Mosaic: A Modular Framework for Private Fuzzy Heavy Hitters

Gayathri Garimella
Brown University, USA

Peihan Miao
Brown University, USA

Eileen Nolan
Brown University, USA

Phuoc Van Long Pham
Brown University, USA

Siddarth Sitaraman
Brown University, USA

November 20, 2026

Abstract

We present Mosaic, a modular cryptographic framework for *Private Fuzzy Heavy Hitters Detection*, a new problem introduced and formalized in this work. In this problem, a service provider aims to identify the most frequent items (i.e., heavy hitters) in private client data without learning any individual client's data. Motivated by real-world applications where inputs are inherently noisy, we consider *fuzzy* heavy hitters that are *close* to a sufficiently large number of client data points under a chosen distance metric.

Mosaic operates in the setting with two non-colluding servers and supports both the known-dictionary setting (where candidate fuzzy heavy hitters are fixed in advance) and the unknown-dictionary setting (where popular items must be discovered). The framework proceeds in two phases: during the *upload phase*, each client shares an encoding of their input with each server; in the *fuzzy matching phase*, the servers jointly identify fuzzy heavy hitters without learning any additional information about individual client's data.

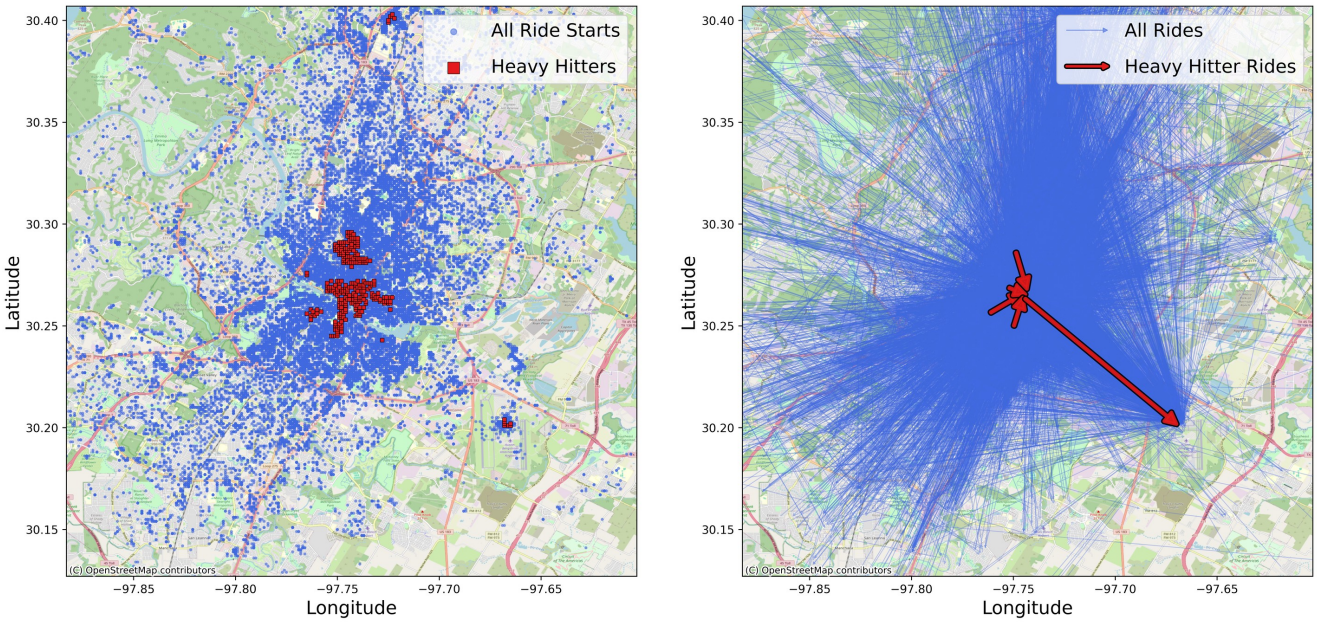
In the upload phase, Mosaic introduces a new cryptographic primitive called *Property-Based Function Secret Sharing (PB-FSS)*, which relaxes the standard notion of FSS (Boyle et al., Eurocrypt 2015) in that the outputs of the FSS evaluations satisfy a certain property, rather than forming exact additive secret shares of the function output. We present a suite of new PB-FSS constructions for the L_∞ and L_p distance metrics from lightweight cryptographic techniques. Furthermore, we introduce new methods to prevent malicious client behavior by detecting malformed PB-FSS shares. In the fuzzy matching phase, Mosaic presents two approaches, one involving only the two servers, and one involving an additional trusted dealer to improve efficiency. All parties are assumed to be semi-honest during the fuzzy matching phase.

Finally, we implement Mosaic and evaluate it using real-world trace data from a ride-sharing service. In one scenario, Mosaic servers discovered over a thousand of the most popular ride start locations among 21 thousand users within 30-90 seconds. On the user side, the protocol requires only lightweight computation (as low as 0.14 millisecond) and communication (as low as 9 KB).

1 Introduction

Service providers often seek to leverage user data to gain actionable insights and inform business decisions. However, user data is sensitive in many applications, and such data analytics must be performed in a privacy-preserving manner to protect individual users. In this work, we address the challenge of securely identifying *fuzzy heavy hitters* to discover emerging "hot spots" in the user data while offering strong privacy guarantees.

Consider the concrete example of an electric vehicle (EV) manufacturer that would like to analyze vehicular traffic patterns and determine locations with high demand for EV parking. Here, the *heavy hitters* correspond to the parking areas most frequently visited by users. However, GPS coordinates are subject to measurement noise, and two cars parked in the same lot may report slightly different locations. Finding *fuzzy* heavy hitters, which relies on approximate matching, is useful in this case. Our second



(a) Most popular start locations

(b) Most popular routes

Figure 1: Mosaic computes the most popular start locations and rides (i.e., fuzzy heavy hitters, shown in red) for a ride-sharing service in Austin, TX, without learning individual client data (shown in blue).

example considers an analyst that wants to infer the top start/end points or the top routes of rides hailed by users of ride-sharing services for planning and pricing purposes (see Figure 1). As a third example, consider a social media analytics provider that wants to find the most trending web searches within a time frame. The fuzzy heavy hitters in this setting are the most popular semantic meanings of users' queries.

Problem Statement. These motivating examples can be formalized as a *Private Fuzzy Heavy Hitters Detection* problem. Consider a large number of clients (i.e., users) each holding a private data point $\vec{q}_i$ from a high-dimensional space, and a small number of data-collection servers who wish to identify popular inputs up to some notion of similarity, without learning any individual client's data. Specifically, given a distance metric $\mathsf{dist}(\cdot,\cdot)$ (e.g., L_p, L_∞, Hamming), a distance threshold δ, and a frequency threshold $t \in \mathbb{N}$, the servers seek to find all points $\vec{x}$ such that at least t client inputs lie within distance δ to $\vec{x}$, namely,

$$\{\vec{x} : |\{\vec{q}_i : \mathsf{dist}(\vec{x}, \vec{q}_i) \leq \delta\}| \geq t\}\,.$$

In the aforementioned examples, EV charging and ride-sharing can be modeled under the L_∞ distance

metric. The search-query example can be modeled under L_2, where queries are embedded as high-dimensional vectors, and semantically similar queries remain close under L_2 distance.

Our Result. We formalize the problem of private fuzzy heavy hitters detection and present Mosaic, a modular framework that solves the problem using lightweight cryptographic techniques. Mosaic works in the setting where a large set of clients communicate with two non-colluding servers. The framework consists of two phases. In the *upload phase*, each (resource-limited) client performs lightweight computation on their private data point $\vec{q}_i$ and sends a single message to each server, after which they may go offline. In the *fuzzy matching phase*, the two servers $\mathcal{S}_0$ and $\mathcal{S}_1$ jointly identify the fuzzy heavy hitters without learning any additional information about any individual client's data.

Mosaic supports two variants of detecting heavy hitters: *known dictionary* and *unknown dictionary*. In the known dictionary setting, the servers determine a candidate set of heavy hitters a priori, referred to as the *dictionary*. They aim to identify for each item $\vec{x}$ in the dictionary whether it is a fuzzy heavy hitter, namely at least t client inputs are within distance δ to $\vec{x}$. This setting is applicable to scenarios where the service provider already knows which candidate items are of potential interest, e.g., a ride-sharing analytics provider that wants to monitor a fixed set of landmarks. The unknown dictionary setting is more challenging, where the two servers do not know a list of candidates beforehand, and must discover them. This setting reflects applications such as finding traffic hotspots in a city or trending search queries, where a predefined dictionary may not exist.

We present a suite of cryptographic building blocks that work for both L_∞ and L_p distance metrics. L_∞ measures the largest coordinate difference between two points and is well suited for location data, while the L_p distance (such as L_2) captures overall geometric closeness and is commonly used in high-dimensional embeddings. We assume all parties are semi-honest as a baseline, and present novel techniques to protect against malicious clients.

At a technical level, Mosaic builds on a relaxed notion of function secret sharing (FSS) [BGI15]. In standard FSS, one may secret share a function into two keys, and evaluations on the two keys produce additive secret shares of the function output. This strict requirement becomes a bottleneck for constructing FSS for the distance functions of the L_∞ and L_p distance metrics. In this work, we relax the notion by requiring the evaluations to satisfy a certain property, formalized as property-based FSS (PB-FSS). This relaxation preserves the privacy guarantees of FSS while allowing for efficient constructions for the distance metrics of interest. The relaxed notion also naturally extends to the *incremental* setting where evaluation can be done on prefixes, which enables efficient discovery of fuzzy heavy hitters when the dictionary of candidates is unknown in advance. Furthermore, we introduce new techniques to detect malformed PB-FSS key shares.

In the fuzzy matching phase, the two servers evaluate the PB-FSS keys on candidate points and follow a two-step approach to detect fuzzy heavy hitters: (1) private property testing, and (2) private threshold comparison. We present two approaches to each step, one based on generic secure two-party computation, and another introducing a trusted dealer to generate correlated randomness to improve efficiency.

Finally, we implement Mosaic and run experiments for the ride-sharing application. In one setting, the client-side computation for the upload phase takes less than a millisecond and server-side phases can process 21K clients within 30-90 seconds.

Limitations of prior work. Prior work has developed protocols that allow service providers to identify heavy hitters in datasets [BBCG$^+$21, RZCGAP24, KMPP24]. However, these approaches consider only *exact* heavy hitters, whereas many real-world scenarios require accounting for approximate or noisy inputs. A straightforward approach to the fuzzy heavy hitters problem is to reduce it to the exact setting: each client would enumerate over all possible points in the neighborhood of its input (i.e., all $\vec{x}$ such that $\mathsf{dist}(\vec{x}, \vec{q}) \leq \delta$) and then upload these candidates to the servers as if they were exact inputs. The servers then run an exact heavy hitters protocol over this expanded dataset, treating any discovered heavy hitter as an approximate heavy hitter of the original data. However, this approach quickly becomes infeasible because the size of each client's enumerated neighborhood grows with $O(\delta^d)$, where δ is the

distance threshold and d is the dimension of the input space. Consequently, both client-side computation and server-side computation and storage scale with the total volume of enumerated neighboring points, making the approach impractical. In contrast, Mosaic enables clients to efficiently encode their inputs without requiring them to enumerate all nearby points. As a result, we ensure minimal communication between an individual client and the servers and eliminate the exponential blowup in server computation and storage. Concretely, in some scenarios, Mosaic outperforms this baseline approach by 48× in terms of key size, 80× in terms of key generation time, and 35× in terms of server computation.

Our contributions. In summary, our contributions are as follows.

- We introduce the private fuzzy heavy hitters detection problem, which models real-world analytics applications where approximate matches are needed when determining popular items. We formalize both the known dictionary and unknown dictionary settings under standard distance metrics, including L_∞ and L_p.
- We present Mosaic, a modular cryptographic framework to solve the private fuzzy heavy hitters problem in the two-server setting. Mosaic consists of two phases: a client upload phase and a fuzzy matching phase. We present several instantiations for each phase of the framework, which can be composed to meet application-specific requirements.
- We introduce a new notion called property-based function secret sharing (PB-FSS) and its incremental variant PB-iFSS, which relaxes the correctness requirements of FSS in that evaluations on the key shares satisfy a specific property rather than providing exact secret shares of the function output. We develop a suite of lightweight PB-FSS and PB-iFSS constructions for the L_∞ and L_p distance metrics.
- In the fuzzy matching phase, we design new protocols for private property testing and private threshold comparison.
- Our framework is semi-honest secure as a baseline. In addition, we develop new sketching techniques to detect malformed PB-(i)FSS shares to protect against malicious clients.
- We implement Mosaic in Rust and evaluate its performance using real-world ride-sharing data from RideAustin, demonstrating the efficiency and scale required for real-world applications.

2 Technical Overview

Starting point: exact heavy hitters from DPF. We begin with the *exact* heavy hitters problem for the known-dictionary case. In this setting, each client holds a private input $\vec{q}$, and two non-colluding servers hold a dictionary $\mathcal{D}$ of candidate heavy hitters (unknown to the clients). The two servers want to identify, for each $\vec{x} \in \mathcal{D}$, whether $\vec{x}$ is a heavy hitter i.e. whether $|\{\vec{q} : \vec{q} = \vec{x}\}| \geq t$, without learning any information about any client's individual input.

We revisit the solution proposed in prior work [BBCG+21] using distributed point functions (DPFs) [GI14]. A DPF is a special function secret sharing (FSS) [BGI15] scheme for *point functions*. Specifically, each client holding input $\vec{q}$ considers the following point function:

$$f_{\vec{q}}(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} = \vec{q} \\ 0 & \text{otherwise} \end{cases}.$$

A DPF (and more generally, FSS) consists of two algorithms, Share and Eval. The Share algorithm allows the client to secret share this point function into two keys (k_0, k_1), and sends k_b to server $\mathcal{S}_b$. DPF provides two guarantees: **(1) privacy:** each individual key k_b does not reveal any information about $\vec{q}$, and **(2) correctness:** the two keys encode additive secret shares of the point function $f_{\vec{q}}(\cdot)$. Specifically, for any point $\vec{x}$,

$$\mathsf{Eval}(k_0, \vec{x}) + \mathsf{Eval}(k_1, \vec{x}) = \begin{cases} 1 & \text{if } \vec{x} = \vec{q} \\ 0 & \text{otherwise} \end{cases}.$$

By the privacy guarantee of DPF, each server $\mathcal{S}_b$ holding one key k_b gains no information about the client's input point $\vec{q}$. Correctness then ensures that, when the two servers evaluate their keys on a candidate $\vec{x} \in \mathcal{D}$, they obtain additive shares of the indicator $\mathbb{1}_{\vec{x}=\vec{q}}$.

For each $\vec{x} \in \mathcal{D}$, to determine whether $\vec{x}$ is a heavy hitter, the two servers compute $\mathsf{Eval}(k_b, \vec{x})$ for each client to obtain additive secret shares of $\mathbb{1}_{\vec{x}=\vec{q}}$. The servers then aggregate all these secret shares and reconstruct the final result to learn the total number of $\vec{q}$'s that equal $\vec{x}$, namely $|\{\vec{q} : \vec{q} = \vec{x}\}|$.

Fuzzy heavy hitters from FSS. We first extend this idea to fuzzy heavy hitters with a known dictionary. Instead of obtaining an indicator for exact matching $\mathbb{1}_{\vec{x}=\vec{q}}$, the servers want to get secret shares of an indicator for *fuzzy* matching $\mathbb{1}_{\mathsf{dist}(\vec{x},\vec{q})\leq\delta}$. A natural idea is to leverage FSS for a broader class of functions. Specifically, each client with input $\vec{q}$ generates FSS key shares for the following function that captures fuzzy matches:

$$f_{\vec{q}}(\vec{x}) = \begin{cases} 1 & \text{if } \mathsf{dist}(\vec{x}, \vec{q}) \leq \delta \\ 0 & \text{otherwise} \end{cases}.$$

If we can construct efficient FSS for the desired distance metric, then the rest of the protocol follows exactly the same way. However, existing FSS constructions for the L_∞ distance metric require both the key size and evaluation time to grow exponentially in the dimension [BGI16]. Moreover, there is currently no FSS construction for the L_p distance metric.

Property-based FSS. To address these challenges, we relax the correctness requirement of FSS and introduce a new notion called *property-based FSS (PB-FSS)*. The high-level idea is to replace the exact secret share requirement of the evaluations $\mathsf{Eval}(k_b, \vec{x})$ with the guarantee that the two evaluations produce outputs that satisfy a predicate indicating whether $\vec{x}$ is within distance δ to $\vec{q}$. In more detail, rather than simply outputting secret shares of 0 or 1, the two evaluations $\mathsf{Eval}(k_0, \vec{x})$ and $\mathsf{Eval}(k_1, \vec{x})$ output values that satisfy a specific *property* iff $\mathsf{dist}(\vec{x}, \vec{q}) \leq \delta$. The property we define for L_∞ distance is *equality*, namely whether $\mathsf{Eval}(k_0, \vec{x}) = \mathsf{Eval}(k_1, \vec{x})$; the property defined for L_p is *μ-bounded*, namely whether $\mathsf{Eval}(k_0, \vec{x}) + \mathsf{Eval}(k_1, \vec{x}) \leq \mu$ for a pre-defined μ.

Such relaxations allow for efficient constructions of PB-FSS for both L_∞ and L_p distance metrics. After the FSS evaluation $\mathsf{Eval}(k_b, \vec{x})$ by each server $\mathcal{S}_b$, the two servers perform a **private property testing** protocol to obtain secret shares of the indicator $\mathbb{1}_{\mathsf{dist}(\vec{x},\vec{q})\leq\delta}$, which we elaborate later. The rest of the protocol then follows the same way as before.

Handling unknown dictionary. We now consider the unknown-dictionary setting, where the dictionary $\mathcal{D}$ is not fixed in advance and the servers must therefore discover fuzzy heavy hitters. The high-level idea is to perform a binary search on the servers by leveraging property-based *incremental* FSS (PB-iFSS), where the evaluation algorithm can take any prefix of $\vec{x}$ as input and output whether the prefix could *potentially* satisfy the matching property. Specifically, the servers start with the empty prefix, expand the prefix bit by bit, testing whether the expanded prefix could potentially have sufficient support (with at least t matched clients), and recurse only on prefixes that pass the test.

To test whether a prefix could potentially match at least t clients, we define an *optimistic* correctness requirement for Eval in PB-iFSS. When evaluated on a prefix $\vec{p}$, $\mathsf{Eval}(k_0, \vec{p})$ and $\mathsf{Eval}(k_1, \vec{p})$ output values that satisfy the matching property iff there exists a completion of $\vec{p}$ (namely, a point with prefix $\vec{p}$) within distance δ to the client's point $\vec{q}$. This optimistic correctness yields sound pruning: if the aggregated optimistic evaluation indicates that fewer than t clients could possibly match under any completion of $\vec{p}$, then no descendant full-length point with prefix $\vec{p}$ can be a fuzzy heavy hitter, and the entire subtree can be safely pruned.

New constructions for PB-FSS. We present a suite of new constructions for property-based (incremental) FSS for the L_∞ and L_p distance metrics, with equality and μ-bounded properties, respectively. All our constructions rely only on lightweight, symmetric-key cryptographic operations such as AES and hash functions. These constructions follow two main paradigms, one based on oblivious

key-value stores (OKVS) [GPR+21] and one based on (incremental) distributed comparison functions (DCF) [BGI15, BGI16]. We present the definition and constructions of PB-FSS in Section 4.

Private property testing. Recall that after obtaining the PB-FSS evaluations $y_b = \mathsf{Eval}(k_b, \vec{x})$ by each server $\mathcal{S}_b$, the two servers need to perform a private property testing protocol to get secret shares of the indicator $\mathbb{1}_{\mathsf{dist}(\vec{x},\vec{q})\leq\delta}$. The properties we define for the L_∞ and L_p distance metrics are *equality* and μ-*bounded*, respectively. Specifically, the servers need to test whether $y_0 = y_1$ in the L_∞ case, and whether $y_0 + y_1 \leq \mu$ in the L_p case. We present two approaches to this problem, one based on generic secure two-party computation (2PC) and one introducing a trusted dealer who generates correlated randomness for the two servers. We describe the protocols in Section 5.

Private threshold comparison. We take a step back and revisit the information revealed to the two servers. After private property testing, the servers aggregate per-client indicators to obtain secret shares of the total number of client points that are close to a candidate heavy hitter $\vec{x}$. Then they reconstruct the value in the clear and compare with the threshold t. As a result, the servers learn not only whether $\vec{x}$ is a fuzzy heavy hitter, but also the exact number of matching clients. This leakage matches the guarantees provided by prior work on exact heavy hitters detection [BBCG+21], but may be undesirable in some settings.

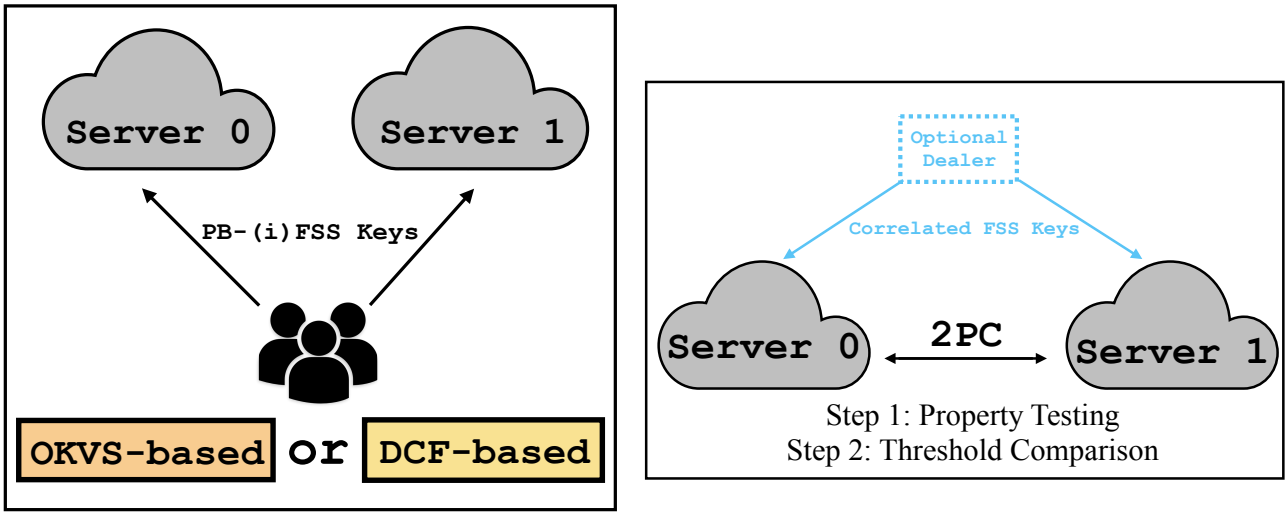
In this work, we further strengthen privacy by avoiding the explicit reconstruction of secret-shared counts. Instead, the servers perform a private threshold comparison protocol that outputs only a single bit indicating whether the count meets the threshold t. Similar to private property testing, we propose two approaches to this comparison, one based on generic 2PC and one relying on a trusted dealer to generate correlated randomness.

Detecting malformed PB-FSS shares. So far our framework assumes that all clients are semi-honest and generate well-formed PB-FSS shares. To further strengthen the robustness of Mosaic, we introduce techniques that detect malformed PB-FSS shares arising from malicious client behavior. Prior work [BGI16, BBCG+21] provides sketching techniques for detecting malformed shares in DPF and DCF constructions, which we build upon. In Section 7, we develop new sketching methods for our DCF-based PB-FSS (or PB-iFSS) constructions, for both the L_∞ and L_p distance metrics.

2.1 The Mosaic Framework

In summary, the Mosaic framework proceeds as follows:

- **Client Upload Phase.** Each client holds a d-dimensional input point $\vec{q}$. The client locally encodes this point using a PB-FSS (resp. PB-iFSS) scheme in the known (resp. unknown) dictionary setting, which produces two key shares. The client then sends these two key shares to non-colluding servers $\mathcal{S}_0$ and $\mathcal{S}_1$, respectively, after which the client may go offline (see Figure 2a).
- **Fuzzy Matching Phase.** Upon receiving all clients' key shares, the two servers collaboratively perform the following two steps to detect fuzzy heavy hitters. In the known dictionary setting, the servers evaluate each client's PB-FSS key share on potential candidates, whereas in the unknown dictionary setting, the servers discover the candidates via PB-iFSS dynamically using a prefix-tree-based search algorithm. We present two approaches for both steps, one based on generic 2PC and one involving a trusted dealer (see Figure 2b).
  - **Step 1: Private Property Testing.** For each potential candidate (prefix) $\vec{x}$, once the servers obtain PB-FSS (or PB-iFSS) evaluations, they jointly run a private property testing protocol to obtain secret shares of the indicator $\mathbb{1}_{\mathsf{dist}(\vec{x},\vec{q})\leq\delta}$ for each client. By summing up these shares, they obtain additive secret shares of a global count s, which is the number of clients whose inputs are close to the candidate.



(a) Client Upload Phase (b) Fuzzy Matching Phase

Figure 2: Mosaic Framework

– **Step 2: Private Threshold Comparison.** The servers then engage in a private threshold comparison protocol to determine whether $s \geq t$, where t is a public frequency threshold. The comparison reveals only the final one-bit outcome (whether the candidate qualifies as a fuzzy heavy hitter) and nothing else about the underlying count s or individual clients' indicators.

3 Preliminaries

3.1 Notation and Parameters

The statistical and computational security parameters are denoted by λ and κ, respectively. Let $\{0,1\}^u$ denote the set of all binary strings of length u, and let $\{0,1\}^{\leq u}$ denote the set of all binary strings of length at most u (including the empty string). For a point $\alpha \in \{0,1\}^u$, let α_i denote its i-th bit, and $\alpha_{(i)}$ the prefix of length i. We also sometimes use a length-u string to represent an element in $\mathbb{Z}_{2^u}$, or an integer. We denote the empty string by ε. We use $[n]$ to denote the range $\{1, 2, \ldots, n\}$. We use $[\![x]\!]$ to denote *additive* shares for x.

We write $x \prec y$ (and equivalently $y \succ x$) if the string x is a prefix of the string y (or y is a completion of x). For d-dimensional points $\vec{x}, \vec{y}$, we write $\vec{x} \prec \vec{y}$ if and only if $x_1 \prec y_1, \ldots, x_d \prec y_d$. We allow comparison between two bit-strings that are not equal in length, and say that $x < y$ if there exists prefix $x' \prec y$ such that $x < x'$.

Protocol Parameters. We consider the Fuzzy Heavy Hitters problem among N clients, where each client holds a d-dimensional point $\vec{q} \in (\{0,1\}^{h_1})^d$. Each coordinate is represented using h_1 bits. For our constructions, the output of the *property-based (incremental) FSS* has $h_2 \cdot d$ bits for the L_∞ distance (the concatenation of d bit strings of length h_2), and h_2 bits for the L_p distances (via modular addition of d values in $\mathbb{Z}_{2^{h_2}}$). The *Property Testing* protocol outputs additive secret shares in $\mathbb{Z}_{2^{h_3}}$. The *Threshold Comparison* protocol outputs a single bit indicating whether a frequency threshold t is exceeded. These parameters are user-defined and can be adjusted based on efficiency and security requirements. A practical guide for selecting them is provided in the paper's full version.

Distance metrics. For two vectors $\vec{x} = (x_1, \ldots, x_d) \in (\{0,1\}^u)^d$ and $\vec{q} = (q_1, \ldots, q_d) \in (\{0,1\}^u)^d$, let $L_\infty(\vec{x}, \vec{q}) := \max_{i=1}^d |x_i - q_i|$, and $L_p(\vec{x}, \vec{q}) := \left(\sum_{i=1}^d |x_i - q_i|^p\right)^{1/p}$. For $\mathsf{dist} \in \{L_\infty, L_p\}$, define $\mathcal{B}_{\mathsf{dist},\vec{q},\delta} :=$

Table 1: Summary of Parameters

Parameter	Description
λ	Statistical security parameter
κ	Computational security parameter
N	Number of clients
d	Dimensionality of client inputs
h_1	Bit-length of each coordinate in client input $(\{0,1\}^{h_1})^d$
h_2	Bit-length of FSS output (per dimension or aggregate)
h_3	Bit-length of Property Testing output space $\mathbb{Z}_{2^{h_3}}$
δ	Distance threshold for fuzzy matching
t	Frequency threshold for heavy hitter determination

$\{\vec{x} \in (\{0,1\}^u)^d \mid \mathsf{dist}(\vec{x}, \vec{q}) \leq \delta\}$ as the d-dimensional dist-ball centered at $\vec{q}$ with radius δ.

Optimistic distance. Let $\vec{p} = (p_1, \ldots, p_d)$ be a d-dimensional prefix, where $p_i \in \{0,1\}^{\leq u}$ for all $i \in [d]$. The *optimistic distance* $\mathsf{dist}_{\mathrm{opt}}(\vec{p}, \vec{q})$ between a prefix $\vec{p}$ and a full vector $\vec{q} = (q_1, \ldots, q_d) \in (\{0,1\}^u)^d$ is defined as the minimum possible L_∞ (or L_p) distance achievable between $\vec{q}$ and any completion of $\vec{p}$ to length u. More formally: $\mathsf{dist}_{\mathrm{opt}}(\vec{p}, \vec{q}) := \min\left\{\mathsf{dist}(\vec{x}, \vec{q}) : \vec{x} \in (\{0,1\}^u)^d \wedge\ \vec{p} \prec \vec{x}\right\}$.

3.2 Secure Multi-party Computation [Yao86, GMW87]

We consider the secure computation functionality between two parties $\mathcal{F}_{\mathsf{2PC}}$. The threat model we consider in this work is semi-honest security, where both parties try to obtain more information but follow the protocol honestly. We follow the Universal Composition (UC) security definition for MPC, and refer readers to [Can01, Lin17] for more details of the security paradigm.

Definition 1 (Security of 2PC in the semi-honest model)**.** *Let $f = (f_1, f_2)$ be a two-party functionality where f_{idx} denotes the output to party idx. Given a protocol π, let $\mathrm{view}^\pi_{\mathsf{idx}}(\kappa, x_0, x_1)$ be the real-world view of party idx, and $\mathrm{output}^\pi(\kappa, x_0, x_1)$ be the real-world output, in a protocol with inputs being x_{idx} from the idx-th party. We say that the protocol π securely computes f in the presence of static semi-honest adversaries if there exist PPT simulators $\mathsf{Sim}_{\mathsf{idx}}, \mathsf{idx} \in \{0,1\}$ such that for all $x_0, x_1 \in \{0,1\}^*$ and security parameter κ, the simulated view in the ideal-world is computationally indistinguishable from the real-world:*

$$\begin{aligned} &\{\mathsf{Sim}_{\mathsf{idx}}(1^\kappa, x_{\mathsf{idx}}, f_{\mathsf{idx}}(x_0, x_1)),\ f(x_0, x_1)\}_{\kappa, x_0, x_1} \\ \cong_\kappa\ &\{\mathrm{view}^\pi_{\mathsf{idx}}(\kappa, x_0, x_1),\ \mathrm{output}^\pi(\kappa, x_0, x_1)\}_{\kappa, x_0, x_1}\ . \end{aligned}$$

3.3 Oblivious Key-Value Stores [GPR+21]

An OKVS is a pair of (randomized) algorithms $(\mathsf{Encode}, \mathsf{Decode})$. Algorithm Encode takes as input a set KV of (key, value) pairs $\{(k_i, v_i)\}_{i=1}^n$ and outputs a vector $\vec{P}$ that is "oblivious" from the input list. When calling Decode with $\vec{P}$, if k is a key of $(k, v) \in KV$, it is guaranteed that $\mathsf{Decode}(\vec{P}, k) = v$.

Definition 2 (Oblivious Key-Value Stores (OKVS) [GPR+21])**.** *An OKVS consists of algorithms Encode and Decode, with an associated key space $\mathcal{K}$ and value space $\mathcal{V}$.*

- Encode *takes as input a set $KV \subseteq \mathcal{K} \times \mathcal{V}$ of (key, value) pairs and outputs an object $\vec{P}$ (or, with statistically small probability, an error indicator $\perp$).*
- Decode *takes as input an object $\vec{P}$, any key $k \in \mathcal{K}$, and outputs a value v.*

An OKVS must satisfy the following properties:

- **Correctness:** *For all $KV \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys, and all $(k, v) \in KV$:*

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(KV), k) = v] = 1.$$

Note that we can invoke $\mathsf{Decode}(\vec{P}, k)$ on any key k on a given object $\vec{P}$.

- **Obliviousness:** *For all distinct $\{k_1^0, \ldots, k_n^0\}$ and distinct $\{k_1^1, \ldots, k_n^1\}$, the output of $\mathcal{R}(k_1^0, \ldots, k_n^0)$ is computationally indistinguishable from that of $\mathcal{R}(k_1^1, \ldots, k_n^1)$, where:*

$\underline{\mathcal{R}(k_1, \ldots, k_n)}$:
  for $i \in [n]$: $v_i \leftarrow \mathcal{V}$
  return $\mathsf{Encode}(\{(k_1, v_1), \ldots, (k_n, v_n)\})$

The obliviousness property guarantees that if the values are pseudorandom, then any two outputs of experiment $\mathcal{R}$, encoding different sets of keys are computationally indistinguishable. Additionally, the use of OKVS in our construction for PB-FSS depends on the Strong OKVS Extraction Conjecture (Conjecture 2.1 in [RS21]) for correctness. The conjecture, intuitively, states that if all n encoded (key, value) pairs have the form $(k_i, H(k_i))$, where H is a hash function modeled as a Random Oracle, then it is computationally infeasible to find another key k outside the list that also has $\mathsf{Decode}(\vec{P}, k) = H(k)$.

3.4 Function Secret Sharing

Given a function family $\mathcal{F}$, an FSS scheme [BGI15, BGI16, HN23, BKLS24] for this family consists of two algorithms: Share and Eval. The algorithm Share takes any function $f \in \mathcal{F}$ and outputs a pair of keys, each key being distributed to a different party. The two parties can then use their respective keys to run Eval on any input x, which gives additive secret shares of $f(x)$. We give the formal definition of (incremental) FSS in Definition 3.

Definition 3 ((Incremental) Function Secret Sharing [BGI15, BGI16, BBCG$^+$21])**.** *A two-party (incremental) function secret sharing scheme* FSS *(or* iFSS *for incremental) for the function family $\mathcal{F}$ with domain $\mathcal{D} = \{0,1\}^u$ ($\mathcal{D} = \{0,1\}^{\leq u}$ for incremental) and payload group $\mathbb{G}$, is a pair of algorithms* (Share, Eval) *with the following syntax:*

- $(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, f)$*: The randomized share function takes as input the security parameter κ and (the description of) function $f \in \mathcal{F}$, and outputs two key shares.*
- $y_{\mathsf{idx}} \leftarrow \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx}}, x)$*: The deterministic evaluation function takes as input a party index $\mathsf{idx} \in \{0,1\}$, the corresponding key share k_{idx} and input $x \in \mathcal{D}$, and outputs a value $y_{\mathsf{idx}} \in \mathbb{G}$.*

The (incremental) FSS scheme for $\mathcal{F}$ is secure if it satisfies the following two conditions:

Correctness: *For every (description of) function $f \in \mathcal{F}$ over input domain $\mathcal{D}$, for every input $x \in \mathcal{D}$ and security parameter κ, the evaluations add up to $f(x)$ with probability 1:*

$$\Pr\Big[y_0 + y_1 = f(x) \mid (k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, f), \; y_0 \leftarrow \mathsf{Eval}(0, k_0, x), \; y_1 \leftarrow \mathsf{Eval}(1, k_1, x)\Big] = 1. \tag{1}$$

Privacy: *There exists a PPT simulator* Sim *such that for all $\mathsf{idx} \in \{0,1\}$ and all (description of) function $f \in \mathcal{F}$, the following distributions are computationally indistinguishable in the security parameter κ:*

$$\boxed{\begin{array}{l}(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, f) \\ \text{return } k_{\mathsf{idx}}\end{array}} \cong_\kappa \mathsf{Sim}(1^\kappa, \mathsf{idx}, u). \tag{2}$$

Families of functions considered. In this work, we use function secret sharing schemes for different types of function, such as DPF (Distributed Point Functions) or DCF (Distributed Comparison Functions). Furthermore, we also provide some new types of function such as LDCF and RDCF, which are *incremental* variants of DCF that allows evaluating on *prefixes* as well. Looking ahead, these incremental variants allows the two servers to evaluate the prefixes while doing tree-based search for the Unknown Dictionary setting of the problem. All mentioned FSS schemes can be constructed following the GGM-tree paradigm [BGI16], and we omit the constructions in this work, only focusing on describing how the functions behave. In all function descriptions below, we take a domain $\mathcal{D} = \{0,1\}^u$ and the payload group being a modulo ring $\mathbb{G} = \mathbb{Z}_{2^k}$.

We use Distributed Point Functions, in which a function can be described by a *point* $\alpha \in \mathcal{D}$ and two *payloads* $a, b \in \mathbb{G}$ for output when either equality happens or not. Looking ahead, we use DPF in the dealer-based equality test (Section 5.2). Formally, the function $f^P_{\alpha,a,b}$ being shared has the following formula:

$$f^P_{\alpha,a,b}(x) = \begin{cases} a & \text{if } x = a, \\ b & \text{otherwise.} \end{cases} \tag{3}$$

Distributed Comparison Functions and its variants are used extensively in this work. DCF is used to implement dealer-based secure comparison (described in Section 5.2, with full protocol in Appendix C.4), while the variants LDCF and RDCF are used for the sharing phase in the unknown dictionary setting (modifications of protocols in Section 4.4). In a DCF, there is a *splitting point* $\alpha \in \mathcal{D}$, and two payload values $a, b \in \mathbb{G}$ for outputs to the left or right of the splitting point. We formally describe the function $f^D_{\alpha,a,b}$ below:

$$f^D_{\alpha,a,b}(x) = \begin{cases} a & \text{if } x \leq a, \\ b & \text{otherwise.} \end{cases} \tag{4}$$

The variant LDCF is an *incremental* variant of DCF, which allows evaluating on prefixes as well. The evaluation output is a if the prefix x evaluated is a prefix of a point $\chi \leq \alpha$.

$$f^{LC}_{\alpha,a,b}(x) = \begin{cases} a & \text{if } \exists \chi : x \prec \chi \wedge \chi \leq \alpha, \\ b & \text{otherwise.} \end{cases} \tag{5}$$

Symmetrically, the RDCF variant for prefixes sharply on the right of the splitting point is also needed:

$$f^{RC}_{\alpha,a,b}(x) = \begin{cases} b & \text{if } \exists \chi : x \prec \chi \wedge \chi \geq \alpha, \\ a & \text{otherwise.} \end{cases} \tag{6}$$

Distributed key generation. In this work, besides using function secret sharing for PB-FSS (would be introduced in Section 4), we also use FSS to implement 2-PC steps, such as comparing with a threshold. We make use of a distributed key generation procedure for FSS, where two parties receive keys for a function secret sharing, as well as *secret shares* for the parameter that the function used. For example, two parties may receive FSS for a DPF $f^P_{\alpha,1,0}$, along with a value α_{idx} for the idx-th party such that $\alpha_0 + \alpha_1 = \alpha$. We use a functionality for *secret-shared truth-table* (with similarity to correlated randomness) to model this distributed generation process. The functionality is shown in Figure 3. In this work, two parties realize this functionality with the help from a dealer.

4 Property-Based Function Secret Sharing

We introduce a new notion called **Property-Based (Incremental) Function Secret Sharing**, which relaxes the correctness requirement of FSS by incorporating a property-testing function. In this section, we present the formal definition of this notion, followed by a suite of constructions for the L_∞ and L_p distance metrics.

Parameters:

- Input domain $\{0,1\}^u$.
- Payload group $\mathbb{G}$.
- Function type: DPF or DCF.
- Public FSS payloads $a, b \in \mathbb{G}$

Functionality:

1. Sample $\alpha \in \{0,1\}^u$ and let $\vec{v}$ be defined as:
   - If function type is DPF: A unit vector where $v_\alpha = a$, and $v_i = b, \forall i \neq \alpha$.
   - If function type is DCF: A vector where $v_i = a, \forall i \leq \alpha$, and $v_i = b, \forall b > \alpha$.
2. Output as follows:
   - *Both parties honest:* Sample $\vec{v}_0 \xleftarrow{\$} \mathbb{G}^{2^u}, \alpha_0 \xleftarrow{\$} \{0,1\}^u$ and let $\vec{v}_1 \leftarrow \vec{v} - \vec{v}_0, \alpha_1 \leftarrow \alpha - \alpha_0 \bmod 2^u$.
   - $\mathcal{S}_{\mathsf{idx}}$ *corrupt:* Receive $(\vec{v}_{\mathsf{idx}}, \alpha_{\mathsf{idx}})$ from $\mathcal{S}_{\mathsf{idx}}$, then let $\vec{v}_{1-\mathsf{idx}} \leftarrow \vec{v} - \vec{v}_{\mathsf{idx}}, \alpha_{1-\mathsf{idx}} \leftarrow \alpha - \alpha_{\mathsf{idx}} \bmod 2^u$.

   Send $(\vec{v}_{\mathsf{idx}}, \alpha_{\mathsf{idx}})$ to $\mathcal{S}_{\mathsf{idx}}$.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{FSS-Gen}}$.

4.1 Definition

A property-based function secret sharing $\mathsf{PB\text{-}FSS}$ scheme consists of two algorithms $(\mathsf{Share}, \mathsf{Eval})$, and is defined over a family of sets $\mathcal{S} \subseteq 2^{\mathcal{D}}$ (where $\mathcal{D} = (\{0,1\}^u)^d$) and a property-testing function Prop. Looking ahead, the set family considered in this work contains d-dimensional balls with a center $\vec{q} \in \mathcal{D}$ and radius δ, under the L_∞ or L_p distance metric. Algorithm Share takes a set $S \in \mathcal{S}$ as input and outputs a key pair (k_0, k_1). Algorithm Eval takes a key k_{idx} and a point $\vec{x} \in \mathcal{D}$ as input, and outputs y_{idx}, for $\mathsf{idx} \in \{0,1\}$. The correctness guarantee is that testing $\mathsf{Prop}(y_0, y_1)$ outputs 1 if and only if $\vec{x} \in S$, and the privacy guarantee requires each key k_{idx} to be simulatable. We further extend this definition to the *incremental* setting, where evaluations can be computed on any prefix $\vec{x}$. In this setting, the property testing outputs 1 if and only if there exists a completion $\vec{\chi}$ of the prefix $\vec{x}$ that lies in S.

Definition 4 (Property-Based (Incremental) Function Secret Sharing)**.** *A two-party property-based (incremental) function secret sharing scheme* $\mathsf{PB\text{-}FSS}$ *(or* $\mathsf{PB\text{-}iFSS}$ *for incremental) for a family of sets* $\mathcal{S} \subseteq 2^{\mathcal{D}}$ *with* $\mathcal{D} = (\{0,1\}^u)^d$ *(or* $\mathcal{D} = (\{0,1\}^{\leq u})^d$ *for incremental), a property-testing function* Prop*, and payload group* $\mathbb{G}$*, is a pair of algorithms* $(\mathsf{Share}, \mathsf{Eval})$ *with the following syntax:*

- $(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, S)$*: The randomized share function takes as input the security parameter* κ *and (the description of) a set* $S \in \mathcal{S}$*, and outputs two key shares.*
- $y_{\mathsf{idx}} \leftarrow \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \vec{x})$*: The deterministic evaluation function takes as input a party index* $\mathsf{idx} \in \{0,1\}$*, the corresponding key share* k_{idx} *and input* $\vec{x} \in \mathcal{D}$*, and outputs a value* $y_{\mathsf{idx}} \in \mathbb{G}$*.*

The $\mathsf{PB\text{-}FSS}$ *(or* $\mathsf{PB\text{-}iFSS}$ *for incremental) scheme for* $\mathcal{S}$ *is secure if it satisfies the following conditions:*

Correctness for yes-instances. *For every security parameter* κ*, every (description of) set* $S \in \mathcal{S}$*, every input* $\vec{x} \in S$ *(for incremental: there exists* $\vec{\chi} \in S$ *such that* $\vec{x} \prec \vec{\chi}$*), the function* Prop *when*

evaluated on (y_0, y_1) *always outputs* 1*:*

$$\Pr\left[\begin{array}{l}\mathsf{Prop}(y_0,y_1)=1 \mid \\ \quad (k_0,k_1)\leftarrow \mathsf{Share}(1^\kappa,S),\\ \quad y_0\leftarrow \mathsf{Eval}(0,k_0,\vec{x}),\\ \quad y_1\leftarrow \mathsf{Eval}(1,k_1,\vec{x}).\end{array}\right]=1.$$

Correctness for no-instances: *For every input* $\vec{x}\in\mathcal{D}, \vec{x}\notin S$ *(for incremental: there does not exist* $\vec{\chi}\in S$ *such that* $\vec{x}\prec\vec{\chi}$*) the function* Prop *when evaluated on* (y_0, y_1) *outputs* 0*, except with negligible probability in the security parameter* λ*:*

$$\Pr\left[\begin{array}{l}\mathsf{Prop}(y_0,y_1)=0 \mid \\ \quad (k_0,k_1)\leftarrow \mathsf{Share}(1^\kappa,S),\\ \quad y_0\leftarrow \mathsf{Eval}(0,k_0,\vec{x}),\\ \quad y_1\leftarrow \mathsf{Eval}(1,k_1,\vec{x}).\end{array}\right]\geq 1-\mathsf{negl}(\lambda).$$

Privacy: *There exists a PPT simulator* Sim *such that for all* $\mathsf{idx}\in\{0,1\}$ *and all* $S\in\mathcal{S}$*, the following distributions are computationally indistinguishable in the security parameter* κ*:*

$$\boxed{\begin{array}{l}(k_0,k_1)\leftarrow \mathsf{Share}(1^\kappa,S)\\ \text{return } k_{\mathsf{idx}}\end{array}}\cong_\kappa \mathsf{Sim}(1^\kappa,\mathsf{idx},u,d).$$

In the following subsections, we present PB-iFSS constructions that support L_∞ and L_p distance metrics. We first define the corresponding property-testing functions in Section 4.2: **equality** for L_∞ and μ**-bounded** for L_p. Next, we describe two construction paradigms for PB-FSS (then extend to PB-iFSS): OKVS-based constructions in Section 4.3 and DCF-based constructions in Section 4.4.

Note that PB-iFSS is a more general primitive that subsumes PB-FSS. In particular, any PB-iFSS scheme can be viewed/used as a PB-FSS scheme by disabling evaluation on prefixes. Both of our construction paradigms start with PB-FSS and then extend to PB-iFSS with incremental evaluation.

4.2 Property-Testing Functions for L_∞ and L_p

Looking ahead, a client with point $\vec{q}$ generates a PB-FSS key pair (k_0, k_1) and distributes them among the two servers. This key pair allows the servers to evaluate on any point $\vec{x}$ and take the results as inputs to the next steps that determine whether $\mathsf{dist}(\vec{x},\vec{q})\leq\delta$. In this subsection, we define the set families and their corresponding property-testing functions for the two distance metrics.

L_∞ **distance metric.** Given a center $\vec{q}$ and a point $\vec{x}$, we test the *equality* property, namely, whether $\mathsf{Eval}(k_0,\vec{x})=\mathsf{Eval}(k_1,\vec{x})$, to determine whether $\vec{x}$ lies in the L_∞-ball with center $\vec{q}$ and radius δ or not. The one-dimensional case of this property says that the value x lying in the interval $[q-\delta, q+\delta]$ implies equal outputs from Eval. For the multi-dimensional case, we generate one PB-FSS key pair for each dimension $i\in[d]$, corresponding to each interval $[q_i-\delta, q_i+\delta]$, and let the evaluation result be the concatenation of Eval outputs in all dimensions. The correctness of this construction relies on the fact that $L_\infty(\vec{x},\vec{q})\leq\delta$ if and only if $|x_i-q_i|\leq\delta$ for all dimensions $i\in[d]$. We let $\mathcal{S}_{\infty,d,\delta}$ denote the set family for d-dimensional L_∞-balls of radius δ.

L_p **distance metric.** For the L_p distance, we construct PB-FSS such that the evaluation outputs are secret shares of $(L_p(\vec{x},\vec{q}))^p$, to be compared with $\mu=\delta^p$. We refer to this property as μ-bounded. Working with the p-th power of the L_p distance simplifies the extension from one dimension to multiple dimensions, as the outputs from all dimensions can be summed up. The construction's correctness relies on the fact that $(L_p(\vec{x},\vec{q}))^p=\sum_{i=1}^d|x_i-q_i|^p$. We let $\mathcal{S}_{p,d,\delta}$ denote the set family for d-dimensional L_p-balls of radius δ.

Definition 5 (Property-Testing Functions)**.** *We consider the following property-testing functions in our paper. These properties are defined for pairs of values in the* PB-FSS *output space* $\mathbb{G}$*.*

- $\mathsf{Equal}(y_0, y_1) = \begin{cases} 1 & \text{if } y_0 = y_1 \\ 0 & \text{otherwise} \end{cases}$.
- $\mathsf{Bounded}_\mu(y_0, y_1) = \begin{cases} 1 & \text{if } y_0 + y_1 \leq \mu \\ 0 & \text{otherwise} \end{cases}$.

4.3 OKVS-Based Construction of PB-iFSS

Construction Overview. Recall that OKVS encodes a list of (key, value) pairs into a data structure, which, when decoded at one of these keys, outputs its corresponding value, and otherwise outputs a random value. In addition, the OKVS encoding hides which keys are encoded. The high-level idea in this construction is to generate two OKVS encodings as the two PB-FSS key shares. Each OKVS encoding uses nearby points of the target point $\vec{q}$ as keys, and uses secret shares of either 0 (for L_∞) or distance to $\vec{q}$ (for L_p) as their associated values. However, this idea would result in the PB-FSS key size growing with $O(\delta^d)$. To reduce the key size, we instead construct OKVS encodings for each dimension and aggregate the results in all dimensions, bringing the key size down to $O(\delta \cdot d)$.

$\underline{\mathsf{Share}(1^\kappa, \mathsf{dist}, (q_1, \ldots, q_d), \delta, D, \mathbb{G}) :}$

1. For each $i = 1, \ldots, d$:
   - Prepare a set of keys $K^{(i)} \leftarrow \{x : q_i - \delta \leq x \leq q_i + \delta\}$.
   - For each key $x \in K^{(i)}$, prepare a random pair $v_{x,0}, v_{x,1}$ such that:
   $$v_{x,0} - v_{x,1} = \begin{cases} 0 & \text{if } \mathsf{dist} = L_\infty, \\ |x - q_i|^p & \text{if } \mathsf{dist} = L_p. \end{cases}$$
   - Prepare two lists of key-value pairs, each containing subtractive shares for one server:
   $$KV^{(i)}_{\mathsf{idx}} \leftarrow \left\{ (x, v_{x,\mathsf{idx}} + H_{\mathsf{idx}}(x)) \mid x \in K^{(i)} \right\}. \tag{7}$$
   - Obtain $k^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{OKVS.Encode}(KV^{(i)}_{\mathsf{idx}})$ for $\mathsf{idx} = 0, 1$.
2. **Output:** $k_{\mathsf{idx}} = (k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}})$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, \mathsf{dist}, k_{\mathsf{idx}}, (x_1, \ldots, x_d)):}$

1. Parse key $\left(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}\right) \leftarrow k_{\mathsf{idx}}$.
2. For each $i = 1, \ldots, d$:
   - Decode the OKVS vector $k^{(i)}_{\mathsf{idx}}$ on x_i and obtain:
   $$y^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{OKVS.Decode}(k^{(i)}_{\mathsf{idx}}, x_i) - H_{\mathsf{idx}}(x_i).$$
3. **Output:** $y_{\mathsf{idx}} = \begin{cases} \left(y^{(1)}_{\mathsf{idx}} || \ldots || y^{(d)}_{\mathsf{idx}}\right) & \text{if } \mathsf{dist} = L_\infty, \\ (-1)^{\mathsf{idx}} \sum_{i=1}^{d} y^{(i)}_{\mathsf{idx}} & \text{if } \mathsf{dist} = L_p. \end{cases}$

Figure 4: OKVS-based PB-FSS scheme (edge cases omitted).

In more detail, the key pairs $k_0^{(i)}, k_1^{(i)}$ for each dimension i are outputs of the $\mathsf{OKVS.Encode}$ algorithm. Our task is to choose the lists of (key, value) pairs $KV_0^{(i)}, KV_1^{(i)}$ appropriately depending on the distance metric (L_∞ or L_p), before obtaining $k_{\mathsf{idx}}^{(i)} \leftarrow \mathsf{OKVS.Encode}(KV_{\mathsf{idx}}^{(i)})$ for $\mathsf{idx} \in \{0,1\}$ as keys. A party idx can evaluate a point $\vec{x}$ by concatenating or summing the evaluation results $\mathsf{OKVS.Decode}(k_{\mathsf{idx}}^{(i)}, x_i)$ of each coordinate. For L_∞ distance, we want the decoding outputs to be equal: $\mathsf{OKVS.Decode}(k_0^{(i)}, x_i) = \mathsf{OKVS.Decode}(k_1^{(i)}, x_i)$ for values x_i in the interval $[q_i - \delta, q_i + \delta]$. For L_p distance, we want the decoding outputs for a value x_i to be secret shares of its one-dimensional L_p-distance from q_i, that is, $\mathsf{OKVS.Decode}(k_0^{(i)}, x_i) - \mathsf{OKVS.Decode}(k_1^{(i)}, x_i) = |x_i - q_i|^p$. When the servers aggregate the results across all dimensions, they get the secret shares of $(L_p(\vec{q}, \vec{x}))^p$.

In fact, for both L_∞ and L_p distance, the set $KV_{\mathsf{idx}}^{(i)}$ we choose contains a (key, value) pair (x, v) if and only if x lies in the interval $[q_i - \delta, q_i + \delta]$. Intuitively, this allows controlling the $\mathsf{OKVS.Decode}$ outputs for all values in this interval, ensuring perfect correctness for yes-instances. The outputs for values more than δ away from q_i would be random (following the Strong Extraction Conjecture), which would make the probability for correct no-instance overwhelming with large enough output range.

L_∞ distance metric. In dimension i, as expected, the encoding algorithm returns $k_0^{(i)}$ and $k_1^{(i)}$ such that decoding these on keys $x \in [q_i - \delta, q_i + \delta]$ returns two equal results. More specifically, for each $x \in [q_i - \delta, q_i + \delta]$, an attempt would be to set the two corresponding key-value pairs $(x, v_{x,0}) \in KV_0^{(i)}$ and $(x, v_{x,1}) \in KV_1^{(i)}$ such that $v_{x,0} = v_{x,1}$. However, in this attempt, for each dimension i the $\mathsf{OKVS.Encode}$ algorithm turns out to encode the same two sets $KV_0^{(i)} = KV_1^{(i)}$, which would result in the same output! The decoding results for values outside the interval $[q_i - \delta, q_i + \delta]$ would then also be equal, which violates the correctness for no-instances. We fix this issue by breaking the linear correlation of the two key value sets and instead storing $(x, v_{x,0} + H_0(x))$ in $KV_0^{(i)}$ and $(x, v_{x,1} + H_1(x))$ in $KV_1^{(i)}$, where $v_{x,0} = v_{x,1}$ and $H_0(\cdot), H_1(\cdot)$ are public hash functions.[1] The encoding algorithm would then return two vectors $k_0^{(i)}$ and $k_1^{(i)}$ that are not linearly related, and party idx can obtain Eval result for each dimension to be $\mathsf{OKVS.Decode}(k_{\mathsf{idx}}^{(i)}, x_i) - H_{\mathsf{idx}}(x_i)$, still guaranteeing equal evaluations for x_i inside the interval.

L_p distance metric. For this distance metric, decoding a key $x \in [q_i - \delta, q_i + \delta]$ with $k_0^{(i)}$ and $k_1^{(i)}$ should return secret shares of $|x - q_i|^p$. This is almost what we set as values; we can store $(x, v_{x,0}) \in KV_0^{(i)}$ and $(x, v_{x,1}) \in KV_1^{(i)}$ such that $v_{x,0} + v_{x,1} = |x - q_i|^p$. Note that one should also randomize the evaluation results outside the interval, again, by storing $(x, v_{x,0} + H_0(x)) \in KV_0^{(i)}$ and $(x, v_{x,1} + H_1(x)) \in KV_1^{(i)}$ instead.

Supporting incremental evaluation. To support incremental evaluation, we further include (key, value) pairs for each prefix length in OKVS. Specifically, if there exists χ inside the interval $[q_i - \delta, q_i + \delta]$ that a has prefix x, we include x as key in $KV_0^{(i)}$ and $KV_1^{(i)}$. The values for the L_∞ distance metric are set similarly as randomization of two equal values. The values for the L_p case are trickier, where we choose a completion χ for prefix x such that $|\chi - q_i|^p$ is minimized, and include secret shares of this formula as values.

A description that captures the essences of our OKVS-based construction (for L_∞ and L_p) in the non-incremental case, while ignoring the edge case when a client's input might be less than δ, is shown in Figure 4. We refer to Appendix A for the full construction of OKVS-based $\mathsf{PB\text{-}iFSS}$ and its proofs.

4.4 DCF-Based Construction of PB-iFSS

Construction Overview. To construct $\mathsf{PB\text{-}FSS}$ for the target point $\vec{q}$, for each dimension i we "secret share" one function $f_i(x)$ depending on the value of q_i and the distance metric. When evaluating the

[1] For implementation, we use $(x, v_{x,0} + H_0(x))$ and $(x, v_{x,1})$ with a single hash H_0 instead, but we describe both hash functions here for symmetry.

function shares on the same input $x \in [q_i - \delta, q_i + \delta]$, the two servers obtain secret shares of $f_i(x)$, which should either be equal (for L_∞) or be secret shares of $|x - q_i|^p$ (for L_p).

Our construction for non-incremental PB-FSS achieves sharing of the described functions by combining multiple Distributed Comparison Functions (DCFs), which is a special FSS for comparison functions. Constructions for DCF have been well known in the literature [BGI16, GGM24], achieving key size $O(u \cdot \kappa)$ given the input domain being $\mathbb{Z}_{2^u}$. In the description of our constructions for L_∞ and L_p that follow, we use DCF in a black-box manner.

L_∞ distance metric. For dimension i, we secret share the interval $[q_i - \delta, q_i + \delta]$ with two DCFs whose breaking points are $q_i - \delta - 1$ and $q_i + \delta$, respectively. The function being shared is then $f_i(x) = f^C_{q_i+\delta,\alpha_1,\beta_1}(x) - f^C_{q_i-\delta-1,\alpha_0,\beta_0}(x)$, where the payloads α_0, β_0 and α_1, β_1 are chosen so that $f_i(x)$ returns the correct output on each segment of the domain:

$$f_i(x) = \begin{cases} \alpha_1 - \alpha_0 & \text{if } x < q_i - \delta. \\ \alpha_1 - \beta_0 & \text{if } q_i - \delta \le x \le q_i + \delta, \\ \beta_1 - \beta_0 & \text{if } x > q_i + \delta. \end{cases}$$

For $f_i(x)$ to return 0 only when x is inside the interval $[q_i - \delta, q_i + \delta]$, we set $\alpha_1 - \alpha_0 = 1$, $\alpha_1 - \beta_0 = 0$, and $\beta_1 - \beta_0 = 1$. The core idea of our construction is shown in Figure 5, omitting the corner cases such as when $q_i < \delta$.

$\underline{\mathsf{Share}(1^\kappa, L_\infty, (q_1, \ldots, q_d), \delta, \mathbb{G}) :}$

1. For each $i = 1, \ldots, d$:
   - Generate DCF keys to share the interval $[q_i - \delta, q_i + \delta]$:
   $$(k^{(i)}_{0,0}, k^{(i)}_{1,0}) \leftarrow \mathsf{DCF.Share}(q_i - \delta - 1, 1, 0), \tag{8}$$
   $$(k^{(i)}_{0,1}, k^{(i)}_{1,1}) \leftarrow \mathsf{DCF.Share}(q_i + \delta, 0, 1), \tag{9}$$
   - Obtain $k^{(i)}_{\mathsf{idx}} \leftarrow \left(k^{(i)}_{\mathsf{idx},0} \| k^{(i)}_{\mathsf{idx},1}\right)$ for $\mathsf{idx} = 0, 1$.
2. **Output:** $k_{\mathsf{idx}} = \left(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}\right)$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, L_\infty, k_{\mathsf{idx}}, (x_1, \ldots, x_d)):}$

1. Parse key $(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}) \leftarrow k_{\mathsf{idx}}$.
2. For each $i = 1, \ldots, d$:
   - Parse $\left(k^{(i)}_{\mathsf{idx},0} \| k^{(i)}_{\mathsf{idx},1}\right) \leftarrow k^{(i)}_{\mathsf{idx}}$.
   - Evaluate the two DCF keys on x_i then combine them to obtain:
   $$y^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{DCF.Eval}(\mathsf{idx}, k^{(i)}_{\mathsf{idx},0}, x_i) \oplus \mathsf{DCF.Eval}(\mathsf{idx}, k^{(i)}_{\mathsf{idx},1}, x_i).$$
3. **Output:** $y_{\mathsf{idx}} = \left(y^{(1)}_{\mathsf{idx}} \| \ldots \| y^{(d)}_{\mathsf{idx}}\right)$.

Figure 5: DCF-based PB-FSS: L_∞ scheme (edge cases omitted).

L_p **distance metric.** In dimension i, to share the function $|x - q_i|^p$, we use $p+1$ DCF key pairs with the same breaking point being q_i, and payloads being essentially powers of q_i. Using DCF in this construction helps split the domain into two halves, since the L_p distance of x from q_i is calculated differently when $x \leq q_i$ (returning $(q_i - x)^p$) and when $x > q_i$ (returning $(x - q_i)^p$). Both expressions can be decomposed into $(p+1)$ parts using the *binomial expansion*, for example:

$$(q_i - x)^p = q_i^p - pq_i^{p-1}x + \binom{p}{2} q_i^{p-2}x^2 - \ldots + (-1)^p x^p.$$

During evaluation, we observe that both servers know the same evaluation point x, hence all of its powers $x^2, x^3, \ldots$ as well. Multiplying these powers by the secret-shared DCF payloads is therefore a local operation and essentially free. The client only needs to set the other components in the binomial expansion as the DCF payloads, namely q_i^p, $-pq_i^{p-1}$, ..., $(-1)^p$ for $x \leq q_i$, and $(-q_i)^p$, $p(-q_i)^{p-1}$, ..., 1 for $x > q_i$.

One caveat for the correctness in the L_p distance metric is that we need to upper bound the evaluation results on *every* point in the domain by a value, which we chose to be $\delta^p + 1$. The solution is simple: we introduce two more sets of $p+1$ DCF key pairs in each dimension, one helps setting the payload when $x < q_i - \delta$ to be always $\delta^p + 1$, and the other one helps setting the payload when $x > q_i + \delta$ to be also $\delta^p + 1$. In our description for the L_p construction (Figure 6), we omit this detail about upper-bounding, and defer it to Appendix B.2.

Supporting incremental evaluation. We support incremental evaluation by controlling the DCF payload for each *prefix* in the domain evaluated. Recall that for the L_∞ distance metric, the PB-FSS construction for the i-th dimension combines two different DCFs for one-sided intervals with their intersection being $[q_i - \delta, q_i + \delta]$. More specifically, the two one-sided intervals are $[0, q_i + \delta]$ (a *left-sided* interval), and $[q_i - \delta, 2^{h_1} - 1]$ (a *right-sided* interval). For the left-sided interval $[0, q_i + \delta]$, the optimistic property for PB-iFSS requires *any* prefix x with a completion $\chi \leq q_i + \delta$ to have a fixed payload, which is the reason we introduce the *left-side* variant LDCF of DCF: Our PB-iFSS construction for L_∞ now combines an LDCF with an RDCF for each dimension instead. The incremental construction for L_p, although more involved, is also essentially a combination of multiple LDCFs and RDCFs. To implement LDCF and RDCF, we modify the GGM-based construction for DCF in a non-black-box way, with details deferred to Appendix B.

5 Private Property Testing

In the *Fuzzy Matching* phase, the two servers jointly identify the heavy hitters from the client data that are collected in the upload phase using PB-FSS as described in Section 4. The fuzzy matching phase proceeds in two steps:

1. **Private Property Testing:** The servers securely check whether their local evaluations, y_0 and y_1, of the client's PB-FSS shares satisfy the required property (e.g., Equal or $\mathsf{Bounded}_\mu$). This step yields secret shares of an indicator $b \in \{0, 1\}$, where $b = 1$ implies the client data $\vec{q}$ satisfies the distance condition, namely $\mathrm{dist}(\vec{x}, \vec{q}) \leq \delta$ for some candidate item $\vec{x}$.
2. **Private Threshold Comparison:** The servers sum up their secret shares of the indicators for all clients and jointly check if the total count is at least the threshold t.

In this section, we detail the Private Property Testing step (ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ formalized in Figure 7), and defer the Private Threshold Comparison step (ideal functionality $\mathcal{F}_{\mathsf{Comp}}$) to the full version.

Construction Overview. We present two approaches to Private Property Testing. The first approach is based on generic secure two-party computation [Yao86] (see Definition 1). This approach is largely straightforward. One subtlety arises in the L_p case, where y_0 and y_1 are secret shares over the ring $\mathbb{Z}_{2^{h_2}}$.

$\underline{\mathsf{Share}(1^\kappa, L_p, (q_1, \ldots, q_d), \delta, \mathbb{G}) :}$

1. For each $i = 1, \ldots, d$:

   - Create two vectors of payloads to help evaluating $|x - q_i|^p$:

$$\vec{\alpha}^{(i)} = \begin{bmatrix}(-1)^p & (-1)^{p-1} q_i & \cdots & (-1)^0 q_i^p\end{bmatrix}$$
$$\vec{\beta}^{(i)} = \begin{bmatrix}(-1)^0 & (-1)^1 q_i & \cdots & (-1)^p q_i^p\end{bmatrix}$$

   - Create a pair of DCF keys with the two vectors of payloads:

$$(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{DCF.Share}\left(q_i, \vec{\alpha}^{(i)}, \vec{\beta}^{(i)}\right). \tag{10}$$

2. **Output:** $k_{\mathsf{idx}} = (k_{\mathsf{idx}}^{(1)} \| \ldots \| k_{\mathsf{idx}}^{(d)})$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, L_p, k_{\mathsf{idx}}, (x_1, \ldots, x_d)):}$

1. Parse key $(k_{\mathsf{idx}}^{(1)} \| \ldots \| k_{\mathsf{idx}}^{(d)}) \leftarrow k_{\mathsf{idx}}$.
2. For each $i = 1, \ldots, d$:

   - Evaluate the DCF key $k_{\mathsf{idx}}^{(i)}$ on x_i then obtain a vector of payloads:

$$\begin{bmatrix}y_0^{(i)} & y_1^{(i)} & \ldots & y_p^{(i)}\end{bmatrix} \leftarrow \mathsf{DCF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}^{(i)}, x_i).$$

   - Combine the payloads using binomial coefficients:

$$y_{\mathsf{idx}}^{(i)} \leftarrow \sum_{j=0}^{p} \binom{p}{j} x_i^{p-j} y_j^{(i)}.$$

3. **Output:** $y_{\mathsf{idx}} = \sum_{i=1}^{d} y_{\mathsf{idx}}^{(i)}$.

Figure 6: DCF-based $\mathsf{PB\text{-}FSS}$: L_p scheme (edge cases omitted).

The property testing over the ring becomes whether $y_0 + y_1 \bmod 2^{h_2} \leq \mu$, which requires careful handling of the overflow case when $y_0 + y_1 \geq 2^{h_2}$.

Our second approach introduces a trusted dealer who generates correlated randomness for the two servers, which is independent of the input data. At a high level, the trusted dealer generates (masked) FSS keys for DPF (Distributed Point Function) or DCF (Distributed Comparison Function, described in Section 4.4), which are then used by the two servers to perform Equal or $\mathsf{Bounded}_\mu$ testing with much less communication than generic 2PC. Here again, we need careful handling of the overflow cases. This approach is inspired by prior work on FSS-enhanced 2PC [BGI19]. In practice, the trusted dealer can be instantiated as another non-colluding server or a trusted execution environment (TEE). Alternatively, the correlated randomness can be produced via another 2PC protocol between the two non-colluding servers in the offline phase, following the offline/online paradigm.

5.1 2PC-Based Approach for L_∞ Distance

The goal of this protocol is to securely check whether the secret shares y_0 and y_1 are equal. We instantiate this equality check using garbled circuits [Yao86], which allows the servers to evaluate the function within a constant number of rounds. Specifically, S_0 and S_1 provide their shares y_0 and y_1 as inputs to a Boolean circuit that compares y_0 and y_1 and outputs secret shares of 1 if they are equal and 0 otherwise. The

Parameters:

- Input share length h_2.
- Output share length h_3.
- Property type: Equal or $\mathsf{Bounded}_\mu$.

Functionality:

1. Receive $y_0 \in \{0,1\}^{h_2}$ from S_0 and $y_1 \in \{0,1\}^{h_2}$ from S_1.
2. Compute b depending on property type:
   - Equal: If $y_0 = y_1$, set $b = 1$, else set $b = 0$.
   - $\mathsf{Bounded}_\mu$: Compute $y = y_0 + y_1 \pmod{2^{h_2}}$. Set $b = 1$ if $y \leq \mu$, else $b = 0$.
3. If S_{idx} is corrupt, receive output share b_{idx} from the adversary and compute $b_{1-\mathsf{idx}} = b - b_{\mathsf{idx}} \pmod{2^{h_3}}$.
4. If both parties are honest, sample $b_0 \leftarrow \mathbb{Z}_{2^{h_3}}$ uniformly at random and compute $b_1 = b - b_0 \pmod{2^{h_3}}$.
5. Send b_0 to S_0 and b_1 to S_1.

Figure 7: Ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$.

detailed protocol $\Pi_{\mathsf{Equal\text{-}2PC}}$ is presented in Figure 8, and the security proof is deferred to Appendix C.1.

Inputs:

- Party S_0 holds input share $y_0 \in \{0,1\}^{h_2}$.
- Party S_1 holds input share $y_1 \in \{0,1\}^{h_2}$.

Circuit:
The protocol uses the following circuit C:

$$C(y_0, y_1) = \mathbb{1}_{y_0 = y_1}.$$

Protocol:

1. S_0 and S_1 invoke the ideal functionality $\mathcal{F}_{\mathsf{2PC}}$ for circuit C, where S_0 provides input y_0 and S_1 provides input y_1.
2. The functionality returns arithmetic shares (b_0, b_1) of the output bit such that $C(y_0, y_1) = b_0 + b_1 \pmod{2^{h_3}}$.

Figure 8: Protocol $\Pi_{\mathsf{Equal\text{-}2PC}}$ for $\mathcal{F}_{\mathsf{PropTest}}$.

Intuition for $\mathsf{Bounded}_\mu$ property testing. For the $\mathsf{Bounded}_\mu$ property testing, the parties hold additive shares y_0, y_1 and wish to test whether the reconstructed value satisfies $y \leq \mu$, where $y = y_0 + y_1 \pmod{2^{h_2}}$. This comparison is complicated by potential modular overflow. If there is no overflow (i.e., the integer sum $y_0 + y_1 < 2^{h_2}$), the condition $y \leq \mu$ is simply $y_0 + y_1 \leq \mu$. However, if there is an overflow ($y_0 + y_1 \geq 2^{h_2}$), then $y_0 + y_1 = y + 2^{h_2}$. In this case, the condition $y \leq \mu$ is equivalent to checking $y_0 + y_1 \leq \mu + 2^{h_2}$. We

handle these cases securely using a comparison 2PC. The formal protocol description and security proof are deferred to Appendix C.2

5.2 Dealer-Based Approach for L_∞ Distance

We propose an alternative solution using a third-party trusted dealer (or an equivalent preprocessing phase). This solution moves the computationally intensive generation of cryptographic keys to an offline phase, leaving only lightweight operations for the online interaction. The core challenge is checking the condition $y_0 = y_1$ without revealing y_0 and y_1 in the clear. To address this, we employ a *random masking* technique that works effectively over the ring $\mathbb{Z}_{2^{h_2}}$. The protocol proceeds as follows:

1. **Offline Key Generation:** The dealer generates a random mask r and a pair of DPF keys for the point function $f^P_{r,1,0}(\cdot)$, which evaluates to 1 *only* when the input is exactly the mask r. Looking forward, this means when there is equality, one count of fuzzy match is added to the aggregated sum.
2. **Online Evaluation:** During the online phase, the servers mask their local shares (computing secret shares of $\tilde{y} = y_0 - y_1 + r \mod 2^{h_2}$) and exchange these values to reconstruct the global masked value $\tilde{y}$. A key challenge in secret-shared comparisons is the modular overflow (wrapping around 2^{h_2}). However, for this *equality* check, the modular ring structure naturally preserves the condition: $y_0 = y_1 \iff \tilde{y} = r \pmod{2^{h_2}}$.

Inputs:

- Party S_0 holds input share $y_0 \in \mathbb{Z}_{2^{h_2}}$.
- Party S_1 holds input share $y_1 \in \mathbb{Z}_{2^{h_2}}$.
- Dealer D.

Protocol:

1. **Preprocessing:** Dealer D samples a random mask $r \leftarrow \mathbb{Z}_{2^{h_2}}$. D generates DPF keys $(k_0, k_1) \leftarrow \mathsf{DPF.Share}(r, 1)$. D creates additive shares of r such that $r = r_0 - r_1 \pmod{2^{h_2}}$. D sends (k_0, r_0) to S_0 and (k_1, r_1) to S_1.
2. **Online Masking:** S_0 computes and sends $\tilde{y}_0 = y_0 + r_0 \pmod{2^{h_2}}$ to S_1. S_1 computes and sends $\tilde{y}_1 = y_1 + r_1 \pmod{2^{h_2}}$ to S_0.
3. **Evaluation:** Both parties reconstruct the masked value:

$$\tilde{y} = \tilde{y}_0 - \tilde{y}_1 = (y_0 - y_1) + (r_0 - r_1) = r \pmod{2^{h_2}}.$$

For $\mathsf{idx} \in \{0, 1\}$, $\mathcal{S}_{\mathsf{idx}}$ outputs: $b_{\mathsf{idx}} = \mathsf{DPF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \tilde{y})$.

Figure 9: Protocol $\Pi_{\mathsf{Equal\text{-}Dealer}}$ for $\mathcal{F}_{\mathsf{PropTest}}$

The protocol for Equality property testing is presented in Figure 9, and its security proof is deferred to Appendix C.3.

Intuition for $\mathsf{Bounded}_\mu$ property testing. We can use FSS with additive masks to check the $\mathsf{Bounded}_\mu$ property. Similar to the 2PC approach, this requires handling modular overflows by checking conditions on the masked values. For example, to check whether two values y_0 and y_1 add up to a value y (assuming no overflow) that is less than or equal to μ, one can prepare a (shared) random mask r and a DCF with breaking point being $r+\mu$, i.e., sharing $f^C_{r+\mu,1,0}(x)$. Then both parties can jointly open $y+r$ and evaluate $f^C(r+\mu, 1, 0)(y+r)$, to test whether $y+r \leq r+\mu$, which is equivalent to $y \leq \mu$. Our detailed description and proof for the protocol are deferred to Appendix C.4.

6 Putting It All Together

In this section, we formally define the ideal functionalities for the fuzzy heavy hitters problem in the known and unknown dictionary settings in Figure 10 and Figure 11, respectively. We then present the details of the Mosaic framework in Figure 12 that depicts the Client Upload Phase and Fuzzy Matching Phase (via Private Property Testing and Threshold Comparison). Theorem 1 (for the known dictionary setting) and Theorem 2 formally state the security of our full protocol. We defer the proof of security, as well as some nuances and trade-offs when choosing different approaches in each step, to Appendix E. We also defer the full details of the threshold comparison step to Appendix D. due to the similarity with the property testing step.

Theorem 1. *For a distance metric* dist *with corresponding set family and property* $(\mathcal{S}, \mathsf{Prop})$*, given* PB-FSS *scheme for* $\mathcal{S}$ *and* Prop*, the protocol in Figure* 12 *realizes* $\mathcal{F}_{\mathsf{known}}$ *(Figure 10) in the* $\mathcal{F}_{\mathsf{PropTest}}$ *and* $\mathcal{F}_{\mathsf{Comp}}$ *hybrid model, in the presence of semi-honest adversaries.*

Theorem 2. *For a distance metric* dist *with corresponding set family and property* $(\mathcal{S}, \mathsf{Prop})$*, given* PB-iFSS *scheme for* $\mathcal{S}$ *and* Prop*, the protocol in Figure* 12 *realizes* $\mathcal{F}_{\mathsf{unknown}}$ *(Figure 11) in the* $\mathcal{F}_{\mathsf{PropTest}}$ *and* $\mathcal{F}_{\mathsf{Comp}}$ *hybrid model, in the presence of semi-honest adversaries.*

Parameters: distance metric dist, similarity threshold δ, heavy hitter threshold t, dimensions d, input string length h_1.
Participants: Two servers $\mathcal{S}_0, \mathcal{S}_1$ and a set of N clients $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_N)$.
Functionality:
1. Receive from both servers $\mathcal{S}_0, \mathcal{S}_1$ a dictionary $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_M\}$ of input vectors, where $\vec{x}_i \in (\{0,1\}^{h_1})^d$ for all $i \in [M]$.
2. Receive from each client $\mathcal{C}_i \in \mathcal{C}$ input vector $\vec{q}_i \in (\{0,1\}^{h_1})^d$.
3. Send output list of heavy hitters to both servers $\mathcal{S}_0$ and $\mathcal{S}_1$:

$$H = \{\vec{x}_i \mid i \in [M] \ \wedge \ |\{\vec{q}_j \mid j \in [N] \ \wedge \ \mathsf{dist}(\vec{x}_i, \vec{q}_j) \leq \delta\}| \geq t\}.$$

Figure 10: Ideal functionality $\mathcal{F}_{\mathsf{known}}$ for the known-dictionary fuzzy heavy hitters problem.

Parameters: distance metric dist, similarity threshold δ, heavy hitter threshold t, input string length h_1, dimensions d
Participants: Two servers $\mathcal{S}_0, \mathcal{S}_1$ and a set of N clients $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_N)$
Functionality:
1. Receive from each client $\mathcal{C}_i \in \mathcal{C}$ input vector $\vec{q}_i \in (\{0,1\}^{h_1})^d$.
2. Compute

$$H = \left\{ \vec{w} \;\middle|\; \begin{array}{l} \vec{w} \in (\{0,1\}^{h_1})^d \\ \wedge \left|\{\vec{q}_j | j \in [N] \wedge \mathsf{dist}(\vec{w}, \vec{q}_j) \leq \delta\}\right| \geq t \end{array} \right\}.$$

3. Compute leakage $\mathcal{L}$ as the set of all d-dimensional prefixes $\vec{x} \in (\{0,1\}^{\leq h_1})^d$ that were explored during the search and did not meet the threshold:

$$\mathcal{L} = \left\{ \vec{x} \;\middle|\; \begin{array}{l} |\{\vec{q}_j \mid j \in [N],\ \mathsf{dist}_{\mathrm{opt}}(\vec{x}, \vec{q}_j) \leq \delta\}| \geq t, \\ \text{and } \nexists \vec{\chi} \succ \vec{x} \text{ such that} \\ \quad |\{\vec{q}_j \mid j \in [N],\ \mathsf{dist}(\vec{w}, \vec{q}_j) \leq \delta\}| \geq t \end{array} \right\}.$$

   where $\mathsf{dist}_{\mathrm{opt}}$ denotes the optimistic distance between the prefix $\vec{x}$ and the full vector $\vec{q}_j$ defined in Section 3.
4. **[output]** Send output list of heavy hitters H to both servers $\mathcal{S}_0$ and $\mathcal{S}_1$.

Figure 11: Ideal functionality $\mathcal{F}_{\mathsf{unknown}}$ for the unknown-dictionary fuzzy heavy hitters problem.

Inputs:

- Servers $\mathcal{S}_0$ and $\mathcal{S}_1$. If $\mathsf{case} = \mathsf{known}$, each server also holds a dictionary $W \subseteq (\{0,1\}^{h_1})^d$ of input points.
- Set of N clients $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N)$, where each client $\mathcal{C}_i$ has an input point $\vec{q}_i \in (\{0,1\}^{h_1})^d$.

Client Uploading: Every client $\mathcal{C}_i \in \mathcal{C}$ does the following:

- Let $D = (\{0,1\}^{h_1})^d$, and let $\mathbb{G}$ be $\{0,1\}^{h_2 d}$ for $\mathsf{dist} = L_\infty$ and being $\mathbb{Z}_{2^{h_2}}$ for $\mathsf{dist} = L_p$. Generate PB-FSS (or PB-iFSS) key pairs:

  If $\mathsf{case} = \mathsf{known}$:

  $$(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Share}(1^\kappa, \mathsf{dist}, \vec{q}_i, \delta, D, \mathbb{G}).$$

  If $\mathsf{case} = \mathsf{unknown}$:

  $$(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Share}(1^\kappa, \mathsf{dist}, \vec{q}_i, \delta, D, \mathbb{G}).$$

- Client sends $k_0^{(i)}$ to $\mathcal{S}_0$ and $k_1^{(i)}$ to $\mathcal{S}_1$.

Subroutine Fuzzy Matching:

- Both servers have as input a prefix $\vec{x} \in (\{0,1\}^{\leq h_1})^d$, and each server $\mathcal{S}_b$ holds a set of keys $(k_b^{(1)}, k_b^{(2)}, \ldots, k_b^{(N)})$.
- **Property Testing:** For each client index $i \in [N]$:
  1. $\mathcal{S}_0$ computes $y_{i,0} \leftarrow \mathsf{Eval}(0, \mathsf{dist}, k_0^{(i)}, \vec{x})$ and $\mathcal{S}_1$ computes $y_{i,1} \leftarrow \mathsf{Eval}(1, \mathsf{dist}, k_1^{(i)}, \vec{x})$.
  2. Both servers invoke $\mathcal{F}_{\mathsf{PropTest}}$ with inputs $y_{i,0}$ from $\mathcal{S}_0$ and $y_{i,1}$ from $\mathcal{S}_1$. As output, $\mathcal{S}_0$ gets $[\![b_i]\!]_0$ and $\mathcal{S}_1$ gets $[\![b_i]\!]_1$.
- **Threshold Comparison:**
  1. Both servers obtain secret shares of the aggregation: $\mathcal{S}_0$ gets $[\![s]\!]_0 \leftarrow \sum_{i=1}^N [\![b_i]\!]_0$ and $\mathcal{S}_1$ gets $[\![s]\!]_1 \leftarrow \sum_{i=1}^N [\![b_i]\!]_1$.
  2. Both servers invoke $\mathcal{F}_{\mathsf{comp}}$ with inputs $[\![s]\!]_0$ from $\mathcal{S}_0$ and $[\![s]\!]_1$ from $\mathcal{S}_1$, and threshold t.
- Both servers obtain output $o_{\vec{x}}$ from $\mathcal{F}_{\mathsf{comp}}$, which is also the output of the subroutine.

Server protocol for known dictionary:

- Both servers initialize an empty set of heavy hitters $H \leftarrow \emptyset$.
- For each point $\vec{w} \in \mathcal{D}$, both servers run the Fuzzy Matching Subroutine and get an output $o_{\vec{w}}$. If $o_{\vec{w}} = 1$, update $H \leftarrow H \cup \{\vec{w}\}$.
- Both servers output H.

Server protocol for unknown dictionary:

- Both servers initialize $H \leftarrow \{(\varepsilon, \varepsilon, \ldots, \varepsilon)\}$.
- Iterate through each prefix length $u = 1, \ldots, h_1$: Iterate through each dimension $i = 1, \ldots, d$:
  1. Initialize an empty set H_{new}.
  2. For each prefix $\vec{x} = (x_1, \ldots, x_d)$ in the current set H:
     - Run the Fuzzy Matching Subroutine for $\vec{x}_0 = (x_1, \ldots, x_{i-1}, x_i \| 0, x_{i+1}, \ldots, x_d)$. Informally, $\vec{x}_0$ is obtained by extending the i-th entry of $\vec{x}$ with a zero-bit. If the subroutine's output $o_{\vec{x}_0}$ is 1, add $\vec{x}_0$ to the new set H_{new}.
     - Run the Fuzzy Matching Subroutine for $\vec{x}_1 = (x_1, \ldots, x_{i-1}, x_i \| 1, x_{i+1}, \ldots, x_d)$. Informally, $\vec{x}_1$ is obtained by extending the i-th entry of $\vec{x}$ with a one-bit. If the subroutine's output $o_{\vec{x}_1}$ is 1, add $\vec{x}_1$ to the new set H_{new}.
  3. Reassign $H \leftarrow H_{\text{new}}$.
- Both servers output H.

Figure 12: The Mosaic framework for Private Fuzzy Heavy Hitters.

7 Sketching for DCF-based PB-FSS

In addition to protecting honest clients, our protocol can also support detecting malformed input from malicious clients through sketching. We introduce new sketching techniques for the DCF-based construc-

tion of PB-FSS: sketching for incremental evaluation and sketching for DCF with payload being a function of the breaking point itself. Our work requires a separate sketching for incremental evaluation due to the fact that we cannot guarantee that previous layers of the incremental DCF form a correctness DCF even if we check the last layer in full. We notice that even in prior constructions such as Extractable DPF [BBCG$^+$21], this issue still occurs: even though the incremental DPF construction in [BBCG$^+$21] guarantees that each layer has at most one potentially non-zero entry, this entry can still be set to be 0 by a malicious client. Intuitively, we construct DCF by taking the sum of evaluation results of all prefix of a string, hence letting the client to control prefix payload freely (setting them to either 0 or 1 by their choice) is disastrous. The second innovation for sketching DCF with more sophisticated payload is due to the PB-FSS construction for L_p, where e.g. a client needs to share a DCF that evaluates into q^2 up until a point q.

Additionally, we made an important optimization that allows sketching for DCF with payloads being ring elements $\mathbb{Z}_{2^k}$, which is necessary if the consequent steps using DCF outputs are more efficient in $\mathbb{Z}_{2^k}$. Sketching for $\mathbb{Z}_{2^k}$ outputs is not easy, since elements in this ring are not all invertible. We first convert the DCF full domain evaluation into secret shares of a unit vector, then use the fact that only one entry in the vector would have different share values from the server (secret shares of a non-zero value). We can then do casework and turn $\mathbb{Z}_{2^k}$ shares into $\mathbb{Z}_P$ without a blowup in complexity.

Conventions for sketching descriptions. In this section, to not over-complicating the explanations, we avoid all edge cases and assume that every DCF in mention has breaking point at least 2δ away from either 0 or the maximum value of the input domain. Further, we only describe DCF, as LDCF and RDCF requires some minor tweaks but the general ideas remain the same. In all following discussions, two servers obtain a subtractive secret-shared vector (after doing full-domain evaluation of the function secret sharing), and want to sketch whether this secret-shared vector is in a specific form. More specifically, two parties have shares $[\![\vec{v}]\!]_0$ and $[\![\vec{v}]\!]_1$ for a vector $\vec{v}$, where $[\![\vec{v}]\!]_0 - [\![\vec{v}]\!]_1 = \vec{v}$. There are some common vector forms that we will use short-hand notations for them:

- *Unit vector $v^P_{\alpha,a}$ (output vectors for point functions):* Vectors with a single entry at index α taking a non-zero value a. For example, $v^P_{2,a} = (0, 0, a, 0, 0, \ldots, 0)$ is a unit vector.

- *Cumulative vectors $v^C_{\alpha,a}$ (Output vectors for comparison functions):* Vectors where every entry with index until α is equal to a, while other entries are zero. We call α the *splitting point* (consistently with the convention for DCF) and a the *payload*. For example, $v^C_{3,a} = (a, a, a, a, 0, 0, \ldots, 0)$ is a cumulative vector. Note that we omit the *length* of the vector in this notation, as this information would be clear from the context.

Linear transformations. The main idea behind the sketches described in the next sections is through *linear transformations*. For two servers to learn whether they are having secret shares for a vector of a particular form, the two servers locally linearly transform their shares, until obtaining secret shares of a (supposedly) unit vector or a cumulative vector. Then, using known techniques on sketching unit vectors and cumulative vectors [BGI16], the servers can sketch with small communication cost. The linear transformations can take the following forms:

- *Taking differences $[\![\vec{v}]\!] \mapsto \Delta[\![\vec{v}]\!]$:* This operation takes a vector $\vec{v} = (v_0, v_1, \ldots, v_{N-1})$ of length N, and outputs a new vector of length $N - 1$. This vector contains the differences between each pair of consecutive elements of $\vec{v}$:

$$\Delta(\vec{v}) = (v_1 - v_0, v_2 - v_1, \ldots, v_{N-1} - v_{N-2}). \tag{11}$$

  Using this operation, one can turn a cumulative vector into a unit vector, and will be used in the sketching scheme for cumulative vectors in Figure 14.

- *Left shifting by t positions $\vec{v} \mapsto \vec{v}_{\to t}$:* Take a secret-shared vector $\vec{v} = (v_0, v_1, \ldots, v_{N-1})$ of length N, and output a new vector of length $N - t$. This output vector is $\vec{v}$ but being left shifted by t positions:
$$\vec{v}_{\to t} = (v_t, v_{t+1}, \ldots, v_{N-1}). \tag{12}$$

- *Duplicating a vector $[\![\vec{v}]\!] \mapsto [\![\vec{v}]\!]_\times$:* Take a secret-shared vector $[\![\vec{v}]\!] = ([\![v_0]\!], [\![v_1]\!], \ldots, [\![v_{N-1}]\!])$ of length N, and output a new vector of length $2N$. This output vector is $[\![\vec{v}]\!]$ but each of its element is repeated twice:
$$[\![\vec{v}]\!]_\times = ([\![v_0]\!], [\![v_0]\!], [\![v_1]\!], [\![v_1]\!], \ldots, [\![v_{N-1}]\!], [\![v_{N-1}]\!])\,. \tag{13}$$

- *Cut-off by t positions $[\![\vec{v}]\!] \mapsto [\![\vec{v}]\!]_{:-t}$:* This operation takes a secret-shared vector $[\![\vec{v}]\!]$ of length N, and outputs a new vector of length $N - t$. This output vector is $[\![\vec{v}]\!]$ but removing the last t elements:
$$[\![\vec{v}]\!]_{:-t} = ([\![v_0]\!], [\![v_1]\!], \ldots, [\![v_{N-t-1}]\!])\,. \tag{14}$$

- *Coefficient-wise multiplying with another public vector $[\![\vec{v}]\!] \mapsto [\![\vec{v}]\!] \odot \vec{V}$:* This operation takes as input a vector $\vec{v} = (v_0, v_1, \ldots, v_{N-1})$ of length N that is secret shared, and another vector $\vec{V} = (V_0, \ldots, V_{N-1})$ that is known by both parties. Both parties then element-wise multiply their shares with $\vec{V}$, locally:
$$[\![\vec{v}]\!] \odot \vec{V} = ([\![v_0]\!] \cdot V_0, [\![v_1]\!] \cdot V_1, \ldots, [\![v_{N-1}]\!] \cdot V_{N-1})\,. \tag{15}$$

Inputs. Two parties have a secret-shared vector $[\![\vec{v}]\!]$ of length N, with elements in a large prime field $\mathbb{Z}_P$.
To verify. Either there exists $q \in [0, N-1]$ such that $\vec{v} = v^P_{q,1}$ or $\vec{v}$ is a zero vector.
Sketch. Two parties do the following:

1. Jointly sample $\vec{r} = (r_0, r_1, \ldots, r_{N-1}) \xleftarrow{\$} \mathbb{Z}_P^N$.

2. Locally obtain two dot products. The first-degree dot product is:
$$[\![v_*]\!] = [\![v_0]\!] \cdot r_0 + [\![v_1]\!] \cdot r_1 + \ldots + [\![v_{N-1}]\!] \cdot r_{N-1}. \tag{16}$$
The second-degree dot product is:
$$[\![v_\bullet]\!] = [\![v_0]\!] \cdot r_0^2 + [\![v_1]\!] \cdot r_1^2 + \ldots + [\![v_{N-1}]\!] \cdot r_{N-1}^2. \tag{17}$$

Decision. Two parties run secure 2PC to check whether $v_*^2 - v_\bullet$ is equal to 0.

Figure 13: Sketching for unit vectors.

Some useful sketching schemes. Before going into the details for sketching of PB-FSS (and PB-iFSS), we recall some useful sketching techniques from previous works that would be useful in this paper as well: sketching for *unit vectors*, *cumulative vectors* and sketching for *equality*. Sketching for unit vectors is a well-known technique [BGI16], which we describe again in Figure 13 for the self-containment of the paper. Note that this sketching scheme only has negligible soundness error when the vector elements are in a large prime field $\mathbb{Z}_P$. To check if a vector is a cumulative vector, we can simply check whether taking its difference results in a unit vector. The sketching technique for cumulative vectors is shown in Figure 14.

Another useful sketching routine is for checking whether two parties are possessing a secret-share of the zero vector, i.e., both shares are equal. The idea is to take a *random linear combination* of the zero vector, then both parties share only this random linear combination to each other to check equality. If the vector elements are in a large prime field $\mathbb{Z}_P$, it is easy to show that the soundness error of this

Inputs. Two parties have a secret-shared vector $[\![\vec{v}]\!]$ of length N.
To verify. Either there exists $q \in [0, N-1]$ such that $\vec{v} = v^C_{q,1}$ or $\vec{v}$ is a zero vector.
Sketch. Two parties do the following:

1. Locally obtaining the difference $[\![\vec{v'}]\!] = \Delta[\![\vec{v}]\!]$.
2. Run the sketch step in Figure 13 with input $[\![\vec{v'}]\!]$, then receive $[\![v'_*]\!]$ and $[\![v'_\bullet]\!]$.

Decision. Two parties run secure 2PC to check whether $(v'_*)^2 - v'_\bullet$ is equal to 0.

Figure 14: Sketching for cumulative vectors.

sketching technique is $1/q$, which is negligible. When the vector elements are in a binary ring $\mathbb{Z}_{2^k}$, taking random linear combination only has soundness error $1/2$, which is not negligible. We take care of this in Section **??**, with another sketching technique for secret-shared zero vectors in $\mathbb{Z}_{2^k}$. The sketch for $\mathbb{Z}_P$ with random linear combination is shown in Figure 15. In practice, two parties can jointly sample a long random vector by only jointly sample a PRG seed, then obtaining the whole pseudo-random vector locally.

Inputs. Two parties have a secret-shared vector $[\![\vec{v}]\!]$ of length N, with elements in a large prime field $\mathbb{Z}_q$.
To verify. $\vec{v}$ is a zero vector, i.e., $v_0 = v_1 = \ldots = v_{N-1} = 0$.
Sketch. Two parties do the following:

1. Jointly sample $\vec{r} = (r_0, r_1, \ldots, r_{N-1}) \xleftarrow{\$} \mathbb{Z}_P^N$.
2. Locally obtaining the dotproduct $[\![\tilde{v}]\!] = [\![v_0]\!] \cdot r_0 + [\![v_1]\!] \cdot r_1 + \ldots + [\![v_{N-1}]\!] \cdot r_{N-1}$.

Decision. Two parties run secure 2PC to check whether $\tilde{v}$ is equal to 0.

Figure 15: Sketching for zero vectors.

7.1 Sketching for L_∞

In this subsection, we provide sketching techniques for PB-FSS of L_∞ distance metric. To keep things intuitive, we assume that the DCF payloads are elements from a ring with large enough modulo $\mathbb{Z}_P$. The case when payloads are $\mathbb{Z}_{2^k}$ elements will be taken care of in Section 7.4. We first give sketching method for non-incremental PB-FSS, which concerns sketching a DCF as well as checking if two DCF are shifted by a given amount.

Recall that the PB-FSS key for L_∞, given a distance threshold δ, contains two DCF keys at splitting points $q - \delta - 1$ and $q + \delta$. If the client is honest, the two servers after doing full-domain evaluation on two DCF keys expect to obtain secret shares of two vectors $[\![\vec{u}]\!]$ and $[\![\vec{v}]\!]$ of length N that:

i) Are both cumulative vectors, // $v^C_{q-\delta-1,1}$ and $v^C_{q+\delta,1}$

ii) Have splitting points being *exactly* $2\delta + 1$ apart from each other.

Property i) can be checked using the sketching scheme for cumulative vectors in Figure 14. To check that the splitting points are exactly $2\delta + 1$ apart from each other, the parties left-shift the secret-shared vector $[\![\vec{v}]\!]$ by $2\delta + 1$ positions, and compare it with the first $N - 2\delta - 1$ elements of $[\![\vec{u}]\!]$ using the zero check in Figure 15. Note that even though the last $2\delta + 1$ elements of $\vec{u}$ are not checked in property ii), their wellformedness *relative to* the first $N - 2\delta - 1$ elements of $\vec{u}$ had already been taken care of when checking for property i). The sketching scheme for two DCF with correlated splitting points is shown in Figure 16.

Inputs. Two parties have two secret-shared vector $[\![\vec{u}]\!]$ and $[\![\vec{v}]\!]$ of length N, with elements in a large prime field $\mathbb{Z}_P$.
To verify. One out of three following things is satisfied:

- There exists $q \in [0, N-2\delta-2]$ such that $\vec{u} = v^C_{q,1}$ and $\vec{v} = v^C_{q+2\delta+1,1}$, or
- There exists $q \in [0, 2\delta-1]$ such that $\vec{v} = v^C_{q,1}$ and $\vec{u}$ is an all-zero vector, or
- There exists $q \in [N-2\delta-1, N-1]$ such that $\vec{u} = v^C_{q,1}$ and $\vec{v}$ is an all-one vector.

Sketch. Two parties do the following:

1. Run the sketch step in Figure 14 with input $[\![\vec{u}]\!]$, then receive $[\![u_*]\!]$ and $[\![u_\bullet]\!]$.
2. Run the sketch step in Figure 14 with input $[\![\vec{v}]\!]$, then receive $[\![v_*]\!]$ and $[\![v_\bullet]\!]$.
3. Left-shift $[\![\vec{v'}]\!] = [\![\vec{v}]\!]_{\to 2\delta+1}$, and remove the last $2\delta+1$ elements of $\vec{u}$ into $[\![\vec{u'}]\!] = [\![\vec{u}]\!]_{:-(2\delta+1)}$.
4. Locally subtract and obtain $[\![\vec{w}]\!] = [\![\vec{u'}]\!] - [\![\vec{v'}]\!]$.
5. Run the sketch step in Figure 15 with input $[\![\vec{w}]\!]$, then receive $[\![\tilde{w}]\!]$.

Decision. Two parties run secure 2PC to check whether all $u_*^2 - u_\bullet$, $v_*^2 - v_\bullet$, and $\tilde{w}$ are equal to 0.

Figure 16: Sketching for cumulative vectors with correlated splitting points.

7.2 Sketching for L_p

The DCF-based construction for PB-FSS in the L_p distance metric, at its core, consists of multiple DCF key pairs, with breaking points being q, $q+\delta$, or $q-\delta$, where q is a given center. Furthermore, the *payload* of these DCFs are *dependent on the splitting point*, which is the main difference compared to PB-FSS for L_∞ distance. Considering the construction in Figure 6 for the L_p distance metric, upon doing full domain evaluation, the two servers will obtain secret shares of $p+1$ different cumulative vectors, where the i-th vector has splitting point q, with the payload on the left of q being $(-1)^{p-i}q^i$, and the payload on the right being $(-1)^i q^i$. By locally subtracting every element in the vector by the right payload, the two servers obtain $p+1$ cumulative vectors of the form $v^C_{q,f_i(q)}$, where each f_i is a polynomial taking q as input.

Let the normalized secret shared vectors that two servers obtained be $[\![\vec{v}_0]\!], [\![\vec{v}_1]\!], \ldots, [\![\vec{v}_p]\!]$, then the properties to sketch can be summarized as follows:

- All $\vec{v}_0$, $\vec{v}_1$, …, $\vec{v}_p$ are cumulative vectors, and
- All have the same splitting point q, and
- **Key challenge:** All have payload being dependent on the splitting point. // $\vec{v}_i = v^C_{q,f_i(q)}$

Property i) can be checked with the sketching protocol in Figure 14, while property ii) can be checked with the equality sketching scheme in Figure 15. In this section, we describe the sketching technique for cumulative vector with payload being dependent on the splitting point. Again, we focus on the case when the vector elements are in a large prime field $\mathbb{Z}_P$ for now.

Sketch for cumulative vector with payload as a function of the splitting point. Consider a secret-shared vector $[\![\vec{v}]\!]$ of length N and a function $f(\cdot)$, in which we want to test whether there exists q such that $\vec{v} = v^C_{q,f(q)}$. The two servers first transform it into a unit vector by taking the difference, more specifically, one expect that $\Delta\vec{v} = v^P_{q,-f(q)}$. The crucial observation is that $v^P_{q,-f(q)}$ can be *linearly*

transformed into a unit vector with payload equals to 1. First, to linearly transform a unit vector $v^P_{q,1}$ into $v^P_{q,-f(q)}$, we always use the element-wise multiplication with the following payload matrix:

$$\begin{aligned} v^P_{q,-f(q)} =& (0,0,\ldots,0,-f(q),0,\ldots,0) \\ =& \begin{array}{cccccccc} (0, & 0, & \ldots, & 0, & 1, & 0, & \ldots, & 0) \\ & & & & \odot & & & \\ (-f(1), & -f(2), & \ldots, & -f(q-1), & -f(q), & -f(q+1), & \ldots, & -f(N-1)) \end{array}. \end{aligned}$$

In the prime field $\mathbb{Z}_P$, to inverse this operation and transform $v^P_{q,-f(q)}$ into $v^P_{q,1}$, we use element-wise multiplication with the vector containing *inverses* of the payloads:

$$v^P_{q,1} = v^P_{q,-f(q)} \odot (-f(1)^{-1}, -f(2)^{-1}, \ldots, -f(q)^{-1}, \ldots, -f(N-1)^{-1}) \bmod P. \tag{18}$$

We call this linear transformation *rescaling*. Later in Section 7.4, we show that rescaling can work even when vector elements are in a binary ring $\mathbb{Z}_{2^k}$, where the inverse might not exist. After this linear transformation, the remaining task is to test for unit vector with the scheme in Figure 13. We show the formal description of the sketching scheme for cumulative vectors with payload dependent on the splitting point in Figure 17. This test can then be combined with the zero check in Figure 15 (to test relation between different DCFs), to complete the sketching scheme for PB-FSS in L_p.

Inputs. Two parties have a secret-shared vector $[\![\vec{v}]\!]$ of length N, with elements in a large prime field $\mathbb{Z}_P$, and given a function $f : [0, N-1] \to \mathbb{Z}_P$.
To verify. Either there exists $q \in [0, N-1]$ such that $\vec{v} = v_{q,-f(q)}$ or $\vec{v}$ is an all-zero vector.
Sketch. Two parties do the following:

1. Locally obtain $[\![\vec{v'}]\!] = \Delta[\![\vec{v}]\!] \odot (-f(1)^{-1}, -f(2)^{-1}, \ldots, -f(N-1)^{-1}) \bmod P$.
2. Run the sketch step in Figure 13 with input $[\![\vec{v'}]\!]$, then receive $[\![v'_*]\!]$ and $[\![v'_\bullet]\!]$.

Decision. Two parties run secure 2PC to check whether $(v'_*)^2 - v'_\bullet$ is equal to 0.

Figure 17: Sketching for cumulative vector with payload being a function of the splitting point.

7.3 Sketch for incremental evaluation

Our incremental PB-iFSS construction, regardless of the distance metric (L_∞ or L_p), contains some incremental LDCF and some incremental RDCF where splitting points and payloads are correlated to each other in some way. To sketch for incremental PB-iFSS, our recipe consists of two parts: *inner sketching* and *outer sketching*. To understand these two parts, we first recall the definition of LDCF (similarly, RDCF) and see the difference between sketching for non-incremental PB-FSS and sketching for incremental PB-iFSS.

Evaluating on prefixes. An incremental DCF allows evaluating on not only the full strings in a binary domain $\{0,1\}^h$, but also on any *prefix* in this domain $\{0,1\}^{\leq h}$. For example, for an LDCF with splitting point $q \in \{0,1\}^h$, the evaluation on a prefix x goes as follows: Let q' be the prefix of q with the same length as x, then output a if $x \leq q'$, and output b otherwise. With this pattern, a full-domain evaluation on the LDCF with splitting point $q \in \{0,1\}^h$ can be divided into h *layers*, corresponding to h possible prefix lengths. For i from 1 to h, let $q^{(i)} \in \{0,1\}^i$ denote the length-i prefix of q, then each layer's evaluation output is described as follows: at layer i, on a prefix $x \in \{0,1\}^1$, output a if $x \leq q^{(1)}$, output b otherwise.

Each layers of prefix evaluations is a cumulative vector. Upon full domain evaluation of LDCF, at layer i with prefix $q^{(i)}$, the two servers will obtain secret shares of a vector with elements equal

to a value a if their index is at most $q^{(i)}$, and equal to another value b otherwise. By locally subtracting their shares by b, the servers obtain secret share of $v^C_{q^{(i)},a-b}$. Let the normalized vectors that the servers obtained at this stage be $[\![\vec{v}_1]\!], [\![\vec{v}_2]\!], \ldots, [\![\vec{v}]\!]_h$, with corresponding length $2^1, 2^2, \ldots, 2^h$, the properties to sketch would be as follows:

i) All $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_h$ are cumulative vectors, and

ii) **Key challenge:** The i-th layer vector $\vec{v}_i$ has splitting point being the length i prefix $q^{(i)}$ of q, and

iii) All have the same payload.

Property i) can be checked with the scheme in Figure 14. After checking property i), it is guaranteed that there exists $q \in \{0,1\}^h$ and a payload a such that $\vec{v}_h = v^C_{q,a}$. To check property ii) and iii), we check other layers' vectors *in relative* to $\vec{v}_h$. In other words, we develop a sketching technique to check *each pair of consecutive layers* on being cumulative vectors with splitting points being prefix/suffix of each other.

Sketch for each pair of consecutive layers. Two servers want to check that any two consecutive layers $\vec{v}_i$ and $\vec{v}_{i+1}$ have the same payloads, with splitting points being $q^{(i)}$ and $q^{(i+1)}$. Further recall that $\vec{v}_i$ has length 2^i and $\vec{v}_{i+1}$ has length 2^{i+1}. By *duplicating* $\vec{v}_i$, we obtain a cumulative vector of length 2^{i+1} with splitting point $2q^{(i)}+1$, which is equal to either $q^{(i+1)}$ or $q^{(i+1)}-1$, since $q^{(i)}$ is a prefix of $q^{(i+1)}$. Hence, the duplicated $[\![\vec{v}_i]\!]_\times$ equals to $[\![\vec{v}_{i+1}]\!]$, either directly or after being left-shifted by 1 position. The two servers do not know which case it is, but can check each with a zero check from Figure 15, then use 2PC to take the OR of the two checks, and finish the sketch. The sketching scheme for checking two cumulative vectors with splitting points being prefix/suffix to each other is shown in Figure 18. The scheme described in Figure 18 only considers cumulative vectors with payload equals to 1, but can be used for any fixed payload after local rescaling.

Inputs. Two parties have two secret-shared vectors $[\![\vec{u}]\!]$ and $[\![\vec{v}]\!]$ of lengths $N = 2^i$ and $2N$, respectively. Both vectors have elements in a large prime field $\mathbb{Z}_P$.
To verify. One out of three following things is satisfied:

- There exists $q \in [0, N-1]$ such that $\vec{u} = v^C_{q,1}$ and $\vec{v} = v^C_{2q,1}$, or
- There exists $q \in [0, N-1]$ such that $\vec{u} = v^C_{q,1}$ and $\vec{v} = v^C_{2q+1,1}$, or
- Both $\vec{u}$ and $\vec{v}$ are zero vectors.

Sketch. Two parties do the following:

1. Run the sketch step for cumulative vector in Figure 14 with input $[\![\vec{v}]\!]$, then receive $[\![v_*]\!]$ and $[\![v_\bullet]\!]$.
2. Locally obtain the duplicated $[\![\vec{u'}]\!] = [\![\vec{u}]\!]_\times$, then subtract by $[\![\vec{v}]\!]$ to get $[\![\vec{w}]\!] = [\![\vec{u'}]\!] - [\![\vec{v}]\!]$.
3. Run the sketch step for zero vector in Figure 15 with input $[\![\vec{w}]\!]$, then receive $[\![\tilde{w}]\!]$.
4. Locally obtain the difference $[\![\vec{w'}]\!] = [\![\vec{u'}]\!]_{\to 1} - [\![\vec{v}]\!]_{:-1}$ between the left-shifted $\vec{u'}$ and $\vec{v}$.
5. Run the sketch step for zero vector in Figure 15 with input $[\![\vec{w'}]\!]$, then receive $[\![\tilde{w'}]\!]$.
6. Output $[\![v_*]\!]$, $[\![v_\bullet]\!]$, $[\![\tilde{w}]\!]$ and $[\![\tilde{w'}]\!]$.

Decision. Two parties run secure 2PC to check whether both $(v_*)^2 - v_\bullet$ and $\tilde{w} \cdot \tilde{w'}$ are equal to 0.

Figure 18: Sketching for two cumulative vectors where one's splitting point is a prefix of the other's.

Using this technique for PB-iFSS in L_p distance.

In the DCF-based construction for L_∞ distance PB-iFSS, we have two iDCF key pairs with breaking points $2\delta+1$ apart from each other. Turns out checking the shift in two last layers of two iDCF, plus the consistency between layers of each iDCF, is already enough. There is no additional check between corresponding previous layers required, and it is not possible as well, since it is not always the case that $[\![\vec{u^i}]\!]$ and $[\![\vec{v^i}]\!]$ are shifted from each other an amount equals to a prefix of $2\delta+1$.

7.4 Optimization: Sketching for payloads in $\mathbb{Z}_{2^k}$

In Mosaic, as bit comparison in the 2PC step afterward is essential, we use DCF output as binary ring elements $\mathbb{Z}_{2^k}$ instead of prime modulo ring $\mathbb{Z}_P$ (more specifically, the ring used in Mosaic is $\mathbb{Z}_{2^{h_2}}$). The current sketching technique for DCF/DPF by taking random linear combination with values in $\mathbb{Z}_{2^k}$ does not work, since there are non-invertible elements in the binary ring, and the soundness argument does not work.

Warm up: Checking equality of secret-shared zero vector in $\mathbb{Z}_{2^k}$. Suppose that the two servers have secret shares for a vector $[\![\vec{u}]\!]$ and want to check that all entries in $\vec{u}$ are zero. For each index i, servers $\mathcal{S}_0$ and $\mathcal{S}_1$ would have shares $[\![u_i]\!]_0$ and $[\![u_i]\!]_1$ such that $[\![u_i]\!]_0 + [\![u_i]\!]_1 = 0 \bmod 2^k$. By letting $\mathcal{S}_0$ obtain $a_i = [\![u_i]\!]_0$, and $\mathcal{S}_1$ obtain $b_i = -[\![u_i]\!]_1 \bmod 2^k$, the problem turns into checking whether $a_i = b_i$ for all indices i. The crucial observation here is: a_i and b_i are equal *integer* values, not just values in $\mathbb{Z}_{2^k}$. Hence, one can "cast" a_i and b_i into $\mathbb{Z}_P$, and it would still be the case that $a_i = b_i$. The two servers can finish sketching by sampling common random values $r_0, r_1, \ldots, r_{2^{h_1}} - 1 \in \mathbb{Z}_P$, then compute $\sum_{i=0}^{2^{h_1}-1} r_i a_i \bmod P$ and $\sum_{i=0}^{2^{h_1}-1} r_i b_i \bmod P$ as sketching values.

The next step: Sketching for unit vectors in $\mathbb{Z}_{2^k}$. Next we assume that two servers have secret shared values for a vector $[\![\vec{u}]\!]$ with values in $\mathbb{Z}_{2^k}$, and want to sketch that this is a unit vector. This is an immediate step as any secret-shared cumulative vector can be transformed into unit vector with a linear transformation. Our crucial observation is that, when again, letting $\mathcal{S}_0$ obtain $a_i = [\![u_i]\!]_0$ and $b_i = -[\![u_i]\!]_1 \bmod 2^k$, we have $a_i = b_i$ at *all but one* entries. Suppose that the entry q is where $a_q \neq b_q$. Since we have $a_q - b_q = 1 \bmod 2^k$, there are only two possible cases: either $a_q = b_q + 1$, or $a_q = b_q + 1 - 2^k$. The two equations we obtained in this case, again, hold for *integer value* of a_i and b_i, and can be casted into $\mathbb{Z}_P$. This means, in $\mathbb{Z}_P$, the two vectors $\vec{a}$ and $\vec{b}$ are actually *subtractive shares* for a weight-one vector in $\mathbb{Z}_P$, where the non-zero entry is either 1 or $1 - 2^k \bmod P$. Here comes the magic: we can check the second case by multiplying the secret shares with a "rescaling" vector in modulo P, that contains all inverses of $1 - 2^k$ in modulo P: $((1-2^k)^{-1} \bmod P, \ldots, (1-2^k)^{-1} \bmod P)$.

In conclusion, there are two cases: either $\vec{a}$ and $\vec{b}$ are subtractive shares of a unit vector in $\mathbb{Z}_P$, or by multiplying $\vec{a}$ and $\vec{b}$ with the "rescaling" vector in modulo P, we get secret shares of a unit vector. Even though we do not know which case is this, the two servers can still obtain sketching values for unit vectors for both cases, and run 2PC to multiply these two sketching values together. If the product obtained is 0, then at least one out of two cases hold, which implies that $[\![\vec{u}]\!]$ is secret shares for a unit vector in $\mathbb{Z}_{2^k}$.

Wrap it up: Sketching for DCF in $\mathbb{Z}_{2^k}$. The remaining step is actually folkore, since we can take different between consecutive elements in $\mathbb{Z}_{2^k}$ to turn a cumulative vector into a unit vector in $\mathbb{Z}_{2^k}$. We note here that this sketching technique is already sufficient even for the PB-FSS construction in L_p, as we only need to do one sketch for DCF, and everything else are equality checks. The formal description of this sketching protocol is shown below.

Sketch. The servers receive from the client subtractive shares of a vector $[\![\vec{u}]\!]$, where $\vec{u} \in \mathbb{Z}_{2^k}^N$. If the client is honest, the vector $\vec{u}$ would be of the form $(1, 1, \ldots, 1, 0, 0, \ldots, 0) \in \mathbb{Z}_{2^k}^N$. Given P being a large enough prime, both parties do the following:

1. Obtain $[\![\vec{u'}]\!] = [\![(u_1 - u_0, u_2 - u_1, \ldots, u_{N-1} - u_{N-2}, 0)]\!]$. Let the share obtained by $\mathcal{S}_0$ be $[\![\vec{u'}]\!]_0$ and the share obtained by $\mathcal{S}_1$ be $[\![\vec{u'}]\!]_1$. If the client is honest, the two servers have $[\![u'_i]\!]_0 = [\![u'_i]\!]_1$ at all but one value $i \in [N-1]$.

2. Jointly sample $\vec{r^1} = (r_0, r_1, \ldots, r_{N-1}) \in \mathbb{Z}_P^N$. Let the powers of $\vec{r}$ be $\vec{r^i} = (r_0^i, r_1^i, \ldots, r_{N-1}^i) \in \mathbb{Z}_p$ for each $i = 2, 3, 6$.

3. Obtain $[\![\vec{u_*}]\!]_b = \left(\frac{[\![u'_0]\!]_b}{1}, \frac{[\![u'_1]\!]_b}{1}, \ldots, \frac{[\![u'_{N-1}]\!]_b}{1}\right) \bmod P$ as the "rescaled" vector for the first case.

4. Obtain $[\![\vec{u_\bullet}]\!]_b = \left(\frac{[\![u'_0]\!]_b}{1-2^k}, \frac{[\![u'_1]\!]_b}{1-2^k}, \ldots, \frac{[\![u'_{N-1}]\!]_b}{1-2^k}\right) \bmod P$ as the "rescaled" vector for the second case. If the client is honest, either $\vec{u_*}$ or $\vec{u_\bullet}$ would be a unit vector in $\mathbb{Z}_P$.

5. Compute the inner products $[\![z_*]\!] = \langle \vec{r^1}, [\![\vec{u_*}]\!] \rangle$ and $[\![z'_*]\!] = \langle \vec{r^2}, [\![\vec{u_*}]\!] \rangle$ for testing whether $\vec{u_*}$ is a unit vector.

6. Compute the inner products $[\![z_\bullet]\!] = \langle \vec{r^1}, [\![\vec{u_\bullet}]\!] \rangle$ and $[\![z'_\bullet]\!] = \langle \vec{r^2}, [\![\vec{u_\bullet}]\!] \rangle$ for testing whether $\vec{u_\bullet}$ is a unit vector.

7. The obtained secret shares $[\![z_*]\!], [\![z'_*]\!], [\![z_\bullet]\!], [\![z'_\bullet]\!]$ are the sketching outputs.

Decision. The parties run secure multiparty computation in $\mathbb{Z}_P$ to check whether the following value is 0:

$$(z_*^2 - z'_*) \cdot (z_\bullet^2 - z'_\bullet). \tag{19}$$

8 Application: Privacy-Preserving Ride-Sharing

We design a complete system in Rust to provide a proof-of-concept for the Mosaic framework. Keeping modularity in mind, we implement all methods that are described in the paper for each phase: Client Uploading Phase and Fuzzy Matching Phase (including Private Property Testing and Private Threshold Comparison), as well as an optional Sketching Phase for preventing malformed inputs from malicious clients. More details on the experiment environment would be shown in Section 8.1, before we introduce the dataset used in Section 8.2. We then first compare our protocol against the baseline solution in Section 8.3, before analysing the scalability for end-to-end runs of our system in Section 8.4.

Table 2: Running time in LAN for Known Dictionary, compared against naïve baseline.

PB-FSS	Server Time (s)						Client Comm (GB)						Client Time (s)					
	L_∞			L_2			L_∞			L_2			L_∞			L_2		
	δ_2	δ_3	δ_4	δ_2	δ_3	δ_4	δ_2	δ_3	δ_4	δ_2	δ_3	δ_4	δ_2	δ_3	δ_4	δ_2	δ_3	δ_4
Naïve	6.53	12.7	21.0	3.39	7.57	12.8	0.59	1.16	1.92	0.31	0.69	1.16	47.9	95.4	157	24.9	55.8	94.9
OKVS	2.15	2.17	2.18	2.23	2.29	2.36	0.04	0.04	0.04	0.04	0.04	0.04	1.46	1.69	1.97	1.46	1.72	1.96
DCF	0.59	0.59	0.59	1.27	1.30	1.34	0.04	0.04	0.04	0.09	0.09	0.10	3.39	3.37	3.38	8.21	8.54	8.98

8.1 Implementation Details

We implement the Mosaic system in Rust.[2] The system contains implementations for three types of parties: the Client who acts as N real clients and generates all of their PB-FSS (or PB-iFSS) key shares, the Server which contains two parties who receive client shares and run the protocol, and an optional Dealer in the case when FSS is required in the Fuzzy Matching Phase. We use AWS.EC2 `c5a.24xlarge` instances to run the two servers and the dealer, while we simulate the clients with a `t2.2xlarge` instance to reflect lower computational resource. As such, we parallelize the protocol run by the two servers and dealer through 32 cores, while letting the client run each key generation process sequentially and measure the total runtime. We test the protocol in the LAN setting. All parties communicate with each other through TCP, using the networking implementation from the `swanky`[3] cryptography suite.

[2] https://github.com/phuocchubeo123/fuzzyheavyhitters.git

[3] `https://github.com/GaloisInc/swanky`

To implement the OKVS-based PB-iFSS construction, we use RB-OKVS from Bienstock et al. [BPSY23], which is currently the state-of-the-art construction with near-linear encoding time and small rate. All of the FSS algorithms in the DCF-based construction for PB-FSS are implemented following the GGM tree paradigm from [BGI16], with the incremental construction following the template from [BBCG$^+$21, GGM24]. For property testing and threshold comparison, we implemented both the 2PC-based and Dealer-based approaches. In 2PC, we use the Garbled Circuits library `fancy-garbling` and the Oblivious Transfer library `ocelot`, both from the `swanky` cryptography suite. Regarding the Dealer, due to memory constraints and the difficulty of predicting the number of FSS keys needed for the unknown dictionary problem, we let the Dealer generate keys on demand, upon request from the two servers.

8.2 Dataset and Protocol Parameters

We run Mosaic on a real-world dataset from the RideAustin app.[4] We focus on coordinates of rides from this dataset, which contains latitude ranging from 29.767 to 30.767 and longitude ranging from -98.243 to -97.243, enclosing the city of Austin, Texas. Upon data cleaning, we normalize floating points into 11-bit value representation, based on the initial accuracy of 3 decimal digits. One unit of change then corresponds to 111 meters in latitude and 88 meters in longitude. We choose three subsets from this dataset to run our experiments, namely rides from the busiest day ($21,581$ rides), the busiest week ($59,040$ rides), and the busiest month ($115,174$ rides).

Protocol parameters. Based on the properties of the cleaned datasets, we choose $h_1 = 11$ to represent each input coordinate. We choose h_3 to be 15, 16, and 18 respectively to store the largest possible count of matches with $21K$, $59K$ and $115K$ rides. The distance threshold we chose is $\delta = 5$, and we run our protocol across multiple distance metrics: L_∞, L_1, and L_2. In order for a point to be considered a *fuzzy heavy hitter*, it would need to be close to at least 2% of points in the dataset. In addition, for parameters related to the protocol, we choose the computational security parameter $\kappa = 128$ and the statistical security parameters $\lambda = 40$.

8.3 Comparison with Baseline Solution

Recall from Section 1 that a straightforward approach to the fuzzy heavy hitters problem is to reduce it to the exact setting, where each client enumerates over all possible points in the neighborhood of its input (i.e., all $\vec{x}$ such that $\mathsf{dist}(\vec{x}, \vec{q}) \leq \delta$). The servers then run an exact heavy hitters protocol over this expanded dataset, treating any discovered heavy hitter as an approximate heavy hitter of the original data. To illustrate our performance against the baseline solution, we test the throughput by running the protocols in the Known Dictionary setting. We take the set of start points from rides in the busiest day in Ride Austin ($21K$ points in 2 dimensions), and test it against one dictionary point from the two servers. We run the experiment in the LAN network setting, with two different distance metrics: L_∞ and L_2, and three different values for three distance thresholds: $\delta_2 = 2$, $\delta_3 = 3$, and $\delta_4 = 4$. The reason we choose these smaller thresholds is due to the RAM consumption blowup in the naïve baseline for larger values of δ. We report the numbers in Table 2.

Client key size and key generation time. One clear distinction between Mosaic and the baseline solution is communication per client. For 2-dimensional inputs with distance threshold ranging from $\delta = 2$ to 4, the client key size of Mosaic is already significantly smaller than the naïve baseline, by a factor of $15 - 48\times$ for L_∞ and $3.5 - 29\times$ for L_p. The key generation time for client shows even larger improvements compared to the naïve solution, by a factor of $14 - 80\times$ and $3 - 48\times$ respectively for the two distance metrics. This performance gap would increase even more sharply as d and δ increase.

Server computation. The number of DCF.Eval (in DCF-based construction) calls or the number of hashes in OKVS for Mosaic is significantly less than the naïve solution. Recall that in the naïve baseline,

[4] `https://data.world/ride-austin/ride-austin-june-6-april-13`

to evaluate a point for each client, the servers would have to call DPF.Eval with the number of times equal to the ball size, which again would grow rapidly as the number of dimensions increase. This translates to a $3-35\times$ improvement in server throughput in the L_∞ distance metric, and a $1.5-9\times$ (and growing) improvement in L_p. Even though Mosaic has some additional communication overhead between servers, this cost is minor compared to the explosion of local computation in the naïve solution.

Drawbacks of baseline solution for Unknown Dictionary. There exists an *overcounting* issue for the naïve solution in the Unknown Dictionary setting. It could be the case that one value can be a prefix of multiple points in the same L_∞ or L_p ball, which causes evaluation of this value to indicate a match on multiple incremental DPF sent by one client. After aggregation of all DPF.Eval outputs, the matching count of one prefix would be reported to be a larger number. This can lead to a false positive in the prefix search algorithm, which leads to more prefixes being searched in the process, leaking more information to the server. The solution in Mosaic provides *exact* matching count, hence false positives do not occur in our system.

8.4 Empirical Performance for RideAustin

Table 3: Experiments for Mosaic. Dealer Communication shows communication from Dealer to one Server. Client communication is the total communication cost of all Clients to both Servers.

Set Size	Metric	PB-iFSS	Fuzzy Match	Client Comm. (GB)	Dealer Comm. (GB)	$\mathcal{S}_0 \to \mathcal{S}_1$ Comm. (GB)	$\mathcal{S}_1 \to \mathcal{S}_0$ Comm. (GB)	Sketch Time(s)	Client Time(s)	Server Time(s)
21K	L_∞	DCF	2PC	0.19	-	2.74	11.6	6.86	2.91	32.4
			Dealer		6.56	0.10	0.10			34.2
		OKVS	2PC	0.09	-	98.5	490	-	6.24	286
			Dealer		302	0.78	0.78			685
	L_1	DCF	2PC	0.51	-	2.95	26.7	30.0	6.66	71.5
			Dealer		15.8	0.73	0.73			88.7
		OKVS	2PC	0.85	-	32.6	265	-	6.30	165
			Dealer		199	0.56	0.56			439
	L_2	DCF	2PC	0.69	-	4.88	42.4	32.9	8.19	92.3
			Dealer		26.9	0.91	0.91			126
		OKVS	2PC	0.09	-	40.1	326	-	6.32	196
			Dealer		244	0.66	0.66	-		603
$59K$	L_∞	DCF	2PC	0.53	-	9.02	38.5	-	7.99	72.2
	L_1			1.40	-	8.78	80.1	-	18.2	188
	L_2			1.89	-	15.2	133	-	22.5	253
$115K$	L_∞	DCF	2PC	1.03	-	16.9	72.2	-	15.6	127
	L_1			2.73	-	17.2	154	-	35.4	154
	L_2			3.69	-	29.4	257	-	43.8	480

After showing our performance against the naïve baseline, we now take a closer look to end-to-end run of Mosaic on different combinations of methods. We first run our protocol in the Unknown Dictionary setting over LAN, for the subset of starting points on the busiest day, which contains 21 thousand two-dimensional points. We then show the scalability of Mosaic by running the protocol with the DCF+2PC method on larger set sizes.

Lightweight client work. In all settings, the total running time for 21 thousand clients is a matter of seconds, which means each client only needs to spend less than a millisecond to generate and send keys to servers. For example, for the L_∞ distance metric, the protocol requires only lightweight computation and communication as low as 0.14 milliseconds and 9 KB.

Trade-offs for different PB-FSS methods. The key sizes of OKVS-based constructions are smaller, while the DCF-based keys allow verifiability via sketching. The DCF-based construction supports better 2PC communication and computation cost, as its output length h_2 can be set as tight as possible to represent the maximum distance, while the output length in the OKVS-based construction needs to be at least λ. In all our experiments, the DCF-based construction achieves faster server running time compared to the OKVS-based construction. For example, in the L_∞ distance metric, using DCF takes 32.4 seconds while using OKVS takes 286 seconds.

2PC-based versus Dealer-based Fuzzy Matching. To compare between the 2PC-based approach and the Dealer-based approach, we look at the experimental results when the Fuzzy Matching phase is

implemented either fully with 2PC (in both Property Testing and Threshold Comparison) or fully with a Dealer. Compared to the 2PC-based approach, the Dealer-based one has less overall communication, and also shifts the majority of communication to the Dealer. All protocol runs for 21 thousand clients with Dealer across the three distance metrics incur less than 1GB of inter-server communication. However, evaluating FSS costs more computation than Garbled Circuits, hence the Dealer-based approach is slower in the LAN setting. For example, with the OKVS-based PB-iFSS construction under L_1, Dealer-based approach takes 439 seconds while 2PC takes only 165 seconds.

Overhead for sketching. As mentioned earlier, the DCF-based construction for PB-FSS supports arithmetic sketching, which is efficient and easily parallelizable. In Table 3, the sketching overhead ranges from 21% for the L_∞ distance metric to 42% for the L_1 distance metric.

9 Related Work

Significant progress has been made in the Private Heavy Hitters line of work. We categorize prior approaches into single-server systems, those relying on generic MPC or mix-nets, and recent advancements using Function Secret Sharing (FSS).

Single Server and Local Differential Privacy. Early approaches relied on local differential privacy [QYY+16], [ZKM+20], [BS15a], [EPK14], [App17], where clients perturb their data before sending it to a collector. This model suffers from high noise levels that can degrade accuracy compared to alternative aggregate statistics methods. Alternative single-server aggregation systems [BBG+20], [BIK+17], [MWA+23], allow a server to securely sum client inputs and sometimes use data structures like count-min sketches [CM04], [BS15b] to identify heavy hitters. However, these systems often face challenges such as requiring multiple rounds of interaction to handle client dropouts or leaking more information than necessary about the data distribution.

Generic MPC and Mix-nets. A common privacy-preserving paradigm involves outsourcing computation to multiple non-colluding servers. One approach uses mix-nets [Cha81], to blindly shuffle and decrypt client inputs. While this is efficient for finding unique items (as in a threshold $t = 1$), this approach does not easily generalize to finding t-heavy hitters without revealing non-heavy hitter items or requiring significant interaction. Other papers [AGJ+22], [DPRS23] employ Verifiable Distributed Aggregation Functions or generic two-party computation for RAM programs [GLO15], [GKK+12], [LO13]. Although generic MPC combined with Oblivious RAM (ORAM) [GO96] offers theoretically optimal asymptotic complexity, it remains concretely expensive due to the high overhead of ORAM implementation [DS17].

Function Secret Sharing (FSS). Our work builds upon FSS introduced by Boyle et al. [BGI15, BGI16]. FSS allows servers to aggregate function evaluations on secret inputs without learning the input themselves. Recent systems like Prio[CGB17] and Poplar [dCP22] utilize MPC and FSS to aggregate statistics with much less noise than differential privacy solutions. Poplar, in particular, uses Distributed Point Functions (DPFs) to enable exact heavy hitter detection with strong privacy guarantees. However, these protocols are strictly limited to exact matching. They fail to capture "fuzzy" heavy hitters, which are items that are semantically similar or close in distance but not bitwise identical. Our framework extends these FSS techniques to support L_∞ and L_p distance metrics, enabling scalable private detection of approximate heavy hitters.

Acknowledgments

This project was supported in part by NSF SaTC Award 2247352, NSF CAREER Award 2442384, Meta Research Award, Google Research Scholar Award, and Amazon Research Award. Gayathri Garimella conducted part of this work while a postdoctoral researcher at Brown University. Part of this work was

also conducted while Gayathri Garimella, Peihan Miao, and Phuoc Van Long Pham were visiting the Simons Institute for the Theory of Computing.

References

[AGJ+22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *Security and Cryptography for Networks (SCN)*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539. Springer, 2022.

[App17] Apple Privacy Team. Learning with privacy at scale. *Differential Privacy Team, Apple, Technical Report*, 1(8), 2017.

[BBCG+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

[BBG+20] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1253–1269. ACM Press, November 2020.

[BCG+21] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900. Springer, 2021.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2019.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

[BKLS24] Elette Boyle, Lisa Kohl, Zhe Li, and Peter Scholl. Direct FSS constructions for branching programs and more from PRGs with encoded-output homomorphism. Cryptology ePrint Archive, Paper 2024/192, 2024.

[BPSY23] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[BS15a] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 127–135, New York, NY, USA, 2015. Association for Computing Machinery.

[BS15b] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 127–135. ACM, 2015.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282, 2017.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[CM04] Graham Cormode and Shanmutha Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN 2004: Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 29–38. Springer, 2004.

[dCP22] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 150–179. Springer, 2022.

[DPRS23] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. *Proceedings on Privacy Enhancing Technologies*, 2023(4):578–592, 2023.

[DS17] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 523–535. ACM, 2017.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1054–1067. ACM Press, November 2014.

[GGM24] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware psi from incremental function secret sharing. In *Annual International Cryptology Conference*, pages 309–345. Springer, 2024.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

[GKK+12] S Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, pages 513–524. ACM, 2012.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 210–229. IEEE, 2015.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, August 2021. Springer, Cham.

[HN23] Keitaro Hiwatashi and Koji Nuida. Explicit and nearly tight lower bound for 2-party perfectly secure FSS. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 541–554. Springer, Cham, June 2023.

[KMPP24] Banashri Karmakar, Shyam Murthy, Arpita Patra, and Protik Paul. QuickPool: Privacy-preserving ride-sharing service. Cryptology ePrint Archive, Paper 2024/1109, 2024.

[Lin17] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 277–346, 2017.

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious ram for secure two-party computation. In *Theory of Cryptography Conference (TCC)*, volume 77 of *Lecture Notes in Computer Science*, pages 377–396. Springer, 2013.

[MWA+23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 477–496. IEEE Computer Society, 2023.

[QYY+16] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 192–203. ACM Press, October 2016.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. Cryptology ePrint Archive, Paper 2021/266, 2021.

[RZCGAP24] Mayank Rathee, Yuwen Zhang, Henry Corrigan-Gibbs, and Raluca Ada Popa. Private Analytics via Streaming, Sketching, and Silently Verifiable Proofs . In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3072–3090, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[Wid12] Martin Widmer. Lipschitz class, narrow class, and counting lattice points. *Proceedings of the American Mathematical Society*, 140(2):677–689, 2012.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZKM+20] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. Federated heavy hitters discovery with differential privacy. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3837–3847. PMLR, 26–28 Aug 2020.

Ethical Considerations

Stakeholders. The primary stakeholders include individuals using applications that may incorporate our research, individuals whose data is collected to improve these applications, service providers that rely on data analytics to enhance their services, and the broader research community working on privacy-preserving computation. Individuals using such applications will likely benefit from improved services. Individuals whose data is collected should not be adversely affected, beyond contributing to aggregate analytics that enable better service provision. Service providers can benefit from performing analytics without compromising user privacy, thereby improving their offerings. Finally, the research community is impacted as our work advances the growing field of privacy-preserving computation, which continues to evolve and expand.

Ethical principles. As outlined in the Menlo Report, our work emphasizes four guiding principles. First, we prioritize *beneficence* by enabling useful aggregate analytics without exposing individual-level data, providing insights for service providers without compromising privacy. Second, we uphold *respect for persons* by guaranteeing that clients' data remains private under the assumption that at least one server behaves honestly. Third, we promote *justice* by designing protocols that are efficient enough to run on low-resource devices, thereby ensuring inclusivity and preventing the exclusion of data from users with limited computational capacity. Finally, we commit to *respect for law and public interest* by relying exclusively on public datasets (such as RideAustin) and avoiding any experiments that might violate terms of service or legal protections.

Potential harms and mitigation. One category of harm arises if parameters are chosen maliciously or incorrectly, which could weaken security guarantees and allow adversaries to infer private information. Another concern is the injection of malicious or corrupted data by clients, which may distort aggregate results and reduce the system's utility. While the design of defenses against malicious clients or adversarial parameter choices is beyond the scope of this work, our focus on the semi-honest threat model provides a clear baseline for security and can be extended to stronger adversarial models in future research. We further limit potential harm from the publication process by restricting our evaluation to public datasets and providing transparent descriptions of our protocols, thereby ensuring that our contributions advance the field without exposing sensitive user data.

Decision to proceed. We believe that the ethical benefits of this research outweigh the risks. Mosaic advances the state of privacy-preserving computation, which provides stronger protections for end users in realistic analytics scenarios. We make our protocols transparent and evaluate them on a public non-sensitive dataset, which reduces the risk of harm while enabling future systems to provide both utility and privacy.

Open Science

The implementation of our protocol is open-source and is hosted on Zenodo along with the Artifact Appendix: `https://zenodo.org/records/21846449`. The implementation might be updated in the future, with all the changes being reflected on the GitHub repository: `https://github.com/phuocchubeo123/fuzzyheavyhitters`.

A More details for OKVS-based PB-FSS and PB-iFSS

We present the full construction for OKVS-based PB-FSS (and its incremental version) for both distance metrics L_∞ and L_p in Figure 19. The main different between the non-incremental case and the incremental case is the set of keys to be encoded in an OKVS encoding. In the non-incremental version, we only encode the values in an interval, while for the incremental version, any prefix of any string in the interval is encoded. Due to edge cases when the client's point might be near the domain bound, which leads to varying size of the OKVS key set, we hide this information by doing padding to the maximum possible key set size. We provide the calculation for the maximum number of prefixes of an interval for the incremental case in Appendix A.3. The proof of correctness and security of the construction for the L_∞ distance metric is provided in Appendix A.1, while the proofs for the L_p distance metric are provided in Appendix A.2.

A.1 OKVS-based PB-FSS for L_∞

Theorem 3 (Security of OKVS-based PB-FSS for L_∞)**.** *Given* OKVS(Encode, Decode) *(see Definition 2), the protocol in Figure 19 realizes a* PB-FSS *(or* PB-iFSS*) for the set family $\mathcal{S}_{\infty,d,\delta}$ and property* Equal *over the domain $\mathcal{D} = (\{0,1\}^{h_1})^d$ that satisfies the properties in Definition 4.*

Proof. In the proof of correctness, we only provide the analysis for PB-iFSS, as this is a generalization of PB-FSS with correctness implying the correctness of the latter as well.

Correctness for yes-instances: If a point $\vec{x}$ satisfies $L_\infty(\vec{x}, \vec{q}) \leq \delta$, it means in every dimension i, we have $|x_i - q_i| \leq \delta$. This is equivalent to the fact that $x_i \in [q_i - \delta, q_i + \delta]$ for every dimension i. For each dimension i, let $y^{(i)}_{\mathsf{idx}} = \mathsf{OKVS.Decode}(k^{(i)}_{\mathsf{idx}}, x_i) - H_{\mathsf{idx}}(x_i)$, then by the correctness of OKVS.Decode and by Equation (21) in Figure 19, we have $y^{(i)}_0 = y^{(i)}_1$. The concatenation of these equal values would also be equal, hence the correctness for yes-instances.

Correctness for no-instances: For a no-instance prefix $\vec{x}$, by definition there exists a dimension i where x_i is not a prefix of any value in $[q_i - \delta, q_i + \delta]$, which implies that $x_i \notin K^{(i)}$. Let $y^{(i)}_{\mathsf{idx}} = \mathsf{OKVS.Decode}(k^{(i)}_{\mathsf{idx}}, x_i) - H_{\mathsf{idx}}(x_i)$, we can show that by the Strong OKVS Extraction Conjecture (Conjecture 2.1 in [RS21]), with high probability we have $y^{(i)}_0 \neq y^{(i)}_1$ as follows. Looking ahead, this would imply rejection in the Equal property test.

Let $KV^{(i)}_0$ and $KV^{(i)}_1$ denote the OKVS encodings for dimension i as in Equation (23) (Figure 19), then by the linearity of OKVS, their subtraction $KV^{(i)} = KV^{(i)}_0 - KV^{(i)}_1$ is itself a valid OKVS encoding, which can be viewed as an encoding of the key–value list $\{(x, H_0(x) - H_1(x)) : x \in K^{(i)}\}$. Since $H_0(x) - H_1(x)$ is a random oracle, this is the necessary condition for the Strong OKVS Extraction Conjecture, which implies that with high probability, we have $\mathsf{OKVS.Decode}(KV^{(i)}, x_i) \neq H_0(x_i) - H_1(x_i)$ when $x_i \notin K^{(i)}$. Hence, we have $y^{(i)}_0 \neq y^{(i)}_1$ with high probability, which means the concatenation of all dimensions y_0 will also be different from y_1, which leads to rejection in Equal property testing.

Privacy: The proof follows directly from the security of the underlying OKVS scheme. Specifically, for each dimension we invoke the simulator guaranteed by [RS21], which ensures indistinguishability of the simulated view from the real execution. Since our construction composes these invocations in a black-box manner, the overall simulator satisfies the requirements of Definition 4. This concludes the proof. □

A.2 OKVS-based PB-FSS for L_p

Theorem 4 (Security of OKVS-based PB-FSS for L_p)**.** *Given* OKVS(Encode, Decode) *(see Definition 2), the protocol in Figure 19 realizes a* PB-FSS *(or* PB-iFSS*) for the set family $\mathcal{S}_{p,d,\delta}$ and property* Bounded$_{\delta^p}$ *over the domain $\mathcal{D} = (\{0,1\}^{h_1})^d$ that satisfies the properties in Definition 4.*

$\underline{\mathsf{Share}(1^\kappa, L_\infty, (q_1, \ldots, q_d), \delta, \mathbb{G})}$:

1. For each $i = 1, \ldots, d$:
   - Let our interval of interest be $[\alpha_i, \beta_i] = [q_i - \delta, q_i + \delta] \cap [0, 2^{h_1} - 1]$.
   - Prepare the set of keys:
   $$K^{(i)} = \begin{cases} \text{the interval } \{x \mid x \in [\alpha_i, \beta_i]\}, & \text{for PB-FSS,} \\ \text{prefixes in the interval } \{x \mid \exists \chi \in [\alpha_i, \beta_i] \wedge x \prec \chi\}, & \text{for PB-iFSS.} \end{cases} \tag{20}$$
   Pad with dummy strings to for $|K^{(i)}|$ to be exactly $2\delta + 1$ for PB-FSS, or exactly $2\delta + 2h_1$ for PB-iFSS.
   - For each key $x \in K^{(i)}$, prepare a random pair $v_{x,0}, v_{x,1}$ such that:
     - If $\mathsf{dist} = L_\infty$:
     $$v_{x,0} - v_{x,1} = 0. \tag{21}$$
     - If $\mathsf{dist} = L_p$:
     $$v_{x,0} + v_{x,1} = \begin{cases} |x - q_i|^p, & \text{for PB-FSS,} \\ \min_{x \prec \chi, \chi \in \{0,1\}^{h_1}} |\chi - q_i|^p, & \text{for PB-iFSS.} \end{cases} \tag{22}$$
   - Prepare two lists of key-value pairs, each containing subtractive shares for one server:
   $$KV^{(i)}_{\mathsf{idx}} \leftarrow \left\{ (x, v_{x,\mathsf{idx}} + H_{\mathsf{idx}}(x)) \mid x \in K^{(i)} \right\}. \tag{23}$$
   - Obtain $k^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{OKVS.Encode}(KV^{(i)}_{\mathsf{idx}})$ for $\mathsf{idx} = 0, 1$.
2. **Output:** $k_{\mathsf{idx}} = (k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}})$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, \mathsf{dist}, k_{\mathsf{idx}}, (x_1, \ldots, x_d))}$:

1. Parse key $\left(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}\right) \leftarrow k_{\mathsf{idx}}$.
2. For each $i = 1, \ldots, d$:
   - Decode the OKVS vector $k^{(i)}_{\mathsf{idx}}$ on x_i and obtain:
   $$y^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{OKVS.Decode}(k^{(i)}_{\mathsf{idx}}, x_i) - H_{\mathsf{idx}}(x_i).$$
3. **Output:**
   - If $\mathsf{dist} = L_\infty$: $y_{\mathsf{idx}} = \left(y^{(1)}_{\mathsf{idx}} || \ldots || y^{(d)}_{\mathsf{idx}}\right)$.
   - If $\mathsf{dist} = L_p$: $y_{\mathsf{idx}} = \sum_{i=1}^{d} y^{(i)}_{\mathsf{idx}}$.

Figure 19: OKVS-based PB-FSS (and PB-iFSS) construction for L_p.

Proof. **Correctness for yes-instances:** If a point $\vec{x}$ is in the L_p ball of radius δ with center $\vec{q}$, then it is also in the L_∞ ball of radius δ with center $\vec{q}$. Hence, in every dimension i, we have x_i lies in the interval $[q_i - \delta, q_i + \delta]$, and calling $\mathsf{OKVS.Decode}(k^{(i)}_{\mathsf{idx}}, x_i)$ would return secret shares for $|x_i - q_i|^p$. Adding up all dimension, we obtain the L_p distance to the power of p, which would pass the Prop check.

Correctness for no-instances: The aggregated shares in this case would either be secret shares of a random value, or a secret share of exactly $L_p(\vec{x}, \vec{q})^p$, which would fail the check of $\mathsf{Bounded}_\mu$.

Privacy: The privacy of the protocol follows directly from the security of OKVS schemes. □

A.3 Bound for the number of prefixes in a range

Given a range $[\alpha, \beta] \subset [0, 2^{h_1}]$, the number of prefixes in $[\alpha, \beta]$ is largest when $\alpha < 2^{h_1-1}$ and $\beta \geq 2^{h_1-1}$. We can decompose the range into $[\alpha, 2^{u-1} - 1] \cup [2^{u-1}, \beta]$ and count prefixes in each half.

For $[2^{h_1-1}, \beta]$, the number of prefixes is $h_1 + f(\beta - 2^{h_1-1})$, where $\{f(i)\}_{i=0}^{\infty} = 0, 1, 3, 4, 7, 8, 10, 11, \ldots$ and $f(x) = \sum_{i=1}^{x}(v_2(i) + 1) = x + \sum_{i=1}^{x} v_2(i) = x + \lfloor x/2 \rfloor + \lfloor x/4 \rfloor + \ldots$.

The same argument applies to $[\alpha, 2^{h_1-1} - 1]$, giving the total number of prefixes in $[\alpha, \beta]$:

$$\begin{aligned} &2h_1 + \sum_{i=1}^{\infty} \left\lfloor \frac{2^{h_1-1} - 1 - \alpha}{2^i} \right\rfloor + \sum_{i=1}^{\infty} \left\lfloor \frac{\beta - 2^{h_1-1}}{2^i} \right\rfloor \\ &\leq 2h_1 + 2(2^{h_1-1} - 1 - \alpha) - 1 + 2(\beta - 2^{h_1-1}) - 1 \\ &\leq 2h_1 + 2\beta - 2\alpha - 4. \end{aligned}$$

Applying this to $[q_i - \delta, q_i + \delta]$, the number of prefixes is at most $2h_1 + 2\delta$.

B DCF-Based Construction Details of PB-FSS and PB-iFSS

In this appendix, we provide the full implementation of the DCF-based construction for PB-FSS for the non-incremental case and PB-iFSS in the incremental case. The full construction for the non-incremental case is almost the same as in Section 4.4, only handle the edge cases when q_i is near the boundary of the domain more carefully. The construction for the incremental PB-iFSS is a bit more trickier, as inside an interval, handling prefixes on the "left" of this interval and on the "right" of this interval requires non black-box modification of the GGM-tree based DCF construction [BGI16, GGM24]. We call these variants LDCF and RDCF, that "share" the functions f^{LC} and f^{RC} that had been defined in Section 4.4. We omit the detailed constructions for LDCF and RDCF in this paper, as they are straight-forward modifications of the original DCF construction, hence security also easily follows. More concretely, we still follow the GGM-based approach to construct incremental DCF, only modifying the exact payload in each level of the tree, which programs the correlation between payloads, but does not affect the pseudorandomness.

B.1 DCF-based PB-iFSS for L_∞

In the proof of security below, we only deal with the incremental case, with security based on the variants LDCF and RDCF of DCF. The incremental case can be seen as the generalization of the non-incremental case, using the correctness and security of LDCF and RDCF instead of DCF, to support evaluation on prefixes.

Furthermore, the protocol presented in Figure 20 uses *correlated* calls to RDCF.Share and LDCF.Share, where the splitting points have a fixed difference being 2δ. We hence need to prove that a pair of correlated calls of RDCF.Share and LDCF.Share is simulatable, using non black-box knowledge about the LDCF and RDCF constructions. For the GGM-based construction without any optimization, this can be done by assuming PRG. We describe the correlated sharing procedure that is simulatable in Lemma **??**, before giving the proof for the PB-iFSS protocol in Theorem 5.

Theorem 5 (Security of DCF-based PB-iFSS for L_∞)**.** *Assuming length-doubling PRG, the protocol in Figure 20 realizes a* PB-FSS *(or* PB-iFSS*) scheme for the set family $\mathcal{S}_{\infty,d,\delta}$ and property* Equal *over the domain $\mathcal{D} = \left(\{0,1\}^{h_1}\right)^d$ that satisfies Definition 4.*

Proof. In the following, we prove the correctness of the PB-iFSS construction.

Correctness for yes-instances: Given $\vec{x} = (x_1, \ldots, x_d)$ such that the optimistic evaluation of x is less than or equal to δ, which means, for every dimension $i = 1, \ldots, d$, there exists χ_i such that $x_i \prec \chi_i$ and $|\chi_i - \alpha_i| \leq \delta$. By the correctness of LDCF and RDCF, we have $y_0^{(i)} \oplus y_1^{(i)} = 0 \oplus 0 = 0$, for all dimensions $i = 1, \ldots, d$. This directly implies that $y_0 = y_1$, hence $\mathsf{Prop}(y_0, y_1) = 1$.

$\underline{\mathsf{Share}(1^\kappa, L_\infty, (q_1, \ldots, q_d), \delta, \mathbb{G}) :}$

1. For each $i = 1, \ldots, d$:
   - Let our interval of interest be $[\alpha_i, \beta_i] = [q_i - \delta, q_i + \delta] \cap [0, 2^{h_1} - 1]$.
   - Generate DCF keys to share the interval $[\alpha_i, \beta_i]$:
     - For PB-FSS:

$$(k^{(i)}_{0,0}, k^{(i)}_{1,0}) \leftarrow \mathsf{DCF.Share}(\alpha_i - 1, 1, 0), \tag{24}$$
$$(k^{(i)}_{0,1}, k^{(i)}_{1,1}) \leftarrow \mathsf{DCF.Share}(\beta_i, 0, 1), \tag{25}$$

     - For PB-iFSS:

$$(k^{(i)}_{0,0}, k^{(i)}_{1,0}) \leftarrow \mathsf{RDCF.Share}(\alpha_i, 1, 0), \tag{26}$$
$$(k^{(i)}_{0,1}, k^{(i)}_{1,1}) \leftarrow \mathsf{LDCF.Share}(\beta_i, 0, 1), \tag{27}$$

   - Obtain $k^{(i)}_{\mathsf{idx}} \leftarrow \left(k^{(i)}_{\mathsf{idx},0} \| k^{(i)}_{\mathsf{idx},1}\right)$ for $\mathsf{idx} = 0, 1$.
2. **Output:** $k_{\mathsf{idx}} = \left(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}\right)$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, L_\infty, k_{\mathsf{idx}}, (x_1, \ldots, x_d)):}$

1. Parse key $(k^{(1)}_{\mathsf{idx}} \| \ldots \| k^{(d)}_{\mathsf{idx}}) \leftarrow k_{\mathsf{idx}}$.
2. For each $i = 1, \ldots, d$:
   - Parse $\left(k^{(i)}_{\mathsf{idx},0} \| k^{(i)}_{\mathsf{idx},1}\right) \leftarrow k^{(i)}_{\mathsf{idx}}$.
   - Evaluate the two keys on x_i, we only use Eval to denote the evaluation algorithm:

$$y^{(i)}_{\mathsf{idx}} \leftarrow \mathsf{Eval}(\mathsf{idx}, k^{(i)}_{\mathsf{idx},0}, x_i) \oplus \mathsf{Eval}(\mathsf{idx}, k^{(i)}_{\mathsf{idx},1}, x_i).$$

3. **Output:** $y_{\mathsf{idx}} = \left(y^{(1)}_{\mathsf{idx}} \| \ldots \| y^{(d)}_{\mathsf{idx}}\right)$.

Figure 20: DCF-based PB-FSS: Full L_∞ scheme.

Correctness for no-instances: Given $\vec{x} = (x_1, \ldots, x_d)$ such that the optimistic L_∞ distance of x from q is larger than δ, it means there exists a dimension i such that x_i is more than δ away from q_i. By the correctness of LDCF and RDCF, we have $y^{(i)}_0 \oplus y^{(i)}_1$ to be either $1 \oplus 0 = 1$ or $0 \oplus 1 = 1$. Hence $y_0 \oplus y_1$ has digit 1 at position i, and $\mathsf{Prop}(y_0, y_1) = 0$.

Privacy: By the privacy definition in Definition 3, there exists a PPT simulator $\mathsf{Sim}_{\mathsf{LDCF}}$ for LDCF and a PPT simulator $\mathsf{Sim}_{\mathsf{RDCF}}$ for RDCF. The simulator Sim for the PB-iFSS construction for L_∞ distance in Figure 20 then consists of multiple calls to $\mathsf{Sim}_{\mathsf{LDCF}}$ and $\mathsf{Sim}_{\mathsf{RDCF}}$:

$\underline{\mathsf{Sim}(1^\kappa, \mathsf{idx}, d, h_1, \mathbb{G}):}$

- For each $i = 1, \ldots, d$:
  - Set $k^{(i)}_{\mathsf{idx},0} \leftarrow \mathsf{Sim}_{\mathsf{RDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$.

- Set $k^{(i)}_{\mathsf{idx},1} \leftarrow \mathsf{Sim}_{\mathsf{LDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$.
- Obtain $k^{(i)}_{\mathsf{idx}} \leftarrow \left(k^{(i)}_{\mathsf{idx},0} \| k^{(i)}_{\mathsf{idx},1}\right)$.

- Output $k_{\mathsf{idx}} = \left(k^{(1)}_{\mathsf{idx}} \| \| \ldots \| k^{(d)}_{\mathsf{idx}}\right)$.

We will prove that, for any $\mathsf{idx} \in \{0,1\}$, any input $\vec{q} = (q_1, q_2, \ldots, q_d) \in \left(\{0,1\}^{h_1}\right)^d$, and any threshold δ, the simulated key is indistinguishable from a honestly generated key:

$$\boxed{\begin{array}{l}(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, L_\infty, (q_1, \ldots, q_d), \delta, \mathbb{G}) \\ \text{return } k_{\mathsf{idx}}\end{array}} \cong_\kappa \mathsf{Sim}(1^\kappa, \mathsf{idx}, d, h_1, \mathbb{G}).$$

The proof is routine. We will prove indistinguishability through a sequence of hybrids, which simply replace each LDCF or RDCF call into their corresponding simulator.

- **Hybrid** 0**:** Run the Share algorithm honestly, following Figure 20.
- **Hybrid** 1**:** Same as Hybrid 0, except that $k^{(1)}_{\mathsf{idx},0}$ is obtained from $\mathsf{Sim}_{\mathsf{RDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$. Hybrid 1 is indistinguishable from Hybrid 0 due to the security of RDCF.
- **Hybrid** 2**:** Same as Hybrid 1, except that $k^{(1)}_{\mathsf{idx},1}$ is obtained from $\mathsf{Sim}_{\mathsf{LDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$. Hybrid 2 is indistinguishable from Hybrid 1 due to the security of LDCF.
- ...
- **Hybrid** $2d-1$**:** Same as Hybrid $2d-2$, except that $k^{(d)}_{\mathsf{idx},0}$ is obtained from $\mathsf{Sim}_{\mathsf{RDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$. Hybrid $2d-1$ is indistinguishable from Hybrid $2d-2$ due to the security of RDCF.
- **Hybrid** $2d$**:** Same as Hybrid $2d-1$, except that $k^{(d)}_{\mathsf{idx},1}$ is obtained from $\mathsf{Sim}_{\mathsf{LDCF}}(1^\kappa, \mathsf{idx}, h_1, \mathbb{G})$. Hybrid $2d$ is indistinguishable from Hybrid $2d-1$ due to the security of LDCF. Hybrid $2d$ is also the simulator $\mathsf{Sim}(1^\kappa, \mathsf{idx}, d, h_1, \mathbb{G})$.

□

B.2 DCF-based PB-iFSS for L_p

We present the DCF-based PB-FSS (and PB-iFSS) for the L_p distance metric in Figure 21. As the incremental case is a generalization of the non-incremental case, we focus on proving the correctness and security of the incremental PB-iFSS protocol. The security of the protocol again requires a simulator for correlated keys, this time being two pairs of LDCF and RDCF, with splitting points being either δ apart or equal. The simulator can be proved in a non black-box way, dependent on the GGM-based construction of LDCF and RDCF, and we again assume PRG for the omitted proof.

The security of the L_p scheme then follows easily, while the correctness requires careful casework, all of which will be shown in Theorem 6.

Theorem 6 (Security of DCF-based PB-iFSS for L_p)**.** *Assuming length-doubling PRG, the protocol in Figure 21 realizes a* PB-FSS *(or* PB-iFSS*) scheme for the set family $\mathcal{S}_{p,d,\delta}$ and property* $\mathsf{Bounded}_{\delta^p}$ *over the domain $\mathcal{D} = \left(\{0,1\}^{h_1}\right)^d$ that satisfies Definition 4.*

We are now ready to state the correctness of DCF-based PB-iFSS for the L_p distance metric.

$\underline{\mathsf{Share}(1^\kappa, L_p, (q_1, \ldots, q_d), \delta, \mathbb{G}) :}$

1. For each $i = 1, \ldots, d$:

   - Let our interval of interest be $[\alpha_i, \beta_i] = [q_i - \delta, q_i + \delta] \cap [0, 2^{h_1} - 1]$.
   - Prepare the following payload vectors:

$$\mathbf{a}_i = \begin{bmatrix}(-1)^p & (-1)^{p-1} q_i & \ldots & (-1)^0 q_i^p\end{bmatrix},$$
$$\mathbf{b}_i = \begin{bmatrix}(-1)^0 & (-1)^1 q_i & \ldots & (-1)^p q_i^p\end{bmatrix}.$$

     Also let $\mathbf{m}$ denote a max vector (in the last entry) and $\mathbf{n}$ denote a null vector:

$$\mathbf{m} = \begin{bmatrix}0 & 0 & \ldots & 0 & \delta^p + 1\end{bmatrix},$$
$$\mathbf{n} = \begin{bmatrix}0 & 0 & \ldots & 0 & 0\end{bmatrix}.$$

   - Generate DCF keys to share the L_p distance:
     - For PB-FSS:

$$(k_{0,0}^{(i)}, k_{1,0}^{(i)}) \leftarrow \mathsf{DCF.Share}(\alpha_i - 1, \mathbf{m} - \mathbf{a}, \mathbf{n}), \qquad (k_{0,1}^{(i)}, k_{1,1}^{(i)}) \leftarrow \mathsf{DCF.Share}(q_i - 1, \mathbf{a}, \mathbf{n}), \tag{28}$$
$$(k_{0,2}^{(i)}, k_{1,2}^{(i)}) \leftarrow \mathsf{DCF.Share}(q_i, \mathbf{n}, \mathbf{b}), \qquad (k_{0,3}^{(i)}, k_{1,3}^{(i)}) \leftarrow \mathsf{DCF.Share}(\beta_i, \mathbf{n}, \mathbf{m} - \mathbf{b}). \tag{29}$$

     - For PB-iFSS:

$$(k_{0,0}^{(i)}, k_{1,0}^{(i)}) \leftarrow \mathsf{RDCF.Share}(\alpha_i, \mathbf{m} - \mathbf{a}, \mathbf{n}), \qquad (k_{0,1}^{(i)}, k_{1,1}^{(i)}) \leftarrow \mathsf{RDCF.Share}(q_i, \mathbf{a}, \mathbf{n}), \tag{30}$$
$$(k_{0,2}^{(i)}, k_{1,2}^{(i)}) \leftarrow \mathsf{LDCF.Share}(q_i, \mathbf{n}, \mathbf{b}), \qquad (k_{0,3}^{(i)}, k_{1,3}^{(i)}) \leftarrow \mathsf{LDCF.Share}(\beta_i, \mathbf{n}, \mathbf{m} - \mathbf{b}). \tag{31}$$

   - Obtain $k_{\mathsf{idx}}^{(i)} \leftarrow \left(k_{\mathsf{idx},0}^{(i)} \| k_{\mathsf{idx},1}^{(i)} \| k_{\mathsf{idx},2}^{(i)} \| k_{\mathsf{idx},3}^{(i)}\right)$ for $\mathsf{idx} = 0, 1$.

2. **Output:** $k_{\mathsf{idx}} = \left(k_{\mathsf{idx}}^{(1)} \| \ldots \| k_{\mathsf{idx}}^{(d)}\right)$ for $\mathsf{idx} = 0, 1$.

$\underline{\mathsf{Eval}(\mathsf{idx}, L_p, k_{\mathsf{idx}}, (x_1, \ldots, x_d)):}$

1. Parse key $(k_{\mathsf{idx}}^{(1)} \| \ldots \| k_{\mathsf{idx}}^{(d)}) \leftarrow k_{\mathsf{idx}}$.
2. for each $i = 1, \ldots, d$:

   - Parse $\left(k_{\mathsf{idx},0}^{(i)} \| k_{\mathsf{idx},1}^{(i)} \| k_{\mathsf{idx},2}^{(i)} \| k_{\mathsf{idx},3}^{(i)}\right) \leftarrow k_{\mathsf{idx}}^{(i)}$.
   - Evaluate the L_p distance components from the left of q_i and from the right of $q_i i$:

$$\mathbf{y}^{(i)} \leftarrow \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx},0}^{(i)}, x_i) + \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx},1}^{(i)}, x_i), \qquad \mathbf{z}^{(i)} \leftarrow \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx},2}^{(i)}, x_i) + \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx},3}^{(i)}, x_i).$$

   - Pad x_i into $x_{\min}^{(i)} = x_i 00 \ldots 0$ and $x_{\max}^{(i)} = x_i 11 \ldots 1$, both have length h_1.
   - Obtain the secret share of L_p distance:

$$y_{\mathsf{idx}}^{(i)} \leftarrow \sum_{j=0}^{p} \binom{p}{j} \left((x_{\max}^{(i)})^{p-j} \mathbf{y}_j^{(i)} + (x_{\min}^{(i)})^{p-j} \mathbf{z}_j^{(i)}\right).$$

3. **Output:** $y_{\mathsf{idx}} = \sum_{i=1}^{d} y_{\mathsf{idx}}^{(i)}$.

Figure 21: DCF-based PB-iFSS: Full L_p scheme

Proof. By the correctness of LDCF and RDCF, we summarize the value of $y_0^{(i)} + y_1^{(i)}$ for each dimension $i \in [d]$, when evaluating on a point x_i as follows:

$$y^(i) = \begin{cases} MAX & \text{if } x_i < \alpha_i, \\ \sum_{j=0}^{p} (-1)^{p-j} q_i^j x_{i,\max}^{p-j} = (q_i - x_{i,\max})^p & \text{if } \alpha_i \le x_i < q_i, \\ 0 & \text{if } x_i \prec q_i, \\ \sum_{j=0}^{p} (-1)^{j} q_i^j x_{i,\min}^{p-j} = (q_i - x_{i,\min})^p & \text{if } q_i < x_i \le \beta_i, \\ MAX & \text{if } x_i > \beta_i. \end{cases}$$

In short, for each dimension $i = 1, \ldots, d$, we have $y^{(i)}$ is either the optimistic L_p distance of x_i from α_i or $\delta^p + 1$:

$$y_0^{(i)} + y_1^{(i)} = \min\left\{ |\chi - \alpha_i|^p : x_i \prec \chi \right\} \cup \left\{ \delta^p + 1 \right\}. \tag{32}$$

Correctness for yes-instances: Given $\vec{x} = (x_1, \ldots, x_d)$ such that the optimistic evaluation of x is less than or equal δ, which implies that for every dimension $i = 1, \ldots, d$, there exists χ_i such that $x_i \prec \chi_i$ and $|\chi_i - \alpha_i|^p \le \delta^p \iff |\chi_i - \alpha_i| \le \delta$.

It follows that, for each dimension $i = 1, \ldots, d$, the value x_i is at most δ away from α_i. We conclude that:

$$y_0 + y_1 = \min\left\{ \sum_{i=1}^{d} |\chi_i - \alpha_i|^p : x \prec \chi \right\} \le \delta^p$$

.

Correctness for no-instances: Given $\vec{x} = (x_1, \ldots, x_d)$ such that the optimistic evaluation of x is greater than δ, there are two cases to consider. If for every dimension $i = 1, \ldots, d$, the value x_i is at most δ away from q_i, then the value of $y_0 + y_1$ is exactly the same as the analysis for yes-instance. Otherwise, there exists a dimension i such that the value x_i is more than δ away from q_i, which means $y_0^{(i)} + y_1^{(i)} = \delta^p + 1$. Hence $y_0 + y_1 \ge y_0^{(i)} + y_1^{(i)} = \delta^p + 1$.

Privacy: The simulator $\mathsf{Sim}(1^\kappa, \mathsf{idx}, d, h_1)$ for Share, again, consists of multiple calls to the correlated DCF simulator $\mathsf{Sim}_{\mathsf{PCorDCF}}$. The simulator for party idx is described as follows:

- For each $i = 1, \ldots, d$, set $k_{\mathsf{idx}}^{(i)} \leftarrow \mathsf{Sim}_{\mathsf{PCorDCF}}(1^\kappa, \mathsf{idx}, d, h_1)$.
- Return $k_{\mathsf{idx}} \leftarrow \left(k_{\mathsf{idx}}^{(1)} \| \ldots \| k_{\mathsf{idx}}^{(d)} \right)$.

The indistinguishability again follows a simple sequence of hybrids: [phuoc: todo] □

C Deferred Protocols and Proofs for Private Property Testing

In this section, we provide deferred protocols and security proofs from Section 5.

C.1 2PC-Based Approach for L_∞

Theorem 7 (Security of $\Pi_{\mathsf{Equal\text{-}2PC}}$)**.** *Protocol $\Pi_{\mathsf{Equal\text{-}2PC}}$ (Figure 8) securely realizes the ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with property Equal in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model against a static, semi-honest adversary that corrupts at most one server.*

Proof. **Correctness.** The protocol evaluates the circuit $C(y_0, y_1) = \mathbb{1}_{y_0 = y_1}$. Ideally, $\mathcal{F}_{\mathsf{PropTest}}$ outputs secret shares of 1 if $y_0 = y_1$ and 0 otherwise. Assuming the underlying $\mathcal{F}_{\mathsf{2PC}}$ functionality computes the circuit correctly, the protocol outputs the correct bit.

Security. The security of $\Pi_{\mathtt{Equal-2PC}}$ (Figure 8) securely realizes the ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model, assuming the underlying 2PC protocol securely evaluates circuits in the presence of semi-honest adversaries. □

C.2 2PC-Based Approach for L_p

We securely evaluate whether a secret-shared value $y = y_0 + y_1 \bmod 2^{h_2}$ is at most a public threshold μ, by first determining whether the addition overflows modulo 2^{h_2} and then performing a comparison appropriate to the overflow case. We describe the complete protocol formally in Figure 22 and break down the case analysis below:

Case 1: No Overflow ($y_0 + y_1 < 2^{h_2}$). To check whether $y_0 + y_1 \leq \mu$: if $y_1 \leq \mu$, the parties compare y_0 with $\mu - y_1$; otherwise, $y_0 + y_1 > \mu$ and the circuit outputs 0. This is implemented as $\hat{b} \wedge \mathbb{1}_{y_0 \leq z}$, where $\mathcal{S}_0$ inputs y_0 and $\mathcal{S}_1$ computes and inputs $\hat{b} = \mathbb{1}_{\{y_1 \leq \mu\}}$ and $z = (\mu - y_1) \bmod 2^{h_2}$.

Case 2: Overflow ($y_0 + y_1 \geq 2^{h_2}$). Let $y = (y_0 + y_1) \bmod 2^{h_2}$. The check $y \leq \mu$ is equivalent to $y_0 + y_1 \leq 2^{h_2} + \mu$. If $y_1 \leq \mu$, then $y_0 + y_1 < 2^{h_2} + \mu$ always holds (output 1); otherwise, compare y_0 with $2^{h_2} + \mu - y_1$. This is implemented as $\hat{b} \vee \mathbb{1}_{y_0 \leq z}$, where $\mathcal{S}_0$ inputs y_0 and $\mathcal{S}_1$ computes and inputs $\hat{b} = \mathbb{1}_{\{y_1 \leq \mu\}}$ and $z = (2^{h_2} + \mu - y_1) \bmod 2^{h_2}$.

Inputs:

- Party $\mathcal{S}_0$ holds an input share $y_0 \in \{0,1\}^{h_2}$.
- Party $\mathcal{S}_1$ holds an input share $y_1 \in \{0,1\}^{h_2}$.
- Private property threshold $\mu \in \mathbb{Z}_{2^{h_2}}$.

Protocol:

1. Party $\mathcal{S}_0$ and $\mathcal{S}_1$ invoke the $\mathcal{F}_{\mathsf{2PC}}$ functionality on the following circuit:
   (a) $\mathcal{S}_0$ inputs y_0.
   (b) $\mathcal{S}_1$ inputs y_1.
   (c) Two servers jointly compute the following values:
   - $z \leftarrow (\mu - y_1) \mod 2^{h_2}$
   - $u \leftarrow \mathbb{1}_{y_1 \leq \mu}$
   - $c \leftarrow \mathbb{1}_{y_0 \leq 2^{h_2} - 1 - y_1}$ // Checks whether $y_0 + y_1$ overflows
   - $c_0 \leftarrow c \wedge u \wedge \mathbb{1}_{y_0 \leq z}$ // When $y_0 + y_1$ not overflow, check $y_1 \leq \mu$ and $y_0 \leq \mu - y_1$
   - $c_1 \leftarrow \neg c \wedge (u \vee \mathbb{1}_{y_0 \leq z})$ When $y_0 + y_1$ overflow, check either $y_0 \leq \mu - y_1$ or $y_1 \geq \mu$
   - $b \leftarrow c_0 \vee c_1$
2. $\mathcal{S}_0$ samples $b_0 \in \mathbb{Z}_{2^{h_3}}$, then compute in $\mathcal{F}_{\mathsf{2PC}}$ with $\mathcal{S}_1$ the value $b_1 \leftarrow b - b_0 \mod 2^{h_3}$[5]. $\mathcal{S}_1$ receives b_1.

Figure 22: Protocol $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathsf{2PC}}$ for $\mathcal{F}_{\mathsf{PropTest}}$.

The final step is to determine which case applies: overflow or no overflow. To do so, the parties compare: $y_0 \leq 2^{h_2} - 1 - y_1$. If this inequality holds, then $y_0 + y_1 < 2^{h_2}$ (i.e., no overflow). Otherwise, overflow has occurred. The full circuit combines these two branches accordingly.

Theorem 8 (Security of $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathsf{2PC}}$)**.** *Protocol $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathsf{2PC}}$ (Figure 22) securely realizes the ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with property* Equal *in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model against a static, semi-honest adversary that can corrupt at most one server.*

Proof. **Correctness.** We analyze the correctness of the circuit in Figure 22:

- **Case 1: No overflow** ($y_0 + y_1 < 2^{h_2}$): This means $c = 1$, hence $c_1 = 0$. The output value b is then only dependent on c_0. Furthermore, we note that $y = y_0 + y_1$.
  - If $y_1 \leq \mu$, then $y \leq \mu \iff y_0 \leq \mu - y_1$.

– If $y_1 > \mu$, then $y > \mu$ and the output is 0.

- **Case 2: Overflow occurs** ($y_0 + y_1 \geq 2^{h_2}$): This means $c = 0$, hence $c_0 = 0$. The output value b is then only dependent on c_1. Furthermore, we note that $y = y_0 + y_1 - 2^{h_2}$.
  - If $y_1 \leq \mu$, then $y < 2^{h_2}$ and $y \leq \mu$ always holds — output is 1.
  - If $y_1 > \mu$, then $y \leq \mu \iff y_0 \leq 2^{h_2} + \mu - y_1$.

Security. The protocol $\Pi_{\mathsf{bounded}_\mu\text{-}\mathsf{2PC}}$ (Figure 22) securely realizes the ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model, assuming the underlying 2PC protocol securely evaluates circuits in the presence of semi-honest adversaries. □

C.3 Dealer-Based Approach for L_∞

We prove security in the $\mathcal{F}_{\mathsf{FSS-Gen}}$-hybrid model, where the dealer's preprocessing is replaced by a call to $\mathcal{F}_{\mathsf{FSS-Gen}}$ (Figure 3). For $\Pi_{\mathsf{Equal\text{-}Dealer}}$, the servers invoke $\mathcal{F}_{\mathsf{FSS-Gen}}$ with domain $\{0,1\}^{h_2}$, payload group $\mathbb{Z}_{2^{h_3}}$, type DPF, and payloads $a = 1$, $b = 0$; server $\mathcal{S}_{\mathsf{idx}}$ receives $(\vec{v}_{\mathsf{idx}}, \alpha_{\mathsf{idx}})$. The mask is $r = \alpha$, with $r_0 = \alpha_0$ and $r_1 = -\alpha_1 \bmod 2^{h_2}$ so that $r = r_0 - r_1$, and $\mathsf{DPF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \tilde{y})$ is replaced by $\vec{v}_{\mathsf{idx}}[\tilde{y}]$.

Theorem 9 (Security of $\Pi_{\mathsf{Equal\text{-}Dealer}}$)**.** *Protocol $\Pi_{\mathsf{Equal\text{-}Dealer}}$ (Figure 9) securely realizes $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with property Equal in the $\mathcal{F}_{\mathsf{FSS-Gen}}$-hybrid model against a static, semi-honest adversary that corrupts at most one server.*

Proof. **Correctness.** Since $r_0 - r_1 = \alpha$, the servers reconstruct $\tilde{y} = \tilde{y}_0 - \tilde{y}_1 = (y_0 - y_1) + \alpha \pmod{2^{h_2}}$. Hence $b_0 + b_1 = \vec{v}[\tilde{y}] = \mathbb{1}_{\tilde{y}=\alpha} = \mathbb{1}_{y_0=y_1}$.

Security. We give the simulator Sim for a corrupted $\mathcal{S}_0$ with input y_0; the case of a corrupted $\mathcal{S}_1$ is symmetric.

1. Sim simulates $\mathcal{F}_{\mathsf{FSS-Gen}}$ for $\mathcal{A}$ and obtains the shares $(\vec{v}_0, \alpha_0)$ of $\mathcal{S}_0$. It sets $r_0 = \alpha_0$ and $\tilde{y}_0 = y_0 + r_0$.
2. Sim samples $\tilde{y}_1 \xleftarrow{\$} \mathbb{Z}_{2^{h_2}}$ and sends it to $\mathcal{A}$ as the message of $\mathcal{S}_1$.
3. Sim computes $\tilde{y} = \tilde{y}_0 - \tilde{y}_1$ and $b_0 = \vec{v}_0[\tilde{y}]$, and sends y_0 and the output share b_0 to $\mathcal{F}_{\mathsf{PropTest}}$.

In the hybrid execution, $\tilde{y}_1 = y_1 - \alpha + \alpha_0$, where α is uniform and hidden from $\mathcal{A}$, so $\tilde{y}_1$ is uniform and independent of the rest of the view of $\mathcal{A}$, as in the simulation. Moreover, by correctness, the honest $\mathcal{S}_1$ outputs $\vec{v}_1[\tilde{y}] = \vec{v}[\tilde{y}] - \vec{v}_0[\tilde{y}] = \mathbb{1}_{y_0=y_1} - b_0$. This is exactly the share $b - b_0$ that $\mathcal{F}_{\mathsf{PropTest}}$ gives to $\mathcal{S}_1$ in the ideal execution. Hence the joint distribution of the view of $\mathcal{A}$ and the honest output is identical in both executions. □

C.4 Dealer-Based Approach for L_p

In the L_p setting, the servers test whether $y = y_0 + y_1 \bmod 2^{h_2}$ satisfies $y \leq \mu$ by opening a masked value and evaluating a single DCF key from the dealer, following the approach of [BCG+21]. The dealer samples a random mask $r \in \mathbb{Z}_{2^{h_2}}$, gives the servers additive shares of r, and gives them DCF keys for the comparison function $\mathbb{1}_{z<r}$. The servers then open $\tilde{y} = y + r \bmod 2^{h_2}$, and let $\tilde{y}' = \tilde{y} - \mu - 1 \bmod 2^{h_2}$. Comparing representatives in $[0, 2^{h_2})$, we have

$$\mathbb{1}_{y\leq\mu} = \mathbb{1}_{\tilde{y}'<r} - \mathbb{1}_{\tilde{y}<r} + \mathbb{1}_{\tilde{y}\leq\mu}. \tag{33}$$

Intuitively, the first two terms are the carries of the additions $\big((y - \mu - 1) \bmod 2^{h_2}\big) + r$ and $y + r$ modulo 2^{h_2}, and the last term is the borrow of the subtraction $\tilde{y} - \mu - 1$. Balancing these carries and borrows recovers the borrow of $y - \mu - 1$, which is exactly $\mathbb{1}_{y\leq\mu}$. The first two terms are obtained by evaluating

the same DCF key at the two public points $\tilde{y}'$ and $\tilde{y}$, and the last term is public, so it is added by only one of the servers. Compared to sharing the interval $[r, r+\mu]$ directly, which requires two DCF keys and a case distinction on whether $r+\mu$ wraps around modulo 2^{h_2}, this uses a single key. The resulting protocol $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathtt{Dealer}}$ is shown in Figure 23.

Inputs:

- Party $\mathcal{S}_0$ holds an input share $y_0 \in \{0,1\}^{h_2}$.
- Party $\mathcal{S}_1$ holds an input share $y_1 \in \{0,1\}^{h_2}$.
- Dealer $\mathcal{D}$.
- Public property threshold $\mu \in \mathbb{Z}_{2^{h_2}}$.

Protocol:

1. **Preprocessing:** $\mathcal{D}$ samples $r_0, r_1 \xleftarrow{\$} \mathbb{Z}_{2^{h_2}}$, and lets $r = r_0 + r_1 \bmod 2^{h_2}$ and $\alpha = r - 1 \bmod 2^{h_2}$. $\mathcal{D}$ generates $(k_0, k_1) \leftarrow \mathsf{DCF.Share}(\alpha, 1, 0)$ with payload group $\mathbb{Z}_{2^{h_3}}$, and sends $(k_{\mathsf{idx}}, r_{\mathsf{idx}})$ to $\mathcal{S}_{\mathsf{idx}}$ for $\mathsf{idx} \in \{0,1\}$.
2. **Online Masking:** $\mathcal{S}_0$ sends $\tilde{y}_0 = y_0 + r_0 \bmod 2^{h_2}$ to $\mathcal{S}_1$, and $\mathcal{S}_1$ sends $\tilde{y}_1 = y_1 + r_1 \bmod 2^{h_2}$ to $\mathcal{S}_0$. Both compute $\tilde{y} = \tilde{y}_0 + \tilde{y}_1 \bmod 2^{h_2}$.
3. **Evaluation:** Let $\tilde{y}' = \tilde{y} - \mu - 1 \bmod 2^{h_2}$. For $\mathsf{idx} \in \{0,1\}$, $\mathcal{S}_{\mathsf{idx}}$ computes $e_{\mathsf{idx}} = \mathsf{DCF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \tilde{y}') - \mathsf{DCF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \tilde{y}) \bmod 2^{h_3}$. $\mathcal{S}_0$ outputs $b_0 = e_0 + \mathbb{1}_{\tilde{y} \le \mu} \bmod 2^{h_3}$, and $\mathcal{S}_1$ outputs $b_1 = e_1$. // Only $\mathcal{S}_0$ adds the public correction term.

Figure 23: Protocol $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathtt{Dealer}}$ for $\mathcal{F}_{\mathsf{PropTest}}$.

We again prove security in the $\mathcal{F}_{\mathsf{FSS-Gen}}$-hybrid model. For $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathtt{Dealer}}$, the servers invoke $\mathcal{F}_{\mathsf{FSS-Gen}}$ with domain $\{0,1\}^{h_2}$, payload group $\mathbb{Z}_{2^{h_3}}$, type DCF, and payloads $a = 1$, $b = 0$; server $\mathcal{S}_{\mathsf{idx}}$ receives $(\vec{v}_{\mathsf{idx}}, \alpha_{\mathsf{idx}})$. The mask shares are $r_0 = \alpha_0 + 1$ and $r_1 = \alpha_1$, so that $\alpha = r - 1 \bmod 2^{h_2}$ as in Figure 23, and $\mathsf{DCF.Eval}(\mathsf{idx}, k_{\mathsf{idx}}, z)$ is replaced by $\vec{v}_{\mathsf{idx}}[z]$.

Theorem 10 (Security of $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathtt{Dealer}}$)**.** *Protocol $\Pi_{\mathsf{Bounded}_\mu\text{-}\mathtt{Dealer}}$ (Figure 23) securely realizes $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with property $\mathsf{Bounded}_\mu$ in the $\mathcal{F}_{\mathsf{FSS-Gen}}$-hybrid model against a static, semi-honest adversary that corrupts at most one server.*

Proof. **Correctness.** Let $N = 2^{h_2}$ and $y = y_0 + y_1 \bmod N$, so that $\tilde{y} = y + r \bmod N$; we compare representatives in $[0, N)$. We have $b_0 + b_1 = \vec{v}[\tilde{y}'] - \vec{v}[\tilde{y}] + \mathbb{1}_{\tilde{y} \le \mu}$, where $\vec{v}[z] = \mathbb{1}_{z \le \alpha}$. For every z, $\vec{v}[z] = \mathbb{1}_{z<r} + \mathbb{1}_{r=0}$: if $r \neq 0$ then $\alpha = r - 1$, and if $r = 0$ then $\alpha = N - 1$ and $\vec{v}[z] = 1$. The constant cancels, so $b_0 + b_1 = \mathbb{1}_{\tilde{y}'<r} - \mathbb{1}_{\tilde{y}<r} + \mathbb{1}_{\tilde{y} \le \mu}$. To see that this equals $\mathbb{1}_{y \le \mu}$, let $z = y - \mu - 1 \bmod N$, and write each modular reduction with its carry or borrow:

$$y + r = \tilde{y} + N \cdot \mathbb{1}_{\tilde{y}<r}, \qquad z = y - \mu - 1 + N \cdot \mathbb{1}_{y \le \mu},$$
$$z + r = \tilde{y}' + N \cdot \mathbb{1}_{\tilde{y}'<r}, \qquad \tilde{y}' = \tilde{y} - \mu - 1 + N \cdot \mathbb{1}_{\tilde{y} \le \mu}.$$

Here the first and third equations hold because $y, z, r < N$, and $z + r \equiv \tilde{y}' \pmod N$. Substituting the other three equations into $z + r$ gives $z + r = \tilde{y}' + N \cdot (\mathbb{1}_{\tilde{y}<r} - \mathbb{1}_{\tilde{y} \le \mu} + \mathbb{1}_{y \le \mu})$. Comparing with the third equation yields $\mathbb{1}_{y \le \mu} = \mathbb{1}_{\tilde{y}'<r} - \mathbb{1}_{\tilde{y}<r} + \mathbb{1}_{\tilde{y} \le \mu} = b_0 + b_1$.

Security. We give the simulator Sim for a corrupted $\mathcal{S}_{\mathsf{idx}}$ with input y_{idx}, for $\mathsf{idx} \in \{0,1\}$.

1. Sim simulates $\mathcal{F}_{\mathsf{FSS-Gen}}$ for $\mathcal{A}$ and obtains the shares $(\vec{v}_{\mathsf{idx}}, \alpha_{\mathsf{idx}})$ of $\mathcal{S}_{\mathsf{idx}}$. It computes r_{idx} from α_{idx} as above and $\tilde{y}_{\mathsf{idx}} = y_{\mathsf{idx}} + r_{\mathsf{idx}}$.
2. Sim samples $\tilde{y}_{1-\mathsf{idx}} \xleftarrow{\$} \mathbb{Z}_{2^{h_2}}$ and sends it to $\mathcal{A}$ as the message of $\mathcal{S}_{1-\mathsf{idx}}$.

3. Sim computes $\tilde{y} = \tilde{y}_0 + \tilde{y}_1$ and the output share b_{idx} as $\mathcal{S}_{\mathsf{idx}}$ does in Step 3, and sends y_{idx} and b_{idx} to $\mathcal{F}_{\mathsf{PropTest}}$.

In the hybrid execution, $r_{1-\mathsf{idx}}$ equals $\alpha - \alpha_{\mathsf{idx}}$ up to the public shift by 1, where α is uniform and hidden from $\mathcal{A}$. So $\tilde{y}_{1-\mathsf{idx}} = y_{1-\mathsf{idx}} + r_{1-\mathsf{idx}}$ is uniform and independent of the rest of the view of $\mathcal{A}$, as in the simulation. Moreover, by correctness, the honest $\mathcal{S}_{1-\mathsf{idx}}$ outputs $\mathbb{1}_{y\le\mu} - b_{\mathsf{idx}}$. This is exactly the share $b - b_{\mathsf{idx}}$ that $\mathcal{F}_{\mathsf{PropTest}}$ gives to $\mathcal{S}_{1-\mathsf{idx}}$ in the ideal execution. Hence the joint distribution of the view of $\mathcal{A}$ and the honest output is identical in both executions. □

D Private Threshold Comparison

Following the Property Testing protocol in Section 5, the servers hold additive shares of indicators for each client. By locally summing these shares, they obtain additive shares s_0, s_1 of a global count $s = s_0 + s_1 \pmod{2^{h_3}}$, where h_3 is sufficiently large to represent the total number of clients.

Distinct from prior approaches that reveal exact frequencies, we introduce this threshold comparison step to strictly limit information leakage. In this phase, the servers securely determine whether the global count s meets the public threshold t (i.e., $s \ge t$) without ever revealing the exact count s. This ensures that only the final heavy-hitter status (a boolean value) is known to the servers. We formalize this interaction in the ideal functionality $\mathcal{F}_{\mathsf{Comp}}$ (Figure 24).

Parameters: input bit length h_3, public threshold $t \in \mathbb{Z}$

Participants: Two servers $\mathcal{S}_0$ and $\mathcal{S}_1$

Functionality:
1. Receive input shares $s_0 \in \mathbb{Z}_{2^{h_3}}$ from S_0 and $s_1 \in \mathbb{Z}_{2^{h_3}}$ from S_1.
2. Reconstruct the global count $s = s_0 + s_1 \pmod{2^{h_3}}$.
3. Set bit $b = 1$ if $s \ge t$, and $b = 0$ otherwise.
4. Send b to both S_0 and S_1.

Figure 24: Ideal functionality $\mathcal{F}_{\mathsf{Comp}}$ for securely comparing a secret-shared value to a public threshold.

Protocol Intuition. Structurally, this protocol mirrors the μ-bounded testing in Appendix C.4. Both face the shared challenge of handling modular overflow when summing secret shares. The primary distinction is the condition: whereas the μ-bounded protocol checks an upper bound ($\le \mu$), this phase checks if the global count meets a lower bound ($s \ge t$).

Given shares s_0, s_1 of the count s, the servers utilize a standard comparison circuit to securely evaluate $s \ge t$. The logic accounts for integer overflow in $s_0 + s_1$ as follows:

Case 1: No Overflow ($s_0 + s_1 < 2^{h_3}$). The reconstructed sum is exactly $s_0 + s_1$. The condition $s \ge t$ is equivalent to checking $s_0 + s_1 \ge t$.

Case 2: Overflow ($s_0 + s_1 \ge 2^{h_3}$). The sum wraps around modulo 2^{h_3}. The true value is $s_0 + s_1 - 2^{h_3}$, so the condition $s \ge t$ is equivalent to checking $s_0 + s_1 \ge t + 2^{h_3}$.

We provide the formal protocols using 2PC and Dealer in Figure 25 and Figure **??**, while the proofs of the security for the protocols are omitted since they strictly follow the structure established in Appendix C.4.

E Nuances in Instantiating Mosaic

Each length parameter h_1, h_2, and h_3 in Mosaic needs to be chosen with care. The minimum requirement is that the length should be at least large enough to represent numbers correctly in each step. For the input length h_1 and the length for output of the Property Testing step h_3, this is exactly the case. The input length h_1, as the name suggests, needs to be large enough to store input points to the system. The

Inputs:

- Party $\mathcal{S}_0$ holds an input share $y_0 \in \{0,1\}^{h_3}$.
- Party $\mathcal{S}_1$ holds an input share $y_1 \in \{0,1\}^{h_3}$.
- Counting threshold $\mu \in \mathbb{Z}_{2^{h_3}}$.

Protocol:

1. Party $\mathcal{S}_0$ and $\mathcal{S}_1$ invoke the $\mathcal{F}_{\mathsf{2PC}}$ functionality on the following circuit:
   (a) $\mathcal{S}_0$ inputs y_0.
   (b) $\mathcal{S}_1$ inputs y_1.
   (c) Two servers jointly compute the following values:
   - $z \leftarrow (t - y_1) \mod 2^{h_2}$
   - $u \leftarrow \mathbb{1}_{y_1 \geq t}$
   - $c \leftarrow \mathbb{1}_{y_0 \leq 2^{h_2}-1-y_1}$ // Checks whether $y_0 + y_1$ overflows
   - $c_0 \leftarrow c \wedge (u \vee \mathbb{1}_{y_0 \geq z})$ // When $y_0 + y_1$ not overflow, check whether $y_1 \geq t$ or $y_0 \geq t - y_1$
   - $c_1 \leftarrow \neg c \wedge (u \wedge \mathbb{1}_{y_0 \geq z})$ When $y_0 + y_1$ overflow, check $y_0 \geq t - y_1$ and $y_1 \geq t$
   - $b \leftarrow c_0 \vee c_1$
2. $\mathcal{S}_0$ samples $b_0 \in \mathbb{Z}_2$, then compute in $\mathcal{F}_{\mathsf{2PC}}$ with $\mathcal{S}_1$ the value $b_1 \leftarrow b \oplus b_0$. $\mathcal{S}_1$ receives b_1.

Figure 25: Protocol $\Pi_{\texttt{Threshold-2PC}}$ for $\mathcal{F}_{\mathsf{Comp}}$.

length h_3, on the other hand, needs to be set for storing the count of matches, with the maximum value for this count being the number of clients N. Hence, h_3 should be chosen to be at least $\log N$.

The recommended length for h_2 is much trickier, as it is also related to the false-positive rate of OKVS-based construction for PB-FSS. We describe the calculations for choosing this parameter in Appendix E.1.

The other nuance is a trade-off between methods for Property Testing. This step costs a lot of computation and communication for servers, as for each point $\vec{x}$ that the servers need to check, they have to run Property Testing for this point against every client point $\vec{q}$. We describe the offline-online communication trade-offs between the two approaches in Appendix E.2.

E.1 Recommended Output Length for PB-iFSS

Each PB-FSS construction (either OKVS-based or DCF-based) requires a careful analysis in choosing the output h_2, in order to maintain correctness. A simple example for a reason to this is to consider when the OKVS-based construction for L_∞ distance metric only has output length $h_2 = 1$ (only binary yes/no). Then, in the no-instance, half of the time, the evaluation results for the two servers would be equal, which means the false-positive rate would be half. Increasing the output length h_2 would be necessary to reduce this false-positive rate into negligible. The appropriate value for h_2 would then be a union bound, depending on two factors: the statistical security parameter λ, as well as the number of times PB-FSS.Eval is called in the protocol.

Recommended output length for OKVS-based construction. The output length h_2 for OKVS-based construction needs to be large enough, to minimize the probability of false-positive for keys outside $[q_i - \delta, q_i + \delta]$ (that evaluate to random values). More specifically, for the L_∞ distance, it could be the case that an integer x is not in the interval but we still have $\mathsf{OKVS}_0.\mathsf{Decode}(x) = \mathsf{OKVS}_1.\mathsf{Decode}(x)$ with probability $\frac{1}{2^{h_2}}$. For the L_p distance, the unlucky case is when there is one dimension i that $x_i \notin [q_i - \delta, q_i + \delta]$, which leads to the sum of decoding results being a random value, and can be less than δ^p with probability $\frac{\delta^p}{2^{h_2}}$. We estimate the total probability that one bad evaluation could happen by taking a union bound over the number of prefixes that could be possibly evaluated, as below.

OKVS-based PB-FSS output length in Known Dictionary. When the dictionary size M is known, the number of PB-FSS.Eval calls is exactly M as well, and no prefix evaluation is required. The probability for bad evaluation for no-instance of OKVS-based PB-FSS construction would then be $\frac{M}{2^{h_2}}$ for L_∞ and $\frac{M\delta^p}{2^{h_2}}$ for L_p. Hence, in this problem setting, the output length h_2 for OKVS-based PB-FSS would be at least $\log M + \lambda$ for L_∞ and $\log M + p\log\delta + \lambda$ for L_p.

OKVS-based PB-iFSS output length in Unknown Dictionary. As the name suggests, it is not clear how to estimate the number of prefixes the two servers need to evaluate in the prefix searching algorithm. In this work we simply bound this by the total number of prefixes possibly touched in the process, which consists of all prefixes that are included in at least one L_∞/L_p ball centered at one of the clients' points. In other words, this upper bound is obtained by taking the maximum number of integer points in a multi-dimensional L_∞/L_p ball, then multiply it with N. While there are tighter bounds for the maximum ball size in the L_p distance [Wid12], we choose the folklore bound $(2\delta+1)^d$ for both metric distances. The probability for bad evaluation for no-instance of OKVS-based PB-iFSS construction would be $\frac{(2\delta+1)^d N}{2^{h_2}}$ for L_∞ and $\frac{(2\delta+1)^d N\delta^p}{2^{h_2}}$ for L_p. Hence, in the unknown dictionary setting, the output length h_2 for OKVS-based PB-iFSS would be at least $d\log\delta + \log N + \lambda$ for L_∞ and $(d+p)\log\delta + \log N + \lambda$ for L_p.

E.2 Trade-offs Between 2PC- and Dealer-Based Property Testing

Output length for Property Testing. The output length h_2 for the Property Testing step only needs to be large enough to store the maximum number of matches, which is upper bounded by the number of client points N. Hence we can set $h_2 = \log N$.

2PC- and Dealer-based Property Testing. We include in Table 4 an estimation for the required communication cost of two methods. In the Dealer-based approach, the majority of communication cost is pushed to the Dealer. The inter-server communication cost is only $O(h_2)$, which is κ-fold smaller than the $O(\kappa h_2)$ communication cost for the Dealer. Asymptotically, the total communication cost of the two approach is similar, but concretely the Dealer-based approach saves more communication. For example, for the Equal property, the total communication for 2PC-based approach is dominated by $4h_2\kappa$, while that quantity for the Dealer-based approach is $2h_2\kappa$. Both approaches requires constant rounds of communication.

Prop	$\mathcal{F}_{\mathsf{PropTest}}$	$\mathcal{D} \to \mathcal{S}_0$ Comm	$\mathcal{S}_0 \to \mathcal{S}_1$ Comm	$\mathcal{S}_1 \to \mathcal{S}_0$ Comm
Equal	2PC	-	$h_2\kappa$	$3h_2\kappa$
	Dealer	$2\kappa h_2 + h_3$	h_2	h_2
$\mathsf{Bounded}_\mu$	2PC	-	$h_2\kappa$	$8h_2\kappa$
	Dealer	$2(2\kappa + 2h_3)h_2$	h_2	h_2

Table 4: Cost breakdown for Property Testing.

F Unified Frameworks for Known and Unknown Dictionary

Proof of Theorem 1. **Clients are corrupt.** The corrupt client does not receive any messages or output during the protocol. It is trivial to see that the clients' view can be simulated independent of the servers' input dictionary D, assuming that the corrupt set of clients do not collude with either server.

Server is corrupt. Without loss of generality, we will consider the case where $\mathcal{S}_0$ is corrupt. Observe that $\mathcal{S}_0$'s view in the real execution protocol consists of a) PB-iFSS key shares $(k_0^{(1)}, \ldots, k_0^{(N)})$, one from each of the N clients, b) output $\left[\!\left[b_i^{(\vec{w})}\right]\!\right]_0$ of ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ for each $\vec{w} \in D$, c) output $o_{\vec{w}}$ from

the ideal functionality $\mathcal{F}_{\mathsf{comp}}$, for each $\vec{w} \in D$. We start with the real execution of the protocol in Hybrid_0 and through a sequence of hybrids move to a simulated view in the ideal world that is independent of the clients input $(\vec{q}_1, \ldots, \vec{q}_N)$.

- Hybrid_0 : We start with the real execution of the protocol, where $\mathcal{S}_0$ interacts with the honest set of clients $\mathcal{C}$ and $\mathcal{S}_1$.
- Hybrid_1 : This hybrid is identical to the real execution, except that we invoke the PB-iFSS simulator $\mathsf{Sim}(1^\kappa, 0, u, d)$ to generate the PB-iFSS key shares received by $\mathcal{S}_0$ in the client upload of the protocol. By the privacy property (see Definition 4), this modification is indistinguishable except with negligible probability from the previous hybrid.
- Hybrid_2 : For each input $\vec{w} \in D$, we replace the output $\left[\!\left[b_i^{(\vec{w})}\right]\!\right]_0$ of private property test ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with uniformly random outputs. This is identically distributed as previous hybrid, by simulatability of the ideal functionality.
- Hybrid_3 : Again, we modify the output received from ideal functionality $\mathcal{F}_{\mathsf{comp}}$. For each input $\vec{w} \in D$, if $\vec{w} \in H$ is in the output of $\mathcal{F}_{\mathsf{known}}$, then forward $o_{\vec{w}} = 1$ as output from $\mathcal{F}_{\mathsf{comp}}$. Otherwise, send $o_{\vec{w}} = 0$.
- *Simulator:* We see that Hybrid_3 defines a valid simulation, independent of client's input $(\vec{q}_1, \ldots, \vec{q}_N)$.

□

Proof of Theorem 2. **Clients are corrupt.** The corrupt client does not receive any messages or output during the protocol. It is trivial to see that the clients' view can be simulated independent of the servers' search prefixes.

Server is corrupt. Without loss of generality, we will consider the case where $\mathcal{S}_0$ is corrupt. Observe that $\mathcal{S}_0$'s view in the real execution protocol consists of a) PB-iFSS key shares $(k_0^{(1)}, \ldots, k_0^{(N)})$, one from each of the N clients, b) output $\left[\!\left[b_i^{(\vec{w})}\right]\!\right]_0$ of ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ for each search prefix $\vec{w}$, c) output $o_{\vec{w}}$ from the ideal functionality $\mathcal{F}_{\mathsf{comp}}$, for each search prefix $\vec{w}$. We start with the real execution of the protocol in Hybrid_0 and through a sequence of hybrids move to a simulated view in the ideal world that is independent of the clients' input $(\vec{q}_1, \ldots, \vec{q}_N)$.

- Hybrid_0 : We begin with the real execution of the protocol, where corrupt $\mathcal{S}_0$ interacts with the honest set of clients $\mathcal{C}$ and honest $\mathcal{S}_1$.
- Hybrid_1 : We modify this hybrid as follows: invoke the PB-iFSS simulator $\mathsf{Sim}(1^\kappa, \mathsf{dist}, 0, u, d)$ to generate the PB-iFSS key shares received by $\mathcal{S}_0$ in the client uploading phase of the protocol. By the privacy property (see Definition 4), this hybrid is indistinguishable from real execution except with negligible probability.
- Hybrid_2 : For each search prefix $\vec{w}$ and for each client $\mathcal{C}_i \in \mathcal{C}$, we replace the output $\left[\!\left[b_i^{(\vec{w})}\right]\!\right]_0$ of property test ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$ (Figure 7) with uniformly random outputs. This is identically distributed as previous hybrid, by simulatability of the ideal functionality $\mathcal{F}_{\mathsf{PropTest}}$.
- Hybrid_3 : Again, we modify the output received from ideal functionality $\mathcal{F}_{\mathsf{comp}}$. For each search prefix $\vec{w}$, if $\vec{w} \in H$ is in the output of $\mathcal{F}_{\mathsf{unknown}}$, then forward $o_{\vec{w}} = 1$ as output from $\mathcal{F}_{\mathsf{comp}}$. Otherwise, send $o_{\vec{w}} = 0$. Additionally, we handle leakage, by forwarding every prefix $p_{\vec{w}} \in \mathcal{L}$ from the ideal functionality $\mathcal{F}_{\mathsf{unknown}}$ to $\mathcal{S}_0$.
- *Simulator:* Observe that, Hybrid_3 defines a valid simulation, independent of client's input $(\vec{q}_1, \ldots, \vec{q}_N)$.

□