the same symbol $x[i_{min}]$ (i.e., their output difference is not 1). Clearly, we have $|\text{Basic}_d(x) - \text{Basic}_d(x \ll 1) - 1| \leq d + 1$.

Algorithm 10: $\text{Basic}_d(x)$

```
begin
    $i \leftarrow 0$, $min \leftarrow \infty$;
    while $i < d$ do
        $t \leftarrow x[i]$;
        if $t < min$ then
            $i_{min} \leftarrow i$, $min \leftarrow t$;
        end
        $i \leftarrow i + 1$;
    end
    Output $i_{min}$;
end
```

A.2 The Random Walk LPHS

The random walk LPHS is useful when the shift is bounded by R, which is (much) bigger than 1 (see Definition 2.2). We think of two LPHS executions as two parties A and B that perform (pseudo) random walks on the symbols of the string $x \in \Sigma_b^n$, starting from $x[0]$ and $x[r]$, respectively. For a parameter L, the step length of each party is uniformly distributed in $[1, L-1]$, and is determined by a shared random function $\psi_{L-1} : \Sigma_b \to [1, L-1]$, as $\psi_{L-1}(x[i])$.

Algorithm 11 describes the random walk LPHS, parameterized by (L, d) which determine the maximal step length and the number of steps, respectively. The algorithm closely resembles Pollard's "kangaroo" random walk algorithm for solving the discrete logarithm in an interval problem using limited memory [39].

Lemma A.2 ([23], Lemma 6 (adapted))**.** *For any positive integer $r \leq R$,*

$$\Pr_x[\text{RW}_{L,d}(x) - \text{RW}_{L,d}(x \ll r) \neq r] = O\left(\frac{r/L + L}{d}\right).$$

In particular, a choice of $L = \sqrt{R}$ gives

$$\Pr_x[\text{RW}_{L,d}(x) - \text{RW}_{L,d}(x \ll r) \neq r] = O\left(\frac{\sqrt{R}}{d}\right).$$

Proof (sketch). Similarly to the proof of Lemma A.1, if the minimal value in both walks is obtained on the same symbol $x[j_{min}]$ (queried by both), then $\text{RW}_{L,d}(x) - \text{RW}_{L,d}(x \ll r) = r$. Otherwise, we say that the parties err, and our goal is to upper bound the error probability.

Suppose that A lands on a symbol queried by B's walk after performing M steps. From this stage, the walks coincide on the remaining $r - M$ steps. Since the symbols are uniform, then the probability that the minimal value in both executions is not obtained on the same symbol and the parties err is $O(M/d)$. In the following, we argue that the expected value of M is $O(r/L + L)$. A formal proof then shows that M is tightly concentrated around its expectation, which is sufficient to upper bound the error probability by $O(M/d) = O(\frac{r/L+L}{d})$, as required.

In order to estimate the expectation of M, partition A's walk into two stages, where the first stage ends when A's walk reaches (or goes beyond) B's starting point. Since the initial distance between the parties is r and step size of A is uniform in $[1, L-1]$, the expected number of steps in the first stage is $O(r/L)$. In the second stage, due to the uniformity of ψ_{L-1}, each step of A has probability of at least $1/L$ to land on a symbol queried by B's walk. Therefore, the expected number of steps until this event occurs is $O(L)$. ■

We further note that the parties travel a distance of $O(L \cdot d)$. In case the parties err, then their final distance is at most $O(R + L \cdot d)$, which evaluates to $O(R + \sqrt{R} \cdot d)$ for $L = \sqrt{R}$, and $O(\sqrt{R} \cdot d)$ when $d = \Omega(\sqrt{R})$.

Algorithm 11: $\mathrm{RW}_{L,d}(x)$

```
begin
    $j \leftarrow 0$, $i \leftarrow 0$, $min \leftarrow \infty$;
    while $i < d$ do
        $t \leftarrow x[j]$;
        if $t < min$ then
            $j_{min} \leftarrow j$;
            $min \leftarrow t$;
        end
        $j \leftarrow j + \psi_{L-1}(x[j])$;
        $i \leftarrow i + 1$;
    end
    Output $j_{min}$;
end
```

A.3 The Iterated Random Walk LPHS

The starting point of the Iterated Random Walk (IRW) LPHS is Algorithm 10. It makes d queries and fails with probability of roughly $2/d$ as noted above. Let us assume that we run this algorithm with only $d/2$ queries, which increases the error probability by a factor of 2 to about $4/d$. On the other hand, we still have a budget of $d/2$ queries and we can exploit them to reduce the error probability.

It would be instructive to think about the executions $\mathrm{Basic}_d(x)$ and $\mathrm{Basic}_d(x \lll 1)$ as being performed by two parties A and B (respectively). After the first $d/2$ queries, we say that A (or B) is placed at index i if $x[i]$ is the minimal value in its computed set of size $d/2$. Assume that A and B fail to synchronize on the same index after the first $d/2$ queries (which occurs with probability of roughly $4/d$). Then, as noted above, they are placed at symbols which are at distance of at most $d/2 + 1$, i.e., if A is placed at $x[i]$ and B is placed at $x[j]$, then $|i - j| \leq d/2 + 1$.

Next, the parties then use the random walk of Algorithm 11 to try and synchronize.[14] Note that

[14] For simplicity, in Algorithm 11 j is initialized to 0 rather than to a number that depends on previous queries.

since both A and B use the same algorithm (which is deterministic given the shared randomness), they remain synchronized if they already are at the beginning of the walks.

The random walk algorithm is applied using $d/2$ queries and step length of $\sqrt{d}$. If the parties are not initially synchronized, according to Lemma A.2, the error probability of the random walk is $O(\sqrt{d}/d) = O(d^{-1/2})$ and the total error probability is $O(d^{-1} \cdot d^{-1/2} = d^{-3/2})$. In this case the distance between the parties is $O(\sqrt{d} \cdot d = d^{3/2})$.

The success probability can be further amplified by reserving an additional number of $O(d)$ queries to be used in another random walk. This is made possible by shortening the first two random walks, without affecting the failure probability significantly. Hence, assume that the parties fail to synchronize after the random walk (which occurs with probability of $O(d^{-3/2})$) and that we still have enough available queries for another random walk with $O(d)$ steps. As in the previous random walk, the step length is about the square root of the initial distance of the parties, namely $\sqrt{d^{3/2}} = d^{3/4}$. Applying Lemma A.2 with these parameters gives an error probability of $O(d^{3/4}/d) = d^{-1/4}$ for the random walk, and a total error probability of $O(d^{-3/2} \cdot d^{-1/4}) = O(d^{-7/4})$.

We continue executing random walk iterations with a carefully chosen step length (distributing a budget of $O(d)$ queries among them). After k random walk iterations, the error probability is reduced to about $d^{-2+2^{-k}}$ (and the expected distance between the parties is roughly $d^{2-2^{-k}}$). Choosing $k \approx \log\log d$ gives an error probability of $\tilde{O}(d^{-2+1/\log d}) = \tilde{O}(d^{-2})$. Additional optimizations allow to reduce the error probability to $O(d^{-2})$. The IRW LPHS is presented in Algorithms 11 and 12.

A.3.1 Details of the Iterated Random Walk LPHS

Algorithm 12 describes the full protocol which is composed of application of the basic LPHS (using $d_0 < d$ queries, reserving queries for the subsequent random walks), and then K additional random walks, where the k'th random walk is parameterized by (L_i, d_i) which determine its maximal step length and number of steps. Between each two iterations in Step 6, both parties are moved forward by a large (deterministic) number of steps, in order to guarantee independence between the iterations. We are free to choose the parameters $K, \{L_i, d_i\}$, as long as $\sum_{i=0}^{K} d_i = d$ is satisfied. By fine tuning the parameters which distribute the number of queries among the iterations and select the step length of each random walk, the following theorem is derived.

Theorem A.3. *[23], Theorem 2 (adapted) There exists a parameter set $PS = (K, d_0, \{(L_i, d_i)_{i=1}^{K}\})$, where $d = \sum_{i=0}^{K} d_i$ for which*

$$\Pr_x[\mathrm{IRW}_{PS}(x) - \mathrm{IRW}_{PS}(x \ll 1) \neq 1] \leq 2^{10.2+o(1)}/d^2.$$

This theorem immediately implies Theorem 3.5.

Remark (Cyclic vs. non-cyclic). The random walk makes queries within an interval of size bounded by $O(d^2)$, hence if $n = \Omega(d^2)$ is large enough, the LPHS gives both a cyclic and non-cyclic LPHS with the same parameters.

A.3.2 The Cyclic Random Walk LPHS

Lemma A.4. *There exists a cyclic (n, b, d, δ)-LPHS with $d = \tilde{O}(n^{1/2})$ and $\delta = n^{-\omega(1)}$.*

This is dealt with in the IRW algorithm by appropriately shifting the input x itself.

Algorithm 12: $\mathrm{IRW}_{K,d_0,\{(L_i,d_i)_{i=1}^{K}\}}(x)$

```
begin
    $j \leftarrow \mathrm{Basic}_{d_0}(x)$;
    $p \leftarrow 0$;
    $k \leftarrow 1$;
    while $k \leq K$ do
        $J \leftarrow \sum_{i<k} d_i L_i$;
        $j \leftarrow j + J$;
        $p \leftarrow p + j$;
        $x \leftarrow x \ll j$;
        $j \leftarrow \mathrm{RW}_{L_i,d_i}(x)$;
        $i \leftarrow i + 1$;
    end
    Output $p + j$;
end
```

Proof (sketch). We assume that $b = (\log n)^a$ for some $a > 1$ (e.g., $b = \log^2 n$) such that the n symbols of x are all distinct with probability $1 - n^{-\omega(1)}$. If b is smaller, then apply Lemma 2.6 and obtain a bigger alphabet.

The main idea is to apply Algorithm 11 with parameters $L = \sqrt{n}$ and $d = \sqrt{n}$ repeatedly $m = (\log n)^a$ times and output the shift from the final j_{min} location.

Once again, we think of two invocations of the LPHS as two parties as performing random walks. We make sure that if the parties are synchronized at the beginning of application i of Algorithm 11, then they remain synchronized. On the other hand, if they are not synchronized, they will agree on the same symbol with some constant probability p, independently of all other applications. Therefore, the total error probability is $p^m = n^{-\omega(1)}$.

In order to make the m applications of the algorithm independent, in application $i \in \{1, 2, \ldots, m\}$, replace symbol j of x with $\phi_i(x[j])$, where each $\phi_i : \Sigma_b \to \Sigma_b$ is an independent (shared) random permutation. Furthermore, after application i, jump (mod n) by $\rho_i(x[j_{min}])$ (and start the next application from this location), where j_{min} is the output of the algorithm, and $\rho_i : \Sigma_b \to \mathbb{Z}_n$ is an independent (shared) random function.

Note that for every $0 < c < 1$, the probability that the parties start application i at distance at most $c \cdot n$ is at least $1/c$ (we define the distance as the minimal cyclic distance between the parties). For a sufficiently small c, Algorithm 11 (applied with $L = \sqrt{n}$ and $d = \sqrt{n}$) succeeds with constant probability. ■

A.3.3 Las Vegas LPHS for Big Shifts

Lemma A.5. *For $R = O(d)$, $n = \Omega(d^2)$, there exists an LPHS h that makes $O(d)$ queries on average such that*

$$\Pr_x[\exists\, r \in [R+1] \,:\, h(x) - h(x \ll r) \neq r] = \tilde{O}\left(\frac{R}{d^2}\right),$$

and for every x such that $\exists\, r \in [R+1] \,:\, h(x) - h(x \ll r) \neq r$, we have $h(x) = \bot$.

The construction gives both a cyclic and non-cyclic LPHS with the same parameters. We note that the lemma can be strengthened by proving a concentration inequality around the expected number of queries of h.

Proof. We assume that $b \geq 3 \log n$ (in case $b < 3 \log n$, the construction can be extended using Lemma 2.6 with logarithmic loss in the error probability).

The LPHS h works in two stages. It first runs Algorithm 10 $R+1$ times: $\mathrm{Basic}_d(x), \ldots, \mathrm{Basic}_d(x \lll R)$. Note that these algorithms require a total of $R+d+1$ queries to $x[0], \ldots, x[R+d]$. Let $S = \{\mathrm{Basic}_d(x \lll r) : r \in [R+1]\}$. In the second stage, h runs the IRW algorithm with d queries on each input in $\{x \lll i \,|\, i \in S\}$, and if all runs agree on the same symbol $x[j]$, then it outputs j. Otherwise, it outputs $\perp$.

The bound $O(R/d^2)$ on the probability of h outputting $\perp$ follows from the IRW analysis and Lemma 2.4. The number of queries of h is $R + d + 1 + |S| \cdot d$, where $|S|$ is the size of S. Consider a pair of executions $\mathrm{Basic}_d(x \lll i), \mathrm{Basic}_d(x \lll (i+1))$ for $i \in [R]$. According to the analysis of Algorithm 10, $\mathrm{Basic}_d(x \lll (i+1))$ contributes a new element to S (that is different from $\mathrm{Basic}_d(x \lll i)$) with probability $O(1/d)$. Hence the expected size of S is $O(R/d) = O(1)$. ■