

A Cryptographic System for Private Fuzzy Heavy Hitters

by
Siddarth K. Sitaraman

A Thesis submitted in partial fulfillment of the requirements
for Honors in the Department of Computer Science at Brown
University

Providence, Rhode Island
April 2026

Abstract

We present Mosaic, a cryptographic system 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 individual clients' 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 the L_∞ metric. Mosaic operates in a 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 system 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 clients' 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 given property, rather than forming exact additive secret shares of the function output. We provide a PB-FSS construction for the L_∞ distance metric from lightweight cryptographic techniques. Furthermore, we introduce new methods to prevent malicious client behavior by detecting malformed PB-FSS shares. The servers are assumed to be semi-honest during the fuzzy matching phase. Finally, we implement Mosaic in Rust and evaluate it using real-world trace data from a ride-sharing service. We show that Mosaic is practical and efficient in that each client can complete their upload within milliseconds, and the servers can process data from over a hundred thousand clients to discover fuzzy heavy hitters in tens of seconds.

Acknowledgements

First and foremost, I would like to thank my honors thesis advisor, Prof. Peihan Miao, for her guidance, encouragement, and thoughtful mentorship. Working with her on this thesis and learning from her cryptography courses has been foundational to my development as a researcher and my decision to enter a doctoral program in the field. I am also grateful to Prof. Anna Lysyanskaya for serving as the reader for this thesis and for the experience of being a teaching assistant for her cryptography course.

This thesis is part of a larger project to build a complete modular framework for private fuzzy heavy hitter analysis that was done in collaboration with Phuoc Pham Van Long, Eileen Nolan, Dr. Gayathri Garimella, and Prof. Miao and will be presented in the ACM Conference on Computer and Communications Security (CCS) 2026 [21]. I am grateful for the opportunity to have collaborated with this team and to have learned from them.

Contents

1	Introduction	1
2	Mosaic: Intuition and Overview	5
2.1	Putting it together	8
3	Mosaic: Detailed Description	10
3.1	Notation and Parameters	10
3.2	Property-Based Function Secret Sharing	11
3.2.1	Property-Testing Function for L_∞	13
3.2.2	DCF-Based Construction of PB-iFSS	14
3.3	Private Property Testing	14
3.4	Private Threshold Comparison	15
3.5	2PC-Based Approach for L_∞ Distance	16
3.6	Sketching for DCF-based PB-FSS	19
3.7	Putting It Together	19
4	Empirical Evaluation	20
5	Related Work	25
6	Conclusion	27
	Bibliography	28
A	Cryptographic Building Blocks	33
A.1	Secure Multi-party Computation	33
A.2	Function Secret Sharing	34

A.3	DCF-Based Construction Details of PB-iFSS	35
A.3.1	Construction Overview	35
A.3.2	DCF-based PB-iFSS for L_∞	36
B	Sketching Details	39
B.1	Sketching for L_∞	40
B.2	Optimization: Sketching for payloads in $\mathbb{Z}_{2^k}$	43
C	Security definitions and proofs	46

Chapter 1

Introduction

Enterprises increasingly rely on user data to derive actionable insights, but such data often contains sensitive information and must be analyzed 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, the GPS coordinates of users are private and subject to measurement noise, and two users whose cars are parked in the same lot may report slightly different locations. Finding *fuzzy heavy hitters*, which relies on approximate matching of the locations, is useful in this case. As another example, consider an analyst who 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 1.1). Using only the approximate GPS location of the riders may be necessary to protect their privacy. As a third example, consider a social media analytics provider that wants to find the most trending web searches within a time frame. Since two semantically-similar queries may not match exactly, it would be relevant to match a semantic embedding of these queries in a fuzzy fashion.

Problem Statement. These motivating examples can be formalized as the *Private Fuzzy Heavy Hitters Detection* problem. Consider a large number of clients (i.e.,

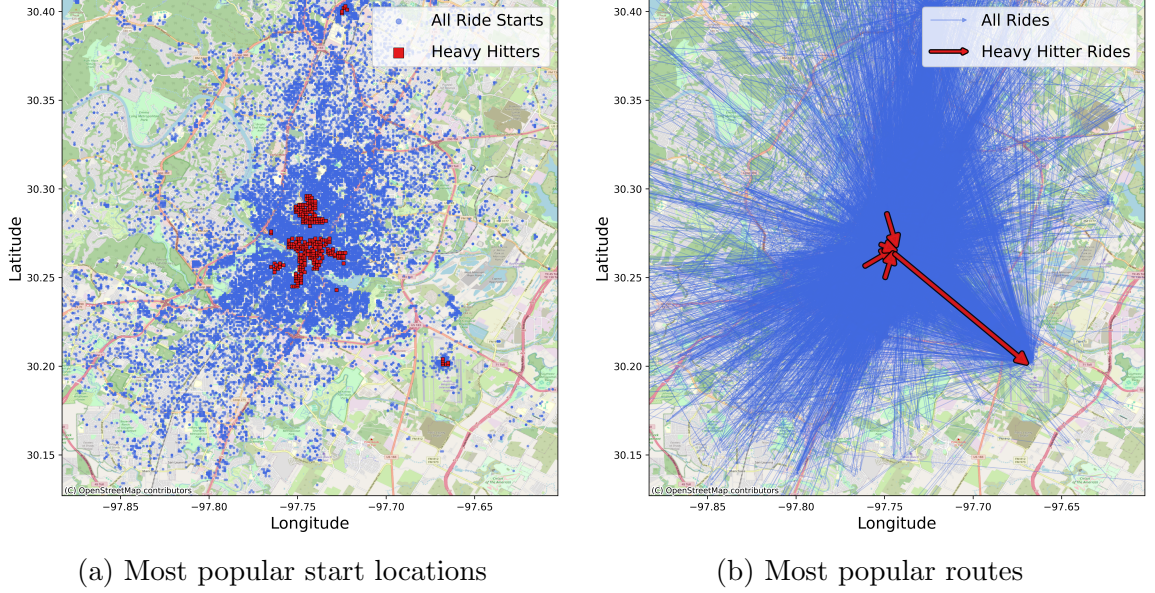


Figure 1.1: Mosaic computes the most popular start locations and rides (i.e., fuzzy heavy hitters shown in red) for a ride-sharing service in Austin, TX, without learning individual client data (shown in blue).

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 $\text{dist}(\cdot, \cdot)$ (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,

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

Mosaic System. In this thesis, we formalize the problem of private fuzzy heavy hitters detection and present Mosaic, a system that solves the problem in the L_∞ distance metric using lightweight cryptographic techniques. Mosaic works in a setting where a large set of clients communicates with two non-colluding servers. The system 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. We assume all parties

are semi-honest as a baseline, and present novel techniques to protect against malicious clients. We show a visualization of the output of Mosaic for the ride-sharing application in Figure 1.1.

Known and Unknown Dictionary Problem Variants. Mosaic supports two variants of detecting heavy hitters. 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 applies 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.

Limitations of Prior Work. Prior work has developed protocols that allow service providers to identify heavy hitters in datasets [7, 32, 27]. 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 $\text{dist}(\vec{x}, \vec{q}) \leq \delta$) 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

80 $\times$ in terms of key size and 35 $\times$ in terms of server computation.

Our Contributions. 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 the L_∞ metric.
- We present Mosaic, a cryptographic system 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. Our system is semi-honest secure as a baseline.
- 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. 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. Further, in the fuzzy matching phase, we design new protocols for private property testing and private threshold comparison. Finally, we develop new sketching techniques to detect malformed PB-iFSS 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. In one setting, the client-side computation for the upload phase takes microseconds, and server-side phases can process over hundred thousand clients within tens of seconds.

Chapter 2

Mosaic: Intuition and Overview

We begin with the *exact* heavy hitters problem for the known-dictionary scenario. 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, namely 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 [7] using distributed point functions (DPFs) [22]. A DPF is a special function secret sharing (FSS) scheme [8] 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 the following 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 $\vec{x}$,

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

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 of DPF ensures that, when

the two servers evaluate their keys on a candidate $\vec{x} \in \mathcal{D}$, they obtain additive secret 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 $\text{Eval}(k_b, \vec{x})$ 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 $\mathbb{1}_{\text{dist}(\vec{x}, \vec{q}) \leq \delta}$. 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:

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

If we can construct an efficient FSS for the desired distance metric, then the rest of the protocol follows exactly the same way. In this work, we focus on the L_∞ distance metric. However, existing FSS constructions for the L_∞ distance metric require both the key size and evaluation time to grow exponentially in the dimension [9].

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 $\text{Eval}(k_b, \vec{x})$ 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 $\text{Eval}(k_0, \vec{x})$ and $\text{Eval}(k_1, \vec{x})$ output values that satisfy a specific *property* iff $\text{dist}(\vec{x}, \vec{q}) \leq \delta$. In particular, the property we define for L_∞ distance metric is *equality*, i.e. we test whether $\text{Eval}(k_0, \vec{x}) = \text{Eval}(k_1, \vec{x})$. After the FSS evaluation $\text{Eval}(k_b, \vec{x})$ by each server $\mathcal{S}_b$, the two servers perform a **private property testing** protocol to obtain secret shares of the indicator $\mathbb{1}_{\text{dist}(\vec{x}, \vec{q}) \leq \delta}$, 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 $\vec{p}$, $\text{Eval}(k_0, \vec{p})$ and $\text{Eval}(k_1, \vec{p})$ 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 construction for PB-FSS. We present a new construction of property-based (incremental) FSS for the L_∞ distance metric, with an equality property. Our construction relies only on lightweight, symmetric-key cryptographic operations such as AES and hash functions. This construction is based on (incremental) distributed comparison functions (DCF) [8, 9]. We present the definition and constructions of PB-FSS in Section 3.2.

Private property testing. Recall that after obtaining the PB-FSS evaluations $y_b = \text{Eval}(k_b, \vec{x})$ by each server $\mathcal{S}_b$, the two servers need to perform a private property testing protocol to get secret shares of the indicator $\mathbb{1}_{\text{dist}(\vec{x}, \vec{q}) \leq \delta}$. The property we define for the L_∞ distance metric is *equality*. Specifically, the servers need to test whether $y_0 = y_1$. We use generic secure two-party computation (2PC) to conduct property testing for L_∞ . We describe the protocols in Section 3.3.

Private threshold comparison. 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 it 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 [7], 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 use generic 2PC as described in Section 3.4.

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, thereby preventing malicious client behavior. Prior work [9, 7] provides sketching techniques for detecting malformed shares in DPF and DCF constructions. We build on these techniques and develop new sketching methods for our DCF-based PB-FSS constructions for the L_∞ distance metric. We present the new sketching techniques in Section 3.6.

2.1 Putting it together

In summary, putting together the cryptographic building blocks outlined earlier in this chapter, the Mosaic system proceeds as follows:

- **Client’s 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 the non-colluding servers $\mathcal{S}_0$ and $\mathcal{S}_1$, respectively, after which the client may go offline.
- **Server’s Fuzzy Matching Phase.** Upon receiving all clients’ key shares, the two servers may (optionally) perform a **sketching** process to eliminate malformed (or, malicious) input from the clients. Further, they collaboratively perform the following two steps to detect fuzzy heavy hitters. In the known dictionary setting, the servers evaluate on each client’s PB-FSS key share with potential candidates, whereas in the unknown dictionary settings, the servers discover the candidates via PB-iFSS dynamically using a prefix-tree based search algorithm.

- **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 $\mathbb{1}_{\text{dist}(\vec{x}, \vec{q}) \leq \delta}$ 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.
- **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.

Chapter 3

Mosaic: Detailed Description

We now present a detailed description of each cryptographic building block outlined in Chapter 2 and finally present the pseudo-code description of Mosaic that puts it all together.

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 denote the empty string by ε . We use $[n]$ to denote the range $\{1, 2, \dots, n\}$. We use $\llbracket x \rrbracket$ to denote *additive* shares for x .

We write $x \prec y$ (and equivalently $y \succ x$) if string x is a prefix of 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 $x_1 \prec y_1, \dots, x_d \prec y_d$. We allow comparison between two bit-strings that are not equal in length, and say that $x < y$ if there exists a 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). 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.

Table 3.1: 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 input (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

Distance metric. For two vectors $\vec{x} = (x_1, \dots, x_d) \in (\{0, 1\}^u)^d$ and $\vec{q} = (q_1, \dots, q_d) \in (\{0, 1\}^u)^d$, let $\text{dist}(\vec{x}, \vec{q}) = L_\infty(\vec{x}, \vec{q}) := \max_{i=1}^d |x_i - q_i|$. We define $\text{BALL}_{\text{dist}, \vec{q}, \delta} := \{\vec{x} \in (\{0, 1\}^u)^d \mid \text{dist}(\vec{x}, \vec{q}) \leq \delta\}$ as the d -dimensional **dist**-ball centered at $\vec{q}$ with radius δ .

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

3.2 Property-Based Function Secret Sharing

Given a function family $\mathcal{F}$, a Function Secret Sharing (FSS) scheme [8, 9, 26, 10] 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 introduce a new notion called *Property-Based (Incremental) Function Secret Sharing*, which relaxes the correctness requirement of FSS by incorporating a

property-testing function. A property-based function secret sharing PB-FSS scheme consists of two algorithms (**Share**, **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**. 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_∞ distance metric. Algorithm **Share** takes a set $S \in \mathcal{S}$ as input and outputs 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 $\text{idx} \in \{0, 1\}$. The correctness guarantee is that the property testing **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 $\vec{p}$. In this setting, the property testing outputs 1 if and only if there exists a completion of the prefix $\vec{q}$ that lies in S .

Definition 1 (Property-Based (Incremental) Function Secret Sharing). *A two-party property-based (incremental) function secret sharing scheme PB-FSS (or PB-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 (**Share**, **Eval**) with the following syntax:*

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

The PB-FSS (or PB-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:*

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

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 κ :

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

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

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

In the following section, we present PB-iFSS constructions that supports the L_∞ distance metric. We first define the **equality** property-testing function for L_∞ in Section 3.2.1. Next, we describe the DCF-based construction in Section 3.2.2. 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. Our construction paradigms start with PB-FSS and then extend to PB-iFSS with incremental evaluation.

3.2.1 Property-Testing Function for L_∞

A client with point $\vec{q}$ generates 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 evaluation results as inputs to the next steps that determine whether $\text{dist}(\vec{x}, \vec{q}) \leq \delta$. For the L_∞ distance metric, given a center $\vec{q}$ and a point $\vec{x}$, we test the *equality* property, namely, whether $\text{Eval}(k_0, \vec{x}) = \text{Eval}(k_1, \vec{x})$, 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 if a value x lies in the interval $[q - \delta, q + \delta]$ then that means the outputs from **Eval** are equal. 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 $|\vec{x} - \vec{q}| \leq \delta$ 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 .

Definition 2 (Property-Testing Functions). *We consider the following property-testing function. These properties are defined for pairs of values in $\mathbb{Z}_{2^k}$.*

$$\text{Equal}(y_0, y_1) = \begin{cases} 1 & \text{if } y_0 = y_1 \\ 0 & \text{otherwise} \end{cases}.$$

3.2.2 DCF-Based Construction of PB-iFSS

Our construction of PB-FSS leverages (incremental) DCF, which is a special FSS for comparison functions:

$$f_{\alpha, a, b}(x) = \begin{cases} a & \text{if } x \leq \alpha \\ b & \text{otherwise} \end{cases}.$$

To construct PB-FSS for the L_∞ distance metric with regard to a point $\vec{q}$, it suffices to secret share the following function for each dimension $i \in [d]$ and concatenate the results:

$$f_i(x) = \begin{cases} 0 & \text{if } x \in [q_i - \delta, q_i + \delta] \\ \neq 0 & \text{otherwise} \end{cases}.$$

Notice that the interval $[q_i - \delta, q_i + \delta]$ can be written as the difference of two one-sided intervals: $[0, q_i + \delta] - [0, q_i - \delta]$, hence $f_i(x)$ can be computed as the difference of two DCFs: $f_{q_i + \delta, 0, 1}(x) - f_{q_i - \delta - 1, 1, 0}(x)$. Furthermore, incremental DCFs give rise to PB-iFSS. The details of the construction of PB-iFSS is given in Appendix A.3.

3.3 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 3.2. The servers first perform Private Property Testing where they securely check whether their local evaluations, y_0 and y_1 , of the client's PB-FSS shares satisfy

the required property (i.e., the **Equal** property). 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 $\text{dist}(\vec{x}, \vec{q}) \leq \delta$ for some candidate item $\vec{x}$. At the start of this step, servers S_0 and S_1 have evaluated their PB-FSS keys on a candidate item $\vec{x}$. They possess local output shares y_0 and y_1 , respectively. The property testing protocol securely computes secret shares of a bit b , which indicates whether the property **Equal** (described in Section 3.2.1) holds for y_0 and y_1 . The ideal functionality $\mathcal{F}_{\text{PropTest}}$ of this step is formalized in Figure 3.1.

Parameters:

- Input share length h_2 .
- Output share length h_3 .
- Property type: **Equal**.

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 as follows. If $y_0 = y_1$, set $b = 1$, else set $b = 0$.
3. If S_{idx} is corrupt, receive output share b_{idx} from the adversary and compute $b_{1-\text{idx}} = b - b_{\text{idx}} \pmod{2^{h_3}}$.
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 3.1: Ideal functionality $\mathcal{F}_{\text{PropTest}}$.

3.4 Private Threshold Comparison

The second step in the fuzzy matching phases is the Private Threshold Comparison, where 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 . Following the Private Property Testing protocol in Section 3.3, 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 step, the servers securely determine whether the global count s meets the public threshold t (i.e., $s \geq t$) 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 $\mathcal{F}_{\text{Comp}}$ (Figure 3.2).

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 $\mathcal{S}_0$ and $s_1 \in \mathbb{Z}_{2^{h_3}}$ from $\mathcal{S}_1$.
2. Reconstruct the global count $s = s_0 + s_1 \pmod{2^{h_3}}$.
3. Set bit $b = 1$ if $s \geq t$, and $b = 0$ otherwise.
4. Send b to both $\mathcal{S}_0$ and $\mathcal{S}_1$.

Figure 3.2: Ideal functionality $\mathcal{F}_{\text{Comp}}$ for comparing a secret-shared value to a public threshold.

A main challenge is handling modular overflow when summing secret shares. Given shares s_0, s_1 of the count s , the servers utilize a standard comparison circuit to securely evaluate $s \geq t$. 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 $s \geq t$ is equivalent to checking $s_0 + s_1 \geq t$.

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

3.5 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 [34], which allows the servers to evaluate the function within a constant number of rounds. Specifically, $\mathcal{S}_0$ and $\mathcal{S}_1$ 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 detailed protocol $\Pi_{\text{Equal-2PC}}$ is presented in Figure 3.3, and the security proof is as follows.

Theorem 3 (Security of $\Pi_{\text{Equal-2PC}}$). *Protocol $\Pi_{\text{Equal-2PC}}$ (Figure 3.3) securely realizes the ideal functionality $\mathcal{F}_{\text{PropTest}}$ (Figure 3.1) in the $\mathcal{F}_{2\text{PC}}$ -hybrid model against semi-honest adversaries.*

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

Security. The security of $\Pi_{\text{Equal-2PC}}$ (Figure 3.3) securely realizes the ideal functionality $\mathcal{F}_{\text{PropTest}}$ (Figure 3.1) in the $\mathcal{F}_{2\text{PC}}$ -hybrid model, assuming the underlying 2PC protocol securely evaluates circuits in the presence of semi-honest adversaries. $\square$

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 $\mathcal{F}_{2\text{PC}}$ for circuit C .
2. S_0 provides input y_0 and S_1 provides input y_1 .
3. 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 3.3: Protocol $\Pi_{\text{Equal-2PC}}$ for $\mathcal{F}_{\text{PropTest}}$.

Inputs:

- Servers $\mathcal{S}_0$ and $\mathcal{S}_1$. If $\text{case} = \text{known}$, each server also holds a dictionary $W \subseteq (\{0, 1\}^{h_1})^d$ of input points.
 - Set of N clients $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_N)$, 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 $\mathbb{G}$ being $\{0, 1\}^{h_2 d}$. Generate PB-FSS (or PB-iFSS) key pairs:

If $\text{case} = \text{known}$: $(k_0^{(i)}, k_1^{(i)}) \leftarrow \text{Share}(1^\kappa, \text{dist}, \vec{q}_i, \delta, D, \mathbb{G})$.

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

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

- Client sends $k_0^{(i)}$ to $\mathcal{S}_0$ and $k_1^{(i)}$ to $\mathcal{S}_1$. From now on, we abuse notation and use Eval for both PB-FSS and PB-iFSS.

Fuzzy Matching Subroutine:

- Both servers have as input being a prefix $\vec{x} \in (\{0, 1\}^{\leq h_1})^d$, and each server $\mathcal{S}_b$ holds a set of keys $(k_b^{(1)}, k_b^{(2)}, \dots, k_b^{(N)})$.
- **Property Testing:** For each client index $i \in [N]$:
 1. $\mathcal{S}_0$ computes $y_{i,0} \leftarrow \text{Eval}(0, \text{dist}, k_0^{(i)}, \vec{x})$ and $\mathcal{S}_1$ computes $y_{i,1} \leftarrow \text{Eval}(1, \text{dist}, k_1^{(i)}, \vec{x})$.
 2. Both servers invoke $\mathcal{F}_{\text{PropTest}}$ with inputs $y_{i,0}$ from $\mathcal{S}_0$ and $y_{i,1}$ from $\mathcal{S}_1$. As output, $\mathcal{S}_0$ gets $\llbracket b_i \rrbracket_0$ and $\mathcal{S}_1$ gets $\llbracket b_i \rrbracket_1$.
- **Threshold Comparison:**
 1. Both servers obtain secret shares of aggregation: $\mathcal{S}_0$ gets $\llbracket s \rrbracket_0 \leftarrow \sum_{i=1}^N \llbracket b_i \rrbracket_0$ and $\mathcal{S}_1$ gets $\llbracket s \rrbracket_1 \leftarrow \sum_{i=1}^N \llbracket b_i \rrbracket_1$.
 2. Both servers invoke $\mathcal{F}_{\text{comp}}$ with inputs $\llbracket cs \rrbracket_0$ from $\mathcal{S}_0$ and $\llbracket s \rrbracket_1$ from $\mathcal{S}_1$, and threshold t .
- Both servers obtain output $o_{\vec{x}}$ from $\mathcal{F}_{\text{comp}}$, which would also be 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 $\vec{w} \in W$, 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 $H \leftarrow \{\varepsilon\}$.
- Iterate through each prefix length $u = 1, \dots, h_1$: Iterate through each dimension $i = 1, \dots, d$:
 1. Initialize an empty set H_{new} .
 2. For each prefix $\vec{x} = (x_1, \dots, x_d)$ in the current set H :
 - Run the Fuzzy Matching Subroutine for $\vec{x}_0 = (x_1, \dots, x_{i-1}, x_i \| 0, x_{i+1}, \dots, x_d)$. 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 $\vec{x}_1 = (x_1, \dots, x_{i-1}, x_i \| 1, x_{i+1}, \dots, x_d)$. 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. Reassigning $H \leftarrow H_{\text{new}}$.
- Both servers output H .

Figure 3.4: The Mosaic system for Private Fuzzy Heavy Hitters.

3.6 Sketching for DCF-based PB-FSS

The Mosaic framework presented in Figure 3.4 assumes that all clients are semi-honest and generate well-formed PB-FSS shares. To further strengthen the robustness of Mosaic, we introduce techniques for detecting malformed PB-FSS shares, thereby preventing malicious client behavior. Our sketching techniques work for the DCF-based PB-FSS, and we highlight the key technical challenges below.

First, as discussed in Section 3.2.2, we require the client to generate multiple DCFs with correlated splitting points (e.g., $q+\delta$ and $q-\delta$ for a hidden q and public δ), as well as correlated payloads (e.g., q and q^2 for a hidden q). Existing sketching techniques only address a standalone DPF or DCF. Second, in the unknown-dictionary setting, our protocol requires incremental evaluations, and hence incremental DCF with correlated splitting points and consistent payloads across different levels. In contrast, prior work [7] only considers sketching for incremental DPF and does not provide payload consistency across levels. Finally, for technical reasons, existing sketching techniques only work for payloads over large fields, such as $\mathbb{Z}_p$ for a large prime p . However, to achieve optimal efficiency in our private property testing, we require payloads over the ring $\mathbb{Z}_{2^k}$. We defer the detailed constructions to Appendix B.

3.7 Putting It Together

We provide the pseudo-code for Mosaic in Figure 3.4. In Appendix C, we define the ideal functionalities for the fuzzy heavy hitters problem in the known and unknown dictionary settings in Figure C.1 and Figure C.2, respectively, and prove the following.

Theorem 4. *For a distance metric dist with corresponding set family and property $(\mathcal{S}, \text{Prop})$, given PB-FSS scheme for $\mathcal{S}$ and Prop , the protocol in Figure 3.4 realizes $\mathcal{F}_{\text{known}}$ (Figure C.1) in the $\mathcal{F}_{\text{PropTest}}$ and $\mathcal{F}_{\text{Comp}}$ hybrid model, in the presence of semi-honest adversaries.*

Theorem 5. *For a distance metric dist with corresponding set family and property $(\mathcal{S}, \text{Prop})$, given PB-FSS scheme for $\mathcal{S}$ and Prop , the protocol in Figure 3.4 realizes $\mathcal{F}_{\text{unknown}}$ (Figure C.2) in the $\mathcal{F}_{\text{PropTest}}$ and $\mathcal{F}_{\text{Comp}}$ hybrid model, in the presence of semi-honest adversaries.*

Chapter 4

Empirical Evaluation

To evaluate our algorithms, we chose the real-world application of finding heavy hitters in data from a ride-sharing service called RideAustin [33]. RideAustin was a non-profit alternative to Uber and Lyft that provided rides for users in Austin, TX, before closing down in 2020. The ride data that we use was reported by the RideAustin app installed on mobile devices used by individual users to hail rides. The reported data for each ride includes the start time, end time, start location, end location, distance, driver rating, and rider rating. We consider a third-party analytics provider that collects the ride data from the users of the RideAustin app in real time to perform heavy hitters analysis, for example, the most popular ride start locations, end locations, and routes.

Preserving privacy of user locations. The user’s locations are reported at the accuracy of one-thousandth degree in both the latitude and longitude, which translates to an accuracy of 111 meters and 88 meters, respectively. Since the location leaks private information about the user, the identity of the user must be held *private* from the analytics provider. Further, precise locations can be deanonymized easily, e.g., if a ride starts in the vicinity of a home, the rider is likely its occupant, who can be deanonymized from the town records or the phone book. For this reason, we study users reporting their locations in a *fuzzy* manner, where their latitudes and longitudes are reported within an L_∞ ball of radius δ . The choice of δ offers a tradeoff between privacy and the usefulness of the analysis. A smaller δ leaks more private information, and location statistics that are too fine-grained are less insightful. However, a δ

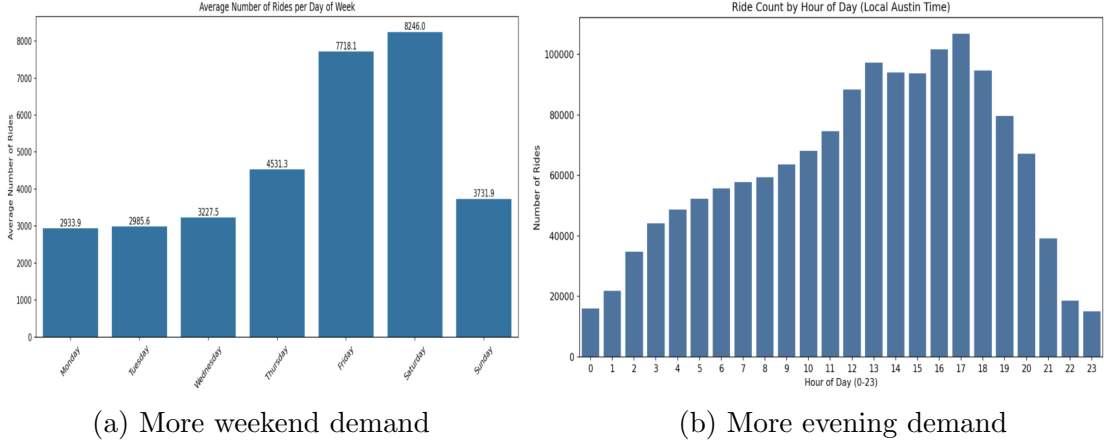


Figure 4.1: Ride Austin Statistics

that is too large can encompass multiple city neighborhoods, also making the traffic hotspot analysis less useful. We examine neighborhoods of radius $1 \leq \delta \leq 3$ as this provides accuracy at a granularity of a few city blocks, where the units of δ are $\frac{1}{1000}^{th}$ of a degree.

Data Characteristics. We use data [33] of nearly 1.5 million rides taken by clients of RideAustin from June 4th 2016, to April 13th, 2017. Figure 4.1 provides the ride statistics that show greater demand in evenings and weekends.

Implementation and Experimental Setup. We implemented Mosaic in approximately 16,000 lines of Rust. We simulate our clients (i.e., users) on an Apple MacBook laptop with 8 cores. The clients report their data to two servers hosted on Amazon AWS locations in the `us-east-1` region (N. Virginia) and the `us-east-1` region (Ohio). We use `c4.8xlarge` servers, which are configured with 36 virtual CPUs. We ran two heavy-hitter query types to experiment with varying dimensions: the “top starts” query has 2-dimensional inputs consisting of the latitude and longitude of the start location of the client, whereas the “top routes” query has four-dimensional inputs with both the start and end location of each ride having a latitude and longitude each. Further, to experiment with varying numbers of clients, we run our experiments on the busiest day (21k rides), week (59k rides), and month (115k rides). The threshold for heavy hitters is fixed at 2% in our experiments, i.e., a heavy hitter has 2% of the total clients in its L_∞ neighborhood.

Client-side Analysis. In the upload phase of Mosaic, each client generates an PB-iFSS share; for L_∞ , this consists of $2d$ DCF keys for string length u . In Figure 4.2,

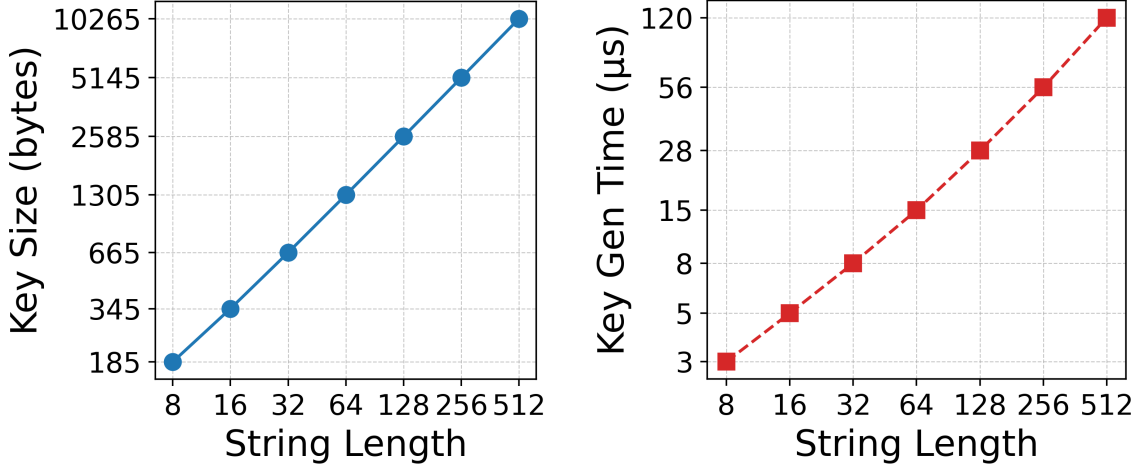


Figure 4.2: Client-side Analysis: The key size and keygen time increase linearly with string length.

we show the key size and key generation time for one such DCF. The clients only require a fraction of a millisecond to generate a pair of DCF keys, when each key is a few KB in size. Both the key generation time and key size grow linearly with string length u since the number of correction words generated is proportional to the string length.

Server time and impact of network latency. Figure 4.3a shows the total server time for processing the busiest day (21K rides), week (59K rides), and month (115K rides) for a radius $\delta = 3$. While the servers are managed by non-colluding entities, we show two possible server deployment scenarios: our standard configuration, where the servers are deployed in different locations (`us-east1` and `us-east2`), and another where the servers are deployed in the same location (both in `us-east1`). In all cases, the server times are in the order of tens of seconds, making it feasible to scale our approach to real-time analysis of much larger data sets. Comparing the server times for the two configurations reveals the significant impact of greater network latency incurred for server-to-server communication in the multi-location setting, resulting in 60–70% more time than the single location.

Component breakdown of server time. During the Fuzzy Matching Phase, the servers jointly do Private Property Testing (Section 3.3) and Threshold Comparison (Section 3.4). We measure the server time for processing the client data and break it down into five subroutines of the Private Property Testing and Threshold Comparison

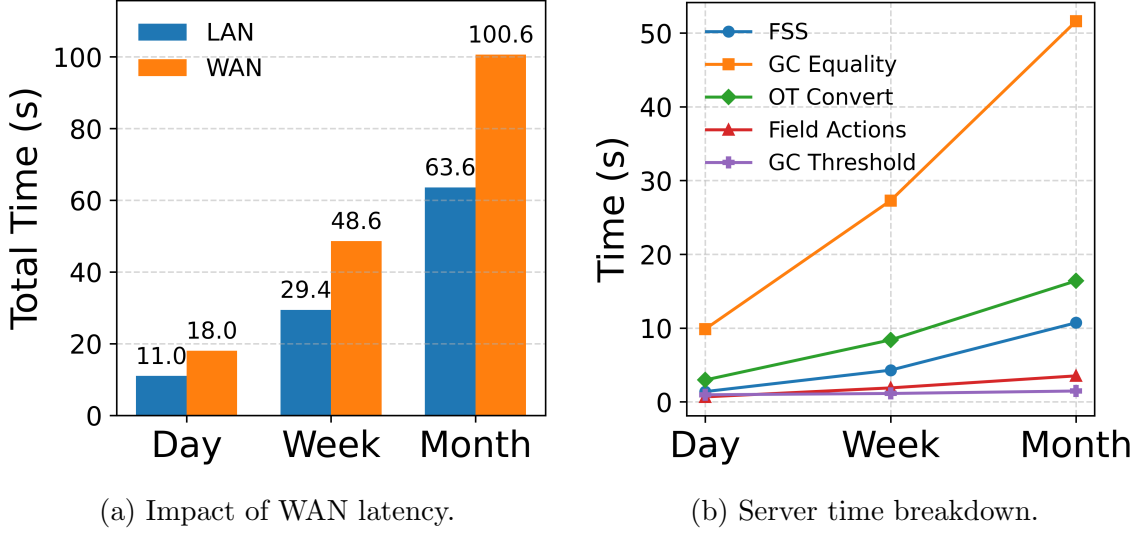


Figure 4.3: Server-side Analysis

processes as follows.

- Private Property Testing consists of FSS Eval, GC Equality, and OT subroutines. The first subroutine, (FSS Eval), is for evaluating the PB-iFSS for each client, resulting in $2dm$ PB-iFSS evaluations for each node of the tree that is explored, where d is the number of dimensions and m is the number of clients. The second subroutine of the Private Property Testing phase is the garbled circuit for equality testing (GC Equality), involving m equality tests of d -bit strings per explored tree node. The third part is time for m oblivious transfers (OT) per tree node.
- Threshold Comparison consists of Field Sum and GC Threshold subroutines. Field Sum, performs the summation of m field secret shares for the explored tree node. GC Threshold is a garbled circuit for threshold comparison, one executed for each explored tree node.

Figure 4.3b shows the total server time and its breakdown for the busiest day (21K rides), week (59K rides), and month (115K rides) for a radius $\delta = 3$. The time spent in each phase by the servers was averaged across the two servers. The server time is dominated by the time for equality testing using garbled circuits. Even though the number of clients increased, the number of tree nodes searched for the busiest day,



Figure 4.4: For a fixed data set of day, week, or month, increasing ball size increases the number of heavy hitters and in proportion the server time.

week, and month happened to be similar, with only small variations. Specifically, the number of explored nodes was 984, 1044, and 1012 respectively. Since time for **GC Threshold** is proportional to only the number of explored nodes and independent of the number of clients, it does not show a significant increase, while the other time components are proportional to the number of clients and show a proportional increase.

Impact of Ball Radius (ρ). Figure 4.4 shows that for a fixed data set (day, week, or month), increasing the ball radius increases the number of heavy hitters, and hence the server time. Note that increasing the neighborhood size increases the number of points in each neighborhood, creating more heavy hitters, resulting in more server time to process them.

Chapter 5

Related 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 [31], [35], [3], [18], [2], 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 [5], [6], [30], allow a server to securely sum client inputs and sometimes use data structures like count-min sketches [13], [4] to identify heavy hitters. However, these systems often require multiple rounds of interaction to handle client dropouts or leak 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 [12], 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 [1], [15] employ Verifiable Distributed Aggregation Functions or generic two-party computation for RAM programs [19], [25], [29]. Although generic MPC combined with Oblivious RAM (ORAM) [24] offers theoretically optimal asymptotic complexity, it remains concretely expensive due to the high overhead of ORAM implementation [17].

Function Secret Sharing (FSS). Our work builds upon FSS introduced by Boyle

et al. [8, 9]. FSS allows servers to aggregate function evaluations on secret inputs without learning the input themselves. Recent systems like Prio[14] and Poplar [16] 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_∞ distance metric, enabling scalable private detection of approximate heavy hitters.

Chapter 6

Conclusion

We introduce the private fuzzy heavy hitters problem, which focuses on identifying popular items in datasets while allowing for approximate matches, and propose the Mosaic system to solve it. In particular, we present a novel scheme that enables efficient analytics for the L_∞ distance metric. We implement this system in Rust, demonstrating its practicality. Additionally, our work introduces new techniques for property-based function secret sharing, which may be of independent interest.

In our broader work [21], we extend Mosaic to be a general framework for fuzzy heavy hitter analysis that can also accommodate the L_p distance metric by providing an alternate instantiation of PB-iFSS that uses a different property testing step. We also add a dealer-based approach for server fuzzy matching. However, other distance metrics that might be appropriate for fuzzy matching in some application domains, such as the hamming distance, the edit distance, or cosine similarity, remain open.

Bibliography

- [1] 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.
- [2] Apple Privacy Team. Learning with privacy at scale. *Differential Privacy Team, Apple, Technical Report*, 1(8), 2017.
- [3] 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.
- [4] 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.
- [5] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrede 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.
- [6] 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.

- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
- [13] 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.
- [14] 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.
- [15] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. *Proceedings on Privacy Enhancing Technologies*, 2023(4):578–592, 2023.

- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] Gayathri Garimella, Peihan Miao, Eileen Nolan, Phuoc Van Long Pham, and Siddarth Sitaraman. Mosaic: A modular framework for private fuzzy heavy hitters. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2026.
- [22] 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.
- [23] 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.
- [24] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

- [25] 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.
- [26] 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 23 International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 541–554. Springer, Cham, June 2023.
- [27] Banashri Karmakar, Shyam Murthy, Arpita Patra, and Protik Paul. Quick-Pool: Privacy-preserving ride-sharing service. Cryptology ePrint Archive, Paper 2024/1109, 2024.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.

- [33] RideAustin. Ride Austin, Data World. <https://data.world/login?next=Fride-austin>, 2017.
- [34] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.
- [35] 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.

Appendix A

Cryptographic Building Blocks

A.1 Secure Multi-party Computation

Secure Multi-party Computation [34, 23] allows multiple servers to jointly evaluate a circuit, while keeping their inputs secret. We consider the secure computation functionality between two parties $\mathcal{F}_{2\text{PC}}$. 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 the readers to [11, 28] for more details of the security paradigm.

Definition 6 (Security of 2PC in the semi-honest model). *Let $f = (f_1, f_2)$ be a two-party functionality. We say that a protocol π securely computes f in the presence of static semi-honest adversaries if there exist PPT simulators S_1 and S_2 such that for all x, y, n :*

$$\begin{aligned} & \{S_1(1^n, x, f_1(x, y)), f(x, y)\}_{x, y, n} \\ & \stackrel{c}{\approx} \{\text{view}_1^\pi(x, y, n), \text{output}^\pi(x, y, n)\}_{x, y, n}. \end{aligned}$$

$$\begin{aligned} & \{S_2(1^n, y, f_2(x, y)), f(x, y)\}_{x, y, n} \\ & \stackrel{c}{\approx} (\text{view}_2^\pi(x, y, n), \text{output}^\pi(x, y, n))_{x, y, n}. \end{aligned}$$

Here, $x, y \in \{0, 1\}^*$ with $|x| = |y|$, $n \in \mathbb{N}$, and “ $\stackrel{c}{\approx}$ ” denotes computational indistinguishability.

A.2 Function Secret Sharing

Definition 7 ((Incremental) Function Secret Sharing [8, 9, 7]). *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:*

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

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

$$\begin{aligned} \Pr \Big[& y_0 + y_1 = f(x) \mid (k_0, k_1) \leftarrow \text{Share}(1^\kappa, f), \\ & y_0 \leftarrow \text{Eval}(0, k_0, x), \\ & y_1 \leftarrow \text{Eval}(1, k_1, x) \Big] \\ & = 1. \end{aligned} \tag{A.1}$$

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

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

FSS Function Families. We list all FSS function families that we use in our PB-FSS constructions in Table A.1, along with their own (Share, Eval) syntax (following the general template in Definition 7).

Function	Incremental Description	Syntax
DPF	$f_{\alpha,a,b}(x) = \begin{cases} a & \text{when } x \not\prec \alpha, \\ b & \text{when } x \prec \alpha. \end{cases}$	DPF.Share($1^\kappa, \alpha, a, b$) DPF.Eval($\text{idx}, k_{\text{idx}}, x$)
LDCF	$f_{\alpha,a,b}(x) = \begin{cases} a & \text{when } x < \alpha, \\ b & \text{when } x \geq \alpha. \end{cases}$	LDCF.Share($1^\kappa, \alpha, a, b$) LDCF.Eval($\text{idx}, k_{\text{idx}}, x$)
RDCF	$f_{\alpha,a,b}(x) = \begin{cases} a & \text{when } x \leq \alpha, \\ b & \text{when } x > \alpha. \end{cases}$	RDCF.Share($1^\kappa, \alpha, a, b$) RDCF.Eval($\text{idx}, k_{\text{idx}}, x$)

Table A.1: (Incremental) DCF variants used in PB-FSS.

A.3 DCF-Based Construction Details of PB-iFSS

The main idea of this construction for the L_∞ distance metric is to share different DCF key pairs per dimension, and carefully choosing the breaking points and payloads for each of these DCF. Later, party idx , to evaluate a coordinate x_i combines the DCF.Eval results using these different keys for dimension i with each other to get the final evaluation.

We explain how to choose the breaking points and the payloads, and show the full construction in Figure A.1. The idea for DCF-based PB-FSS of L_∞ distance has been used before, such as in [9, 20], we simply fix the interval size to be 2δ for our specific use case.

A.3.1 Construction Overview

L_∞ distance metric. For dimension i , we secret share the interval $[q_i - \delta, q_i + \delta]$ with two DCF where breaking points are $q_i - \delta - 1$ and $q_i + \delta$. Formally, the Share algorithm runs **DCF.Share**($q_i - \delta - 1, \alpha_0, \beta_0$) and **DCF.Share**($q_i + \delta, \alpha_1, \beta_1$), with the choice of $\alpha_0, \beta_0, \alpha_1, \beta_1$ being made clear after showing the evaluation algorithm. A party idx , with keys $k_{0,\text{idx}}^{(i)}$ and $k_{1,\text{idx}}^{(i)}$ received from the two DCF.Share calls, evaluates a coordinate x_i by obtaining **DCF.Eval**($k_{1,\text{idx}}^{(i)}, x_i$) - **DCF.Eval**($k_{0,\text{idx}}^{(i)}, x_i$). Following the correctness of DCF, one can show that the evaluation results for the three cases when $x_i < q - \delta$, $q_i - \delta \leq x_i \leq q_i + \delta$, and $x_i > q_i + \delta$ would be $\alpha_1 - \alpha_0$, $\alpha_1 - \beta_0$, and $\beta_1 - \beta_0$, respectively. The payloads $\alpha_0, \beta_0, \alpha_1, \beta_1$ are chosen such that the three mentioned values are 1, 0, 1 accordingly, which leads to secret shares of 0 for evaluation of x_i inside the interval $[q_i - \delta, q_i + \delta]$, and 1 otherwise. The secret shares of 0 could be easily converted into equal values afterward.

A.3.2 DCF-based PB-iFSS for L_∞

We present the DCF-based PB-iFSS construction for the L_∞ distance metric in Figure A.1 and prove its security below.

Share($1^\kappa, L_\infty, (q_1, \dots, q_d), \delta, D, \mathbb{G}$) :

For $i = 1, \dots, d$:

1. Assign $[\alpha_i, \beta_i] = [q_i - \delta, q_i + \delta] \cap [0, 2^u - 1]$.

2. Create pairs of LDCF and RDCF keys:

$$(k_{0,0}^{(i)}, k_{1,0}^{(i)}) \leftarrow \text{LDCF.Share}(\alpha_i, 1, 0), \quad (\text{A.3})$$

$$(k_{0,1}^{(i)}, k_{1,1}^{(i)}) \leftarrow \text{RDCF.Share}(\beta_i, 0, 1), \quad (\text{A.4})$$

Then let $k_0^{(i)} \leftarrow k_{0,0}^{(i)} \| k_{0,1}^{(i)}$ and $k_1^{(i)} \leftarrow k_{1,0}^{(i)} \| k_{1,1}^{(i)}$.

Output $k_b = (k_b^{(1)} \| \dots \| k_b^{(d)})$ for $b = 0, 1$.

Eval($\text{idx}, L_\infty, k_{\text{idx}}, (x_1, \dots, x_d)$) :

Parse the key $(k_{\text{idx}}^{(1)} \| \dots \| k_{\text{idx}}^{(d)}) \leftarrow k_{\text{idx}}$.

Also parse each $k_{\text{idx}}^{(i)}$ into $k_{\text{idx},0}^{(i)} \| k_{\text{idx},1}^{(i)}$. For $i = 1, \dots, d$:

$$y_{\text{idx}}^{(i)} \leftarrow \text{LDCF.Eval}(\text{idx}, k_{\text{idx},0}^{(i)}, x_i) \\ \oplus \text{RDCF.Eval}(\text{idx}, k_{\text{idx},1}^{(i)}, x_i).$$

Output $y_{\text{idx}} = y_{\text{idx}}^{(1)} \| \dots \| y_{\text{idx}}^{(d)}$.

Figure A.1: DCF-based PB-iFSS for L_∞

Theorem 8 (Security of DCF-based PB-iFSS for L_∞). *For L_∞ distance metric, distance threshold δ , number of dimensions d , and domain $\mathcal{D} = (\{0, 1\}^u)^d$, with the corresponding set family $\mathcal{S}_{\infty,d,\delta}$ and property Equal, given LDCF and RDCF scheme defined over domain $\{0, 1\}^u$, the protocol in Figure A.1 realizes an PB-iFSS scheme for the set family $\mathcal{S}_{\infty,d,\delta}$ and property Equal (Definition 1).*

Proof. Correctness for yes-instances: Given $\vec{x} = (x_1, \dots, x_d)$ such that the optimistic evaluation of x is less than or equal to δ , which means, for every dimension $i = 1, \dots, d$, 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 $i = 1, \dots, d$. This directly implies that $y_0 = y_1$, hence $\text{Prop}(y_0, y_1) = 1$.

Correctness for no-instances: Given $\vec{x} = (x_1, \dots, x_d)$ such that the optimistic L_∞ distance of x from α is larger than δ , it means there exists a dimension i_0 such that x_{i_0} is more than δ away from α_{i_0} . By the correctness of LDCF and RDCF, we have $y_0^{(i_0)} \oplus y_1^{(i_0)}$ to be either $1 \oplus 0 = 1$ or $0 \oplus 1 = 1$. Hence $y_0 \oplus y_1$ has digit 1 at position i_0 , and $\text{Prop}(y_0, y_1) = 0$.

Privacy: By the security in Definition 7, there exists a simulator Sim_0 for LDCF, and a simulator Sim_1 for RDCF. The simulator $\text{Sim}(1^\kappa, \text{idx}, d, u)$ for Share does the following:

- For each $i = 1, \dots, d$, get $k_{\text{idx},0}^{(i)} \leftarrow \text{Sim}_0(1^\kappa, \text{idx}, u)$ and $k_{\text{idx},1}^{(i)} \leftarrow \text{Sim}_1(1^\kappa, \text{idx}, u)$.
- Output $k_{\text{idx}} = (k_{\text{idx},0}^{(1)} \| k_{\text{idx},1}^{(1)} \| \dots \| k_{\text{idx},0}^{(d)} \| k_{\text{idx},1}^{(d)})$.

We will prove indistinguishability through a sequence of hybrids. Define $\text{Hybrid}_h(1^\kappa, L_\infty, \alpha, \delta, d, u)$ as follow:

- For $i = 1, \dots, \lfloor h/2 \rfloor$, gets $k_{\text{idx},0}^{(i)} \leftarrow \text{Sim}_0(1^\kappa, \text{idx}, u)$ and $k_{\text{idx},1}^{(i)} \leftarrow \text{Sim}_1(1^\kappa, \text{idx}, u)$.
- If h is odd, assume that $h = 2i - 1$. Get $k_{\text{idx},0}^{(i)} \leftarrow \text{Sim}_0(1^\kappa, \text{idx}, u)$ and generate $(k_{0,1}^{(i)}, k_{1,1}^{(i)})$ as in (A.4) to get $k_{\text{idx},1}^{(i)}$.
- For i from $\lfloor h/2 \rfloor + 2$ to d , generate $(k_{0,0}^{(i)}, k_{1,0}^{(i)})$ as in (A.3) and $(k_{0,1}^{(i)}, k_{1,1}^{(i)})$ as in (A.4). Then get $k_{\text{idx},0}^{(i)}$ and $k_{\text{idx},1}^{(i)}$.
- Output $k_{\text{idx}} = (k_{\text{idx},0}^{(1)} \| k_{\text{idx},1}^{(1)} \| \dots \| k_{\text{idx},0}^{(d)} \| k_{\text{idx},1}^{(d)})$.

Claim 9. *Hybrid 0 is exactly the same as running the Share algorithm honestly:*

$$\begin{aligned} \{k_{\text{idx}} \leftarrow \text{Hybrid}_0(1^\kappa, L_\infty, \alpha, \delta, d, u)\} \equiv \\ \{k_{\text{idx}} : (k_0, k_1) \leftarrow \text{Share}(1^\kappa, L_\infty, \alpha, \delta, d, u)\}. \end{aligned} \quad (\text{A.5})$$

Claim 10. *Hybrid d is exactly the same as the simulator:*

$$\begin{aligned} \{k_{\text{idx}} \leftarrow \text{Hybrid}_d(1^\kappa, L_\infty, \alpha, \delta, d, u)\} \equiv \\ \{k_{\text{idx}} \leftarrow \text{Sim}(1^\kappa, \text{idx}, d, u)\}. \end{aligned} \quad (\text{A.6})$$

The indistinguishability between hybrids follow Lemma 11.

Lemma 11. *For $0 \leq k \leq 2d - 1$, suppose that there exists PPT adversary $\mathcal{A}$ that can distinguish between the k -th Hybrid and the $(k + 1)$ -th Hybrid with non-negligible probability ϵ , or formally:*

$$\begin{aligned} & \left| \Pr \left[k_{\text{idx}} \leftarrow \text{Hybrid}_k(1^\kappa, L_\infty, \alpha, \delta, d, u); \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx}}) : c = 1 \right] \right. \\ & \quad \left. - \Pr \left[k_{\text{idx}} \leftarrow \text{Hybrid}_{k+1}(1^\kappa, L_\infty, \alpha, \delta, d, u); \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx}}) : c = 1 \right] \right| \geq \epsilon. \end{aligned}$$

- *If $k = 2i - 2$ where $1 \leq i \leq d$, then there exists a PPT adversary $\mathcal{B}$ that can attack LDCF with advantage at least ϵ :*

$$\begin{aligned} & \left| \Pr \left[(k_{0,0}^{(i)}, k_{1,0}^{(i)}) \text{ generated from (A.3);} \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx},0}^{(i)}) : c = 1 \right] \right. \\ & \quad \left. - \Pr \left[k_{\text{idx},0}^{(i)} \leftarrow \text{Sim}_0(1^\kappa, \text{idx}, u); \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx},0}^{(i)}) : c = 1 \right] \right| \geq \epsilon. \end{aligned}$$

- *If $k = 2i - 1$ where $1 \leq i \leq d$, then there exists a PPT adversary $\mathcal{B}$ that can attack RDCF with advantage at least ϵ :*

$$\begin{aligned} & \left| \Pr \left[(k_{0,0}^{(i)}, k_{1,0}^{(i)}) \text{ generated from (A.4);} \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx},0}^{(i)}) : c = 1 \right] \right. \\ & \quad \left. - \Pr \left[k_{\text{idx},0}^{(i)} \leftarrow \text{Sim}_1(1^\kappa, \text{idx}, u); \right. \right. \\ & \quad \left. \left. c \leftarrow \mathcal{A}(1^\kappa, k_{\text{idx},0}^{(i)}) : c = 1 \right] \right| \geq \epsilon. \end{aligned}$$

□

Appendix B

Sketching Details

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 construction 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 [7], this issue still occurs: even though the incremental DPF construction in [7] guarantees that each layer has at most one potentially non-zero entry, this entry can in fact 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.

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 $\mathbb{Z}_q$ without a blowup in complexity.

In this section, to not over-complicate 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.

B.1 Sketching for L_∞

In this section, we provide intuition on how to sketch PB-FSS when the payloads are elements from a ring with large enough modulo $\mathbb{Z}_q$. We first give a sketching method for PB-FSS, which concerns sketching a DCF as well as checking if two DCF are shifted by a given amount. We then extend sketching to the incremental cases, by giving a sketching method for checking consistency between evaluation layers of PB-iFSS.

Sketch for PB-FSS. Recall that the PB-FSS key for L_∞ , given a distance threshold δ , contains two DCF keys at breaking points $q - \delta - 1$ and $q + \delta$. If the client is honest, the two servers would receive two vectors of the form $(1, 1, \dots, 1, 0, 0, \dots, 0)$, which we refer to as cumulative vectors, and the last 1-index is again, referred to as breaking point.

Let the output that two servers receive after full domain evaluation of the two DCF key pairs be secret shares for two vectors $[\![\vec{u}]\!]$ and $[\![\vec{v}]\!]$. If the client is honest, both $\vec{u}$ and $\vec{v}$ would be cumulative vectors, with breaking points being $2\delta + 1$ apart from each other. We can check that $\vec{v}$ is a cumulative vector with breaking point $q - \delta$ by taking differences of consecutive elements $v_i - v_{i+1}$, which would turn $\vec{v}$ into a unit vector with the only non-zero entry at index $q + \delta - 1$. Then, the sketching technique for unit vector is well-known¹ [9]. We are not done yet as the two breaking points are also required to be $2\delta + 1$ away from each other. This sketching step is done by shifting $[\![\vec{v}]\!]$ to the left by $2\delta + 1$ indices, into secret shares of a vector $[\![\vec{v}']]\!$, then taking a random linear combination of all entries in $[\![\vec{u}]\!] - [\![\vec{v}']]\!$ to check whether $\vec{u}$ and $\vec{v}'$ are equal. We describe this sketching scheme below.

Sketch. The servers receive from the client additive shares of two vectors $[\![\vec{u}]\!]$ and $[\![\vec{v}]\!]$. If the client is honest, both $\vec{u}$ and $\vec{v}$ would be vectors of the form $(1, 1, \dots, 1, 0, 0, \dots, 0)$,

¹In fact, this idea for sketching of DCF has also been mentioned in [9].

where the last 1-index of $\vec{u}$ is 2δ positions to the right of the last 1-index of $\vec{v}$. Both parties do the following:

1. Obtain $\llbracket \vec{u}' \rrbracket = \llbracket (u_0 - u_1, u_1 - u_2, \dots, u_{N-2} - u_{N-1}, 0) \rrbracket$. We also obtain $\llbracket \vec{v}' \rrbracket$ in the same way.
2. Jointly sample $\vec{r}^1 = (r_0, r_1, \dots, r_{N-1}) \in \mathbb{Z}_q^N$. Let powers of $\vec{r}$ be $\vec{r}^i = (r_0^i, r_1^i, \dots, r_{N-1}^i)$ for each $i = 2, 3, 6, 4$.
3. Compute the inner products $\llbracket z_u \rrbracket = \langle \vec{r}^1, \llbracket \vec{u}' \rrbracket \rangle$ and $\llbracket z'_u \rrbracket = \langle \vec{r}^2, \llbracket \vec{u}' \rrbracket \rangle$ for sketching whether $\vec{u}'$ is a unit vector.
4. Compute the inner products $\llbracket z_v \rrbracket = \langle \vec{r}^3, \llbracket \vec{v}' \rrbracket \rangle$ and $\llbracket z'_v \rrbracket = \langle \vec{r}^6, \llbracket \vec{v}' \rrbracket \rangle$ for sketching whether $\vec{v}'$ is a unit vector.
5. Obtain $\llbracket \vec{u}_\delta' \rrbracket = \llbracket (u'_{2\delta}, u'_{2\delta+1}, \dots, u'_{N-1}, 0, 0, \dots, 0) \rrbracket$ being the vector $\vec{u}'$, shifted 2δ positions to the left, and pad to the right with zeros.
6. Compute the inner product $\llbracket z_\delta \rrbracket = \langle \vec{r}^4, \llbracket \vec{u}_\delta' \rrbracket - \llbracket \vec{v}' \rrbracket \rangle$ for sketching whether $\vec{u}'' = \vec{v}'$.
7. The obtained secret shares $\llbracket z_u \rrbracket, \llbracket z'_u \rrbracket, \llbracket z_v \rrbracket, \llbracket z'_v \rrbracket, \llbracket z_\delta \rrbracket$ are the sketching outputs.

Decision. The parties run secure 2PC to check whether the following value is 0:

$$(z_u^2 - z'_u) + (z_v^2 - z'_v) + z_\delta. \quad (\text{B.1})$$

Sketch for PB-iFSS. One need to make sure that the full domain evaluation vectors in each layer are consistent to sketch for PB-iFSS. We give intuition on sketching for an incremental DCF input length h (prefixes of length at most h). Doing full domain evaluation on all prefixes gives secret shares of h vectors $\llbracket \vec{u}^1 \rrbracket, \llbracket \vec{u}^2 \rrbracket, \dots, \llbracket \vec{u}^h \rrbracket$, correspond to h possible prefix lengths.

Our task would be to check that each of the h vectors $\vec{u}^1, \vec{u}^2, \dots, \vec{u}^h$ are all cumulative vectors. Furthermore, their breaking points are all prefixes of a single value q . The general idea is to check that every two consecutive layers are consistent, more formally, if the cumulative vector $\vec{u}^{i+1}$ has breaking point q_{i+1} being a prefix of q , then the previous vector $\vec{u}^i$ should have breaking point q_i satisfying either $q_{i+1} = 2q_i$, or $q_{i+1} = 2q_i + 1$, making sure this is prefix of q as well. We do this check by first

duplicating $\llbracket \vec{u}^i \rrbracket$, making every entry repeat twice, and obtain secret shares of $\llbracket \vec{u}^{i*} \rrbracket$, which would be equal to either $\llbracket \vec{u}^{i+1} \rrbracket$ or this vector shifted by one position to the left. The remaining task is to take two random linear combinations for equality check for these two cases, which would return two secret-shared values that can be multiplied (in 2PC) together, getting a product equal to zero if either case holds true.

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 $\llbracket \vec{u}^i \rrbracket$ and $\llbracket \vec{v}^i \rrbracket$ are shifted from each other an amount equals to a prefix of $2\delta + 1$. We describe this sketching scheme below.

Sketch. The servers receive from the client subtractive shares of two vectors $\llbracket \vec{u} \rrbracket$ of length N and $\llbracket \vec{v} \rrbracket$ of length $2N$. If the client is honest, both $\vec{u}$ and $\vec{v}$ would be vectors of the form $(\beta, \beta, \dots, \beta, 0, 0, 0)$, where the last β -index of $\vec{u}$ is denoted α , and the last β -index of $\vec{v}$ is either 2α or $2\alpha + 1$. Both parties do the following:

1. Obtain $\llbracket \vec{u}' \rrbracket = \llbracket (u_0 - u_1, u_1 - u_2, \dots, u_{N-2} - u_{N-1}, 0) \rrbracket$. We also obtain $\llbracket \vec{v}' \rrbracket$ in the same way.
2. Jointly sample $\vec{r}^1 = (r_0, r_1, \dots, r_{2N-1}) \in \mathbb{Z}_q^{2N}$. Let $\vec{r}^i = (r_0^i, r_1^i, \dots, r_{2N-1}^i)$ for each $i = 2, 3, 6, 4$.
3. Compute the inner products $\llbracket z_u \rrbracket = \langle \vec{r}^1, \llbracket \vec{u}' \rrbracket \rangle$ and $\llbracket z'_u \rrbracket = \langle \vec{r}^2, \llbracket \vec{u}' \rrbracket \rangle$ for sketching whether $\vec{u}'$ is a unit vector.
4. Compute the inner products $\llbracket z_v \rrbracket = \langle \vec{r}^3, \llbracket \vec{v}' \rrbracket \rangle$ and $\llbracket z'_v \rrbracket = \langle \vec{r}^6, \llbracket \vec{v}' \rrbracket \rangle$ for sketching whether $\vec{v}'$ is a unit vector.
5. Obtain $\llbracket \vec{u}^* \rrbracket = \llbracket (u'_0, u'_0, u'_1, u'_1, \dots, u'_{N-1}, u'_{N-1}) \rrbracket$ and $\llbracket \vec{u}^\bullet \rrbracket = \llbracket (u'_0, u'_1, u'_1, \dots, u'_{N-1}, u'_{N-1}, 0) \rrbracket$. Here, $\vec{u}^*$ is the vector $\vec{u}'$ with each element duplicated twice, and $\vec{u}^\bullet$ is the vector $\vec{u}^*$ left-shifted by 1 position, and pad into length $2N$.
6. Compute the inner product $\llbracket z_* \rrbracket = \langle \vec{r}^4, \llbracket \vec{u}^* \rrbracket - \llbracket \vec{v}' \rrbracket \rangle$ for sketching where $\vec{u}^* = \vec{v}'$.
7. Compute the inner product $\llbracket z_\bullet \rrbracket = \langle \vec{r}^5, \llbracket \vec{u}^\bullet \rrbracket - \llbracket \vec{v}' \rrbracket \rangle$ for sketching where $\vec{u}^\bullet = \vec{v}'$.

8. The obtained secret shares $\llbracket z_u \rrbracket, \llbracket z'_u \rrbracket, \llbracket z_v \rrbracket, \llbracket z'_v \rrbracket, \llbracket z_* \rrbracket, \llbracket z_\bullet \rrbracket$ are the sketching outputs.

Decision. The parties run secure multiparty computation to check whether the following value is 0:

$$(z_u^2 - z'_u) + (z_v^2 - z'_v) + z_* \cdot z_\bullet. \quad (\text{B.2})$$

B.2 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 $\mathbb{Z}_q$ (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 $\llbracket \vec{u} \rrbracket$ 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 $\llbracket u_i \rrbracket_0$ and $\llbracket u_i \rrbracket_1$ such that $\llbracket u_i \rrbracket_0 + \llbracket u_i \rrbracket_1 = 0 \bmod 2^k$. By letting $\mathcal{S}_0$ obtain $a_i = \llbracket u_i \rrbracket_0$, and $\mathcal{S}_1$ obtain $b_i = -\llbracket u_i \rrbracket_1 \bmod 2^k$, 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 $\mathbb{Z}_q$, and it would still be the case that $a_i = b_i$. The two servers can finish sketching by sampling common random values $r_0, r_1, \dots, r_{2^{h_1}-1} \in \mathbb{Z}_q$, then compute $\sum_{i=0}^{2^{h_1}-1} r_i a_i \bmod q$ and $\sum_{i=0}^{2^{h_1}-1} r_i b_i \bmod q$ 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 $\llbracket \vec{u} \rrbracket$ 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 $a_i = \llbracket u_i \rrbracket_0$ and $b_i = -\llbracket u_i \rrbracket_1 \bmod 2^k$, 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 $\mathbb{Z}_q$. This means, in $\mathbb{Z}_q$, the two vectors $\vec{a}$

and $\vec{b}$ are actually *subtractive shares* for a weight-one vector in $\mathbb{Z}_q$, where the non-zero entry is either 1 or $1 - 2^k \bmod q$. Here comes the magic: we can check the second case by multiplying the secret shares with a “rescaling” vector in modulo q , that contains all inverses of $1 - 2^k$ in modulo q : $((1 - 2^k)^{-1} \bmod q, \dots, (1 - 2^k)^{-1} \bmod q)$.

In conclusion, there are two cases: either $\vec{a}$ and $\vec{b}$ are subtractive shares of a unit vector in $\mathbb{Z}_q$, or by multiplying $\vec{a}$ and $\vec{b}$ with the “rescaling” vector in modulo q , 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 $\llbracket \vec{u} \rrbracket$ 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 folklore, since we can take difference between consecutive elements in $\mathbb{Z}_{2^k}$ to turn a cumulative vector into a unit vector in $\mathbb{Z}_{2^k}$. The formal description of this sketching protocol is shown below.

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

1. Obtain $\llbracket \vec{u}' \rrbracket = \llbracket (u_1 - u_0, u_2 - u_1, \dots, u_{N-1} - u_{N-2}, 0) \rrbracket$. Let the share obtained by $\mathcal{S}_0$ be $\llbracket \vec{u}' \rrbracket_0$ and the share obtained by $\mathcal{S}_1$ be $\llbracket \vec{u}' \rrbracket_1$. If the client is honest, the two servers have $\llbracket u'_i \rrbracket_0 = \llbracket u'_i \rrbracket_1$ at all but one value $i \in [N - 1]$.
2. Jointly sample $\vec{r}^1 = (r_0, r_1, \dots, r_{N-1}) \in \mathbb{Z}_q^N$. Let the powers of $\vec{r}$ be $\vec{r}^i = (r_0^i, r_1^i, \dots, r_{N-1}^i) \in \mathbb{Z}_p$ for each $i = 2, 3, 6$.
3. Obtain $\llbracket \vec{u}_* \rrbracket_b = \left(\frac{\llbracket u'_0 \rrbracket_b}{1}, \frac{\llbracket u'_1 \rrbracket_b}{1}, \dots, \frac{\llbracket u'_{N-1} \rrbracket_b}{1} \right) \bmod q$ as the “rescaled” vector for the first case.
4. Obtain $\llbracket \vec{u}_\bullet \rrbracket_b = \left(\frac{\llbracket u'_0 \rrbracket_b}{1-2^k}, \frac{\llbracket u'_1 \rrbracket_b}{1-2^k}, \dots, \frac{\llbracket u'_{N-1} \rrbracket_b}{1-2^k} \right) \bmod q$ 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 $\mathbb{Z}_q$.
5. Compute the inner products $\llbracket z_* \rrbracket = \langle \vec{r}^1, \llbracket \vec{u}_* \rrbracket \rangle$ and $\llbracket z'_* \rrbracket = \langle \vec{r}^2, \llbracket \vec{u}_* \rrbracket \rangle$ for testing whether $\vec{u}_*$ is a unit vector.

6. Compute the inner products $\llbracket z_{\bullet} \rrbracket = \langle \vec{r}^1, \llbracket \vec{u}_{\bullet} \rrbracket \rangle$ and $\llbracket z'_{\bullet} \rrbracket = \langle \vec{r}^2, \llbracket \vec{u}_{\bullet} \rrbracket \rangle$ for testing whether $\vec{u}_{\bullet}$ is a unit vector.
7. The obtained secret shares $\llbracket z_* \rrbracket, \llbracket z'_* \rrbracket, \llbracket z_{\bullet} \rrbracket, \llbracket z'_{\bullet} \rrbracket$ are the sketching outputs.

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

$$(z_*^2 - z'_*) \cdot (z_{\bullet}^2 - z'_{\bullet}). \quad (\text{B.3})$$

Appendix C

Security definitions and proofs

In this section, we formally define the ideal functionalities for the fuzzy heavy hitters problem in the known and unknown dictionary settings in Figure C.1 and Figure C.2, respectively and show that Mosaic (see Figure 3.4) realizes the ideal functionalities.

Theorem 12. *For a distance metric dist with corresponding set family and property $(\mathcal{S}, \text{Prop})$, given PB-FSS scheme for $\mathcal{S}$ and Prop , the protocol in Figure 3.4 realizes $\mathcal{F}_{\text{known}}$ (Figure C.1) in the $\mathcal{F}_{\text{PropTest}}$ and $\mathcal{F}_{\text{Comp}}$ hybrid model, in the presence of semi-honest adversaries.*

Proof. **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 $(k_0^{(1)}, \dots, k_0^{(N)})$, one from each of the N clients, b) output $\llbracket b_i^{(\vec{w})} \rrbracket_0$ of ideal functionality $\mathcal{F}_{\text{PropTest}}$ for each $\vec{w} \in D$, c) output $o_{\vec{w}}$ from the ideal functionality $\mathcal{F}_{\text{Comp}}$, 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 $(\vec{q}_1, \dots, \vec{q}_N)$.

- **Hybrid₀** : We start with the real execution of the protocol, where $\mathcal{S}_0$ interacts with the honest set of clients $\mathcal{C}$ and $\mathcal{S}_1$.

- **Hybrid₁** : This hybrid is identical to the real execution, except that we invoke the PB-iFSS simulator $\text{Sim}(1^\kappa, 0, u, d)$ 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 1](#)), this modification is indistinguishable except with negligible probability from the previous hybrid.
- **Hybrid₂** : For each input $\vec{w} \in D$, we replace the output $\llbracket b_i^{(\vec{w})} \rrbracket_0$ of private property test ideal functionality $\mathcal{F}_{\text{PropTest}}$ (Figure 3.1) with uniformly random outputs. This is identically distributed as previous hybrid, by simulatability of the ideal functionality.
- **Hybrid₃** : Again, we modify the output received from ideal functionality $\mathcal{F}_{\text{comp}}$. For each input $\vec{w} \in D$, if $\vec{w} \in H$ is in the output of $\mathcal{F}_{\text{known}}$, then forward $o_{\vec{w}} = 1$ as output from $\mathcal{F}_{\text{comp}}$. Otherwise, send $o_{\vec{w}} = 0$.
- *Simulator*: We see that **Hybrid₃** defines a valid simulation, independent of client's input $(\vec{q}_1, \dots, \vec{q}_N)$.

□

Theorem 13. *For a distance metric dist with corresponding set family and property $(\mathcal{S}, \text{Prop})$, given PB-FSS scheme for $\mathcal{S}$ and Prop , the protocol in Figure 3.4 realizes $\mathcal{F}_{\text{unknown}}$ (Figure C.2) in the $\mathcal{F}_{\text{PropTest}}$ and $\mathcal{F}_{\text{Comp}}$ hybrid model, in the presence of semi-honest adversaries.*

Proof. **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 $(k_0^{(1)}, \dots, k_0^{(N)})$, one from each of the N clients, b) output $\llbracket b_i^{(\vec{w})} \rrbracket_0$ of ideal functionality $\mathcal{F}_{\text{PropTest}}$ for each search prefix $\vec{w}$, c) output $o_{\vec{w}}$ from the ideal functionality $\mathcal{F}_{\text{comp}}$, for each search prefix $\vec{w}$. We start with the real execution of the protocol in **Hybrid₀** and through a sequence of hybrids move to a simulated view in the ideal world that is independent of the clients' input $(\vec{q}_1, \dots, \vec{q}_N)$.

- **Hybrid₀** : 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 $\mathcal{S}_1$.
- **Hybrid₁** : We modify this hybrid as follows: invoke the PB-iFSS simulator $\text{Sim}(1^\kappa, \text{dist}, 0, u, d)$ 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 1](#)), this hybrid is indistinguishable from real execution except with negligible probability.
- **Hybrid₂** : For each search prefix $\vec{w}$ and for each client $\mathcal{C}_i \in \mathcal{C}$, we replace the output $\left\llbracket b_i^{(\vec{w})} \right\rrbracket_0$ of property test ideal functionality $\mathcal{F}_{\text{PropTest}}$ (Figure 3.1) with uniformly random outputs. This is identically distributed as previous hybrid, by simulatability of the ideal functionality $\mathcal{F}_{\text{PropTest}}$.
- **Hybrid₃** : Again, we modify the output received from ideal functionality $\mathcal{F}_{\text{comp}}$. For each search prefix $\vec{w}$, if $\vec{w} \in H$ is in the output of $\mathcal{F}_{\text{unknown}}$, then forward $o_{\vec{w}} = 1$ as output from $\mathcal{F}_{\text{comp}}$. Otherwise, send $o_{\vec{w}} = 0$. Additionally, we handle leakage, by forwarding every prefix $p_{\vec{w}} \in \mathcal{L}$ from the ideal functionality $\mathcal{F}_{\text{unknown}}$ to $\mathcal{S}_0$.
- *Simulator*: Observe that, Hybrid₃ defines a valid simulation, independent of client's input $(\vec{q}_1, \dots, \vec{q}_N)$.

□

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 $\mathcal{C} = (\mathcal{C}_1, \dots, \mathcal{C}_N)$.

Functionality:

1. Receive from both servers $\mathcal{S}_0, \mathcal{S}_1$ a dictionary $\mathcal{D} = \{\vec{x}_1, \dots, \vec{x}_M\}$ 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 $\mathcal{S}_1$:

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

Figure C.1: Ideal functionality $\mathcal{F}_{\text{known}}$ 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 $\mathcal{C} = (\mathcal{C}_1, \dots, \mathcal{C}_N)$

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

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

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:

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

where dist_{opt} denotes the optimistic distance between the prefix $\vec{x}$ and the full vector $\vec{q}_j$ defined in Section 3.1.

4. **[output]** Send output list of heavy hitters H to both servers $\mathcal{S}_0$ and $\mathcal{S}_1$.

Figure C.2: Ideal functionality $\mathcal{F}_{\text{unknown}}$ for the unknown-dictionary fuzzy heavy hitters problem.