ProxCode: Efficient Biometric Proximity Searchable Encryption from Error Correcting Codes

Maryam Rezapour* Benjamin Fuller†

December 12, 2024

Abstract

This work builds approximate proximity searchable encryption. Secure biometric databases are the primary application. Prior work (Kuzu, Islam, and Kantarcioglu, ICDE 2012) combines locality-sensitive hashes, or LSHs, (Indyk, STOC '98), and secure multimaps. The multimap associates LSH outputs as keywords to biometrics as values.

When the desired result set is of size at most one, we show a new preprocessing technique and system called `ProxCode` that inserts shares of a linear secret sharing into the map instead of the full biometric. Instead of choosing shares independently, shares are correlated so exactly one share is associated with each keyword/LSH output. As a result, one can rely on a map instead of a multimap. Secure maps are easier to construct with low leakage than multimaps.

For many parameters, this approach reduces the required number of LSHs for a fixed accuracy. Our scheme yields the most improvement when combining a high accuracy requirement with a biometric with large underlying noise. Our approach builds on any secure map. To benchmark, we implement the scheme using the tree based oblivious map of Wang et al. (CCS, 2014) and evaluate efficiency and accuracy for both iris data and random data.

1 Introduction

This work builds approximate proximity searchable encryption, called APSS [KIK12,BT21,HCD+23,FWG+16, WYLH14, LPW+20]. Secure biometric databases [BBOH96, Dau14, Fou] are a major application of APSS. Searchable encryption considers two parties a client, denoted as Client, and a server, denoted as Server. The goal of the searchable encryption is for Client to outsource a database, $\mathcal{DB}$, to Server and be able to query this outsourced database. The system should limit what is learned by a semi-honest Server while still allowing the client to retrieve the relevant results from the database. See prior reviews [BHJP14, FVY+17, KKM+22, RW23,IKK12] and work on searchable encryption [CGPR15,KKNO16,WLD+17,GSB+17,GLMP18,KPT19, MT19, KE19, KPT20, FMC+20, FP22, GPP23, APP+23, HKR+24].

We now set notation of APSS and provide a review of prior work. Let $\mathcal{DB} = x_1, ..., x_M$ be a collection of records. For metric $\mathcal{D}$, a distance parameter t, and query y, the goal of the search is to find the set $\texttt{Res} = \{x_i \in \mathcal{DB} | \mathcal{D}(x_i, y) \leq t\}$.

When a biometric database is used for 1-to-n matching, one assumes that for all y there exists at most one $x \in \mathcal{DB}$ such that $\mathcal{D}(y, x) \leq t$. Multimaps [KM19], MM, allow association of keywords with values x_i.[1] An encrypted multimap is the same except there are interactive protocols between the Client and Server. We use the following notation for the encrypted version (with the Client's inputs and outputs on the top row

*University of Connecticut. Email: maryam.rezapour@uconn.edu.

†University of Connecticut. Email: benjamin.fuller@uconn.edu.

[1] A multimap has two operations: 1) $\mathsf{MM}.\texttt{Init}(\{\texttt{keyword}, \texttt{value}\})$ that takes a set of keyword, value pairs and associates value with keyword, and 2) $\mathsf{Resp} \leftarrow \mathsf{MM}[\texttt{keyword}]$ which returns all values associated with value.

and the Server's inputs and outputs on the bottom row):

$$\begin{pmatrix} K \\ \mathsf{EMM} \end{pmatrix} \leftarrow \mathsf{MM.Setup} \begin{pmatrix} \mathsf{MM} \\ \perp \end{pmatrix}$$

$$\begin{pmatrix} \mathsf{MM}[\texttt{keyword}] \\ \perp \end{pmatrix} \leftarrow \mathsf{MM.Search} \begin{pmatrix} \texttt{keyword}, K \\ \mathsf{EMM} \end{pmatrix}$$

Most work in APSS uses preprocessing techniques and encrypted multimaps. The goal of preprocessing is to create accurate search while minimizing the size of the multimap, which impacts all cryptographic efficiency.

Prior work [KIK12, BT21, HCD+23] use locality sensitive hashes or LSHs [IM98] to preprocess. We provide a brief overview of LSHs and then explain how they are used for preprocessing. LSHs map close items to the same value more frequently than they map far items to the same value. For some n number of LSHs, a multimap MM, and LSH family LSH, the Client proceeds as follows, $\mathsf{APSS.Init}(\mathcal{DB} = x_1, ..., x_M)$:

1. Sample n LSHs, $\mathsf{LSH}_1, ..., \mathsf{LSH}_n \leftarrow \mathsf{LSH}$.
2. For $j = 1, ..., n$ & $i = 1, ..., M$,
$$\mathsf{MM.add}(\texttt{keyword} = (j, \mathsf{LSH}_j(x_i)), \texttt{value} = x_i).$$
3. Store MM on Server by executing
$$\begin{pmatrix} K \\ \mathsf{EMM} \end{pmatrix} \leftarrow \mathsf{MM.Init} \begin{pmatrix} \mathsf{MM} \\ \perp \end{pmatrix}$$

Then, $\mathsf{APSS.SearchSearch}(y)$ is:

1. Client computes $\mathsf{LSH}_1(y), ..., \mathsf{LSH}_n(y)$.
2. For $i = 1$ to n:
$$\begin{pmatrix} \mathsf{Resp}_i \\ \perp \end{pmatrix} \leftarrow \mathsf{MM.Search} \begin{pmatrix} \mathsf{LSH}_i(y), K \\ \mathsf{EMM} \end{pmatrix}$$
3. Let Client response be $\mathsf{Resp} = (\mathsf{Resp}_1, ..., \mathsf{Resp}_n)$.

There are two aspects to understand for the above construction for a fixed number of LSH's n: security and accuracy. We discuss these issues below:

Security. The security of the above construction derives from the encrypted multimap. Constructions of (dynamic) multimaps [SWP00, KMO18, GKM21, GPP23, AG22, RW23, APP+23, WSL+22, PPYY19][2] have nonzero leakage including query equality. Patel et al. [PPYY19] showed that avoiding query equality leakage for multimaps requires higher overhead techniques similar to oblivious RAM.

Accuracy. LSH-based preprocessing solutions have imperfect accuracy. The two accuracy parameters are:

1. $\delta_{\texttt{Close}}$ measures how frequently the close record is not returned, and
2. $\delta_{\texttt{Far}}$ measures what fraction of the database is (incorrectly) returned.

High accuracy systems have three advantages:

1. Returned records are more likely to be relevant.
2. A decrease in the maximum number of values associated with a keyword, a key efficiency metric for secure multimaps. If this is made to a small constant, one can use a map instead.
3. In a three party system where the querier doesn't know the whole dataset it reduces unintentional exposure of biometrics (discussion in Section 1.2).

[2]This generation of low leakage maps followed attacks on the prior generation of map constructions [CGPR15, KKNO16, ZKP16, GLMP18, GLMP19, GJW19, KPT20, DHP21, OK21].

1.1 Our Contribution

This work contributes 1) a data preprocessing method for accurate proximity search, 2) an implementation of this processing technique using an oblivious tree-based map [WNL+14], and 3) an evaluation of accuracy and efficiency on iris and random data. Our approach is to transform the query from a disjunction, where one concatenates the list of all records that match some LSH, to a k-out-of-n query, where one retrieves at most a single record that matches at least k LSHs. We call the system `ProxCode` for efficient proximity search from error correcting codes.[3] The high-level approach proceeds in five steps:

1. Compute n LSHs and for every record x, compute $\mathsf{LSH}_i(x)$ for $1 \leq i \leq n$.
2. For each x_i, define $P_{x_i} = \{p \mid \text{degree } \leq k \wedge p(0) = x_i\}$,[4]
3. For each pair $1 \leq i, j \leq M$ with $i < j$ do the following:
   (a) If p_{x_i} has not been defined, set $p_{x_i} \leftarrow P_{x_i}$ uniformly.
   (b) For each $1 \leq \beta \leq n$ if $\mathsf{LSH}_\beta(x_i) = \mathsf{LSH}_\beta(x_j)$ remove those $p_j \in P_{x_j}$ such that $p_i(\beta) \neq p_j(\beta)$.
   (c) If $P_{x_j} = \emptyset$, go to step 1.
4. Set p_{x_M} as a uniformly random element of P_{x_M}
5. Associate in the map for all i, j
$$\mathsf{Map}[(j, \mathsf{LSH}_j(x_i))] = p_{x_i}(j).$$

At search time, one queries the map for all $(j, \mathsf{LSH}_j(x'))$ and collects results from the map and uses polynomial interpolation to reconstruct the relevant record (or record position).

Roughly, instead of associating x_i with LSH keyword, one associates polynomial points. If one gets enough points on the same polynomial one can reconstruct the original value. Each LSH collision between records x_i, x_j where $i < j$ of the database removes a degree of freedom from the polynomial or P_{x_j} but does not require one to associate multiple values with the same LSH keyword. The procedure succeeds as long as no record x_j has no more than $k-1$ LSH collisions with other x_i (More precisely, more than $k-1$ distinct symbols have collisions).

1.1.1 Improving n for fixed accuracy

Our analysis shows for the error regimes present in biometrics one can sample a set of M polynomials using fewer LSHs than the Baseline disjunctive search's requirement described by Ha et al. [HCD+23].

Mean FHD (Fractional Hamming Distance) is the Hamming distance divided by the length of the vector. FHD for biometrics varies between $10\% - 30\%$ depending on the biometric, collection conditions, and the feature extractor. Ha et al. [HCD+23] point out it is necessary to consider distance higher than the mean of biometric error rate to achieve low $\delta_{\texttt{Close}}$. They consider the iris with mean FHD of $\approx 20\%$. For one accuracy regime, their analysis suggests $n = 80$ LSHs suffices to capture the distribution mean but $n \approx 1000$ LSHs are needed to capture the distribution tail.

We show the improvement in the number of required LSHs in Table 1 for a dataset of size $10,000$. This directly translates to the overall size of the $|\mathsf{Map}|$ that must be stored. Many cryptographic constructions have storage of $\omega(|\mathsf{Map}|)$. The accuracy level of $\delta_{\texttt{Far}}$ presented in Table 1 has a slightly different meaning for the baseline and `ProxCode`. Roughly for `ProxCode` it is the probability of a query returning a far value. For the baseline, it is a traditional false accept rate or `FAR`. Our efficiency improvements are highest for high-accuracy regimes with large underlying biometric noise.

There are four primary statistics that matter from the combination of biometric and the feature extractor: the means and variance of the FHD comparison for readings of the same biometric and different biometric. Reducing the variance of either is beneficial for us, yielding fewer LSHs. We care about how different the

[3] In the majority of our discussion, we use the notation of Reed-Solomon codes, as we discuss in the Conclusion, it may be possible to achieve better parameters with custom codes. We use map notation instead of multimap notation. In our system one only associates one value with each keyword.

[4] The system works perfectly well if one associates the value i and uses a separate mechanism to retrieve x_i from i. As we discuss in Section 3, associating x_i prevents a second lookup with associated leakage [GPPW24, GPP23].

Error Rate	Improvement ($\log_{10}(n)$)				
	.10	.15	.20	.25	.30
$\delta_{\texttt{Far}} = 10^{-3}$	-0.5	-0.2	0.3	2.1	1.3
$\delta_{\texttt{Far}} = 10^{-4}$	-0.2	0.1	0.7	1.6	2.8
$\delta_{\texttt{Far}} = 10^{-6}$	0	0.6	1.4	2.4	3.9

Table 1: Summary of improvement in number of required LSHs across biometric error rates ($1 - \epsilon'_t$) and accuracy of the scheme with respect to false accepts denoted as $\delta_{\texttt{Far}}$, fully described in Appendix B. We note the substantial improvement for the high error rate regime. Dataset of size $M = 10^4$.

System name	Records M	LSHs n	$\lvert\mathsf{Map}\rvert$	Setup	Storage	Seq. Search	Par. Search
Ha et al. [HCD+23]	25000	3500	87.5M	153 hrs	285 GB	7.9	.035
`ProxCode`	50000	1119	56.0M	23 hrs	73 GB	10.5	.006

Table 2: Comparison of largest tested parameters. In Ha et al. [HCD+23] there is a filtering phase that finds non-null values and only searches for 20 items in the map. As we discuss in Section 4, if one implements a similar phase in our scheme we estimate the sequential time to be .44 seconds. The fairest comparison is parallel search time which is agnostic to such an optimization.

means are, this is shown in Table 7. These quantities are determined by a variety of environmental factors and the feature extractor.

For random data, setup always succeeds with predicted parameters. We generate queries with the same distribution as in prior work [HCD+23], we observe a $\delta_{\texttt{Close}} = 0$ until the query error rate is $> 120\%$ of the mean error rate (Table 7), at this setting $\delta_{\texttt{Close}} = .2$. The mean error rate is used to set parameters.

Table 2 shows a concrete comparison between the largest tested parameters of Ha et al. and this work. In both works the TAR is set to .9 ($1 - \delta_{\texttt{Close}}$). On a dataset twice as large as Ha et al., `ProxCode` uses 1/3 the number of LSHs. This results in a smaller map, less storage, and faster parallel search. (As we discuss in Section 4, Ha et al. use a filtering technique to search fewer items in the map, this technique works in our setting as well but is not reported. Parallel time is the appropriate comparison.)

1.1.2 Security and Cryptographic overhead

`ProxCode` can be secured using any map. Our implementation and efficiency results consider an oblivious tree-based map [WNL+14]. We consider security when instantiated with 1) a map that leaks search [LZWaT14, OK21] & access pattern [SWP00,CGKO06] and 2) an oblivious map [Mic97,WNL+14,BT21]. When the map reveals query and access pattern, our scheme reveals the query and access pattern for each individual search term which we call subquery and subaccess pattern leakage using the language of Falzon et al. [FMET22].

If one uses a map with access pattern leakage, subquery and subaccess leakage are in 1-1 correspondence. We observe roughly 2000 subquery repeats on 200 queries when $n \in [1000, 30000]$. This is on queries of different irises. One expects much higher subquery equality if multiple readings of the same iris are in the set of queries.

We present a prototype implementation of the above `ProxCode` scheme integrated into an implementation of Wang et al.'s [WNL+14] oblivious map (Github). We evaluate accuracy and overhead on 1) the IITD iris data set [KP10] for realism using the ThirdEye feature extractor [AF19a] and 2) random data to scale beyond the hundreds of biometrics in existing datasets.

Table 2 compares the largest tested parameters between `ProxCode` and Ha et al. [HCD+23]. The main comparison point is the number of LSHs which improves all other algorithms. Some of the improvements are due to a more efficient map implementation. See further results in Section 4.

1.2 Implications for Client Security.

Throughout the body of this work, we define a traditional two-party setting of searchable encryption where there is a data owner that outsources data to a server. This is done for simplicity. Some searchable encryption systems operate in the three party setting where there is a data owner, server, and a client [FMC+15,

HSWW18, WP21]. In this setting, the database contents and queries are both private. In this setting, the data owner gives the client a token that allows them to execute their query. Our scheme shows that the client is unlikely to gain enough code symbols to learn anything about any records far from their query. In the body, we measure for real data how many code symbols are gained by a client across multiple queries. We compare this to the number of biometrics that are completely leaked using the baseline setting (Section 4.3). In `ProxCode`, learning anything about a non-queried biometric requires a persistent adversary [GRS17] with many queries.

Further Prior Work. Ha et al. [HCD+23] presented a zero-leakage iris proximity search system called private-eyes. Private-eyes is built using the oblivious tree-based map of Wang et al. [WNL+14].

Boldyreva and Tang considered the related problem of zero-leakage k-nearest-neighbor search [BT21]. In (approximate) k-nearest neighbors, the goal is to retrieve the k closest records [WHL16, WHL20, BT21]. There have been leakage abuse attacks against k-nearest neighbor systems that reveal access pattern [KPT19, KPT20, LMWY20, CCD+20].

Organization. Section 2 introduces preliminary notation including the definition of APSS. Section 3 presents `ProxCode` and proves that it is a secure APSS and is correct under the assumption that all data in the database is far apart (this condition is an analytic tool that is not used in our testing on real/random data). Section 4 presents accuracy for real data. Section 5 concludes and discusses future work.

In Appendix A, we introduce the Baseline construction described in the Introduction and informally describe the required number of LSHs for fixed accuracy parameters. Appendix B gives theoretical parameter regimes, this analysis assumes LSHs have exactly $\epsilon_{\mathtt{f}}$ and $\epsilon_{\mathtt{t}}$ matching on random data, ignoring the variance in these parameters. Finally, Appendix C describes implementation results on random data.

2 Preliminaries

Throughout this work we use the following notation:

1. Let λ be a security parameter,
2. Let stored records x_i be values over $\{0,1\}^\gamma$,
3. Let $M = |\mathcal{DB}|$, the number of records,
4. We consider prime fields over prime power p, denoted $\mathbb{F}_p$.
5. For a Reed Solomon code, let k be the dimension, $\mathsf{k}_{correct}$ be the required number of correct symbols, and k_{error} be the maximum number of incorrect symbols. See Definition 2.
6. Let n denote the number of LSHs, and
7. If one uses an extended LSH, let α denote the number of LSHs that are concatenated.

We use $\vec{x} = (x_1, ..., x_\ell)$ to denote a vector. We focus on the Hamming distance. For vectors $x, y \in \{0,1\}^\gamma$, let $\mathcal{D}(x,y) = |\{i | x_i \neq y_i\}|$ denote the Hamming distance between x and y. The Hamming distance metric is frequently used in iris recognition [Dau09]. Our techniques apply to any metric with locality sensitive hashes [IM98]. For a positive integer x, let $[x]$ denote the set $\{1, ..., x\}$. `TAR` stands for True Accept Rate; in the same way `FAR` stands for False Accept Rate. For protocols Prot between a client Client and a server Server we use notation

$$\begin{pmatrix} o_{\mathsf{Client}} \\ o_{\mathsf{Server}} \end{pmatrix} \leftarrow \mathsf{Prot} \begin{pmatrix} i_{\mathsf{Client}} \\ i_{\mathsf{Server}} \end{pmatrix}$$

with $i_{\mathsf{Client}}, o_{\mathsf{Client}}, i_{\mathsf{Server}}, o_{\mathsf{Server}}$ denoting the client's and the server's inputs and outputs respectively. Protocols are written from the client's perspective.

Definition 1 (Locality-sensitive Hashing (LSH))**.** *Let $t \in \mathbb{N}$, $c > 1$ and $\epsilon_{\mathtt{t}}, \epsilon_{\mathtt{f}} \in [0,1]$ with $\epsilon_{\mathtt{t}} > \epsilon_{\mathtt{f}}$. $\mathcal{H}$ defines a $(t, ct, \epsilon_{\mathtt{t}}, \epsilon_{\mathtt{f}})$-sensitive hash family if for any $x, y \in \{0,1\}^\gamma$ one has:*

- *If $\mathcal{D}(x,y) \leq t$ then $\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] \geq \epsilon_{\mathtt{t}}$*

- *If* $\mathcal{D}(x,y) \geq ct$ *then* $\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] \leq \epsilon_{\mathtt{f}}$

where $\mathcal{D}(x,y)$ *denotes the Hamming distance between binary vectors* x *and* y*.*

An extended LSH is formed by concatenating α *independently sampled LSHs. This output is an LSH, with parameters* $\epsilon_{\mathtt{t}} = \epsilon_{\mathtt{t}}'^{\alpha}$ *and* $\epsilon_{\mathtt{f}} = \epsilon_{\mathtt{f}}'^{\alpha}$*. This is used to compute parameters.*

Definition 2 (Reed-Solomon Codes)**.** *The* (n,k) *Reed-Solomon code over* $\mathbb{F}_p$ *is the*

$$\mathsf{RS}_{(n,k)} := \{\mathtt{C} | \mathtt{C}_i = P(i) \text{ for some } k-1 \text{ degree } P\}$$

Such codes are linear with the Vandermonde matrix representing one encoding matrix. A (n,k) *Reed-Solomon code has the following features:*

1. *Let* $\mathtt{C}_1, ..., \mathtt{C}_n \in \mathsf{RS}_{n,k}$ *and let* $\mathtt{C}'_1, ..., \mathtt{C}'_n$ *be some value. Define*

$$\begin{aligned} \mathsf{k}_{correct} &= |\{i | \mathtt{C}_i = \mathtt{C}'_i\}|. \\ \mathsf{k}_{error} &= |\{i | \mathtt{C}_i \neq \mathtt{C}'_i \wedge \mathtt{C}'_i \neq \perp\}| \\ \mathsf{e}_{erase} &= |\{i | \mathtt{C}'_i = \perp\}|. \end{aligned}$$

   where $\mathsf{k}_{correct} + \mathsf{k}_{error} + \mathsf{e}_{erase} = n$*. There exists an efficient procedure* $\mathsf{Decode}(\mathtt{C}'_1, ..., \mathtt{C}'_n) = \mathtt{C}_1, ..., \mathtt{C}_n$ *when* $2\mathsf{k}_{error} + \mathsf{e}_{erase} < n - k$*. The Berlekamp-Welch algorithm is one such procedure. We assume that* $\mathsf{Decode}_{\mathsf{RS}}$ *outputs* $\perp$ *if* $\mathsf{e}_{erase} > n - k$*.*

2. *For any* $\ell \leq k$*,* $\mathtt{Agree} = (i_1, y_{i_1}, ..., i_\ell, y_{i_\ell})$ *there exists a nonempty set* $\mathsf{RS}_{\mathtt{Agree}}$ *such that for all* $\mathtt{C} \in \mathsf{RS}_{\mathtt{Agree}}$*,* $\mathtt{C}_{i_j} = y_{i_j}$ *for* $j = 1, ..., \ell$*. (This is because all* $k \times k$ *minors of the Vandermonde matrix are full rank.) Furthermore, one can efficiently sample from* $\mathsf{RS}_{\mathtt{Agree}}$ *using an algorithm denoted as* $\mathsf{Inv}_{\mathsf{RS}}$*.*

Definition 3. *A map* $\mathsf{Map} = (\mathsf{Map}.\mathit{insert}, \mathsf{Map}.\mathit{retrieve})$ *is a pair of algorithms where*

1. $\mathsf{Map}.\mathit{insert}(L, R)$*: Adds* (L, R) *where* L *is the key and* R *is its associated value.*

2. $\mathsf{Map}.\mathit{retrieve}(L)$*: Receives* L *and returns the last assigned value* R *or* $\perp$ *if no value has been assigned.*

We assume that values L *and* R *are both binary strings of a fixed length. Looking ahead, values* R *will be from a field* $\mathbb{F}_p$ *we assume that* $|\log p|$ *is at most the supported length of the map. In a multimap, denoted as* MM*,* $\mathsf{MM}.\mathit{retrieve}(L)$ *returns all previously assigned values* R_i*.*

2.1 Approximate Proximity Search

We now turn to defining our cryptographic goal. We begin with the notion of a well-spread database which captures the intuition that its records are far apart.

Definition 4 (Well-spread database)**.** *For parameters* $c > 1, t \in \mathbb{Z}^+$*. For some value* $y \in \{0,1\}^\gamma$ *and* $\mathcal{DB} \in \{0,1\}^{\gamma \times M}$*, define*

$$\begin{aligned} \mathtt{Close}(y, \mathcal{DB}) &= \{x_i \mid x_i \in \mathcal{DB} \ \&\ \mathcal{D}(x_i, y) \leq t\}, \\ \mathtt{Far}(y, \mathcal{DB}) &= \{x_i \mid x_i \in \mathcal{DB} \ \&\ \mathcal{D}(x_i, y) \geq ct\}. \end{aligned}$$

A database $\mathcal{DB} \in \{0,1\}^{\gamma \times M}$ *is said to be* (c,t)*-well-spread if*

$$\forall x \in \{0,1\}^\gamma, |\mathtt{Far}(x, \mathcal{DB})| \geq |\mathcal{DB} - 1|.$$

This implies that $\forall x \in \{0,1\}^\gamma, |\mathtt{Close}(x, \mathcal{DB})| \leq 1$*.*

As discussed in the Introduction, Definition 4 is a strong condition useful for analysis. It is not satisfied by biometric data, see histograms for the IITD dataset in Figure 3(a). However, accuracy measurements in Section 4 consider real data.

Definition 5 (Approximate Proximity Search Scheme)**.** *Let* $\mathsf{APSS} = (\mathsf{APSS.Init}, \mathsf{APSS.Setup}, \mathsf{APSS.Search})$*. For* $c > 1, t \in \mathbb{Z}^+$ APSS *is a* $(t, c, q, \delta_{\mathtt{Far}}, \delta_{\mathtt{Close}})$*-approximate proximity search scheme if for all* (c, t)*-well-spread* $\mathcal{DB} \in \{0,1\}^{\gamma M}, y_1, ..., y_q \in \{0,1\}^{\gamma}$*, define*

$$\begin{aligned}
\begin{pmatrix} \mathsf{sk}, \mathsf{pp} \\ \mathsf{pp} \end{pmatrix} &\leftarrow \mathsf{APSS.Init} \begin{pmatrix} 1^\lambda \\ 1^\lambda \end{pmatrix}, \\
\begin{pmatrix} \perp \\ \mathcal{I}_0 \end{pmatrix} &\leftarrow \mathsf{APSS.Setup} \begin{pmatrix} \mathcal{DB}, \mathsf{sk} \\ \mathsf{pp} \end{pmatrix}, \\
\begin{pmatrix} J_i \\ \mathcal{I}_i \end{pmatrix} &\leftarrow \mathsf{APSS.Search} \begin{pmatrix} y_i, \mathsf{sk} \\ \mathcal{I}_{i-1}, \mathsf{pp} \end{pmatrix}.
\end{aligned}$$

Then it is true that, $\forall i, 1 \leq i \leq q$*,*

$$\begin{aligned}
\Pr\left[\mathtt{Far}(y_i, \mathcal{DB}) \cap J_i = \emptyset\right] &\geq 1 - \delta_{\mathtt{Far}}, \\
\Pr\left[\mathtt{Close}(y_i, \mathcal{DB}) \subseteq J_i\right] &\geq 1 - \delta_{\mathtt{Close}}.
\end{aligned}$$

where `Far`, `Close` *are defined as in Definition 4.*

Definition 6 (Adaptive Security for Search Protocol)**.** *Let* $\mathsf{SSE} = (\mathsf{Init}, \mathsf{Setup}, \mathsf{Search})$ *be a triple of algorithms with associated leakage functions* $(\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathtt{Search}})$*. Let* λ *be a security parameter.*

For an adversary $\mathcal{A}$ *and simulator* $\mathcal{S}$ *define* $Exp_{\mathsf{SSE},\mathcal{A}}(\cdot)$ *and* $Exp_{\mathcal{S},\mathcal{A}}(\mathcal{L} = (\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathtt{Search}}))$ *as in Figure 1. We say* SSE *is semantically secure in the adaptive setting if for all PPT* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that*

$$|\Pr[Exp_{\mathsf{SSE},\mathcal{A}}(\cdot)) = 1] - \Pr[Exp_{\mathcal{S},\mathcal{A}}(\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathtt{Search}}) = 1]| \leq \mathtt{ngl}(\lambda).$$

We use Definition 6 for maps, multimaps, and approximate proximity schemes, which we denote at Map, MM and APSS respectively. We consider the following leakage functions:

1. $\mathcal{L}^{\mathtt{Setup}}_{\mathsf{Size}}$ which leaks the size of the created $\mathcal{DB}$. For the case of a map this leaks the number of (keyword, value) pairs inserted. Size is often padded to a power of 2, (e.g. [HCD+23]).
2. $\mathcal{L}^{\mathtt{Search}}_0$ which leaks the occurrence of a query [BIPW17, BT21], and
3. $\mathcal{L}^{\mathtt{Search}}_{\mathsf{AccPatt}}$ which leaks identifiers returned with a query [SWP00, CGKO06]. These identifiers are consistent across queries.
4. $\mathcal{L}^{\mathtt{Search}}_{\mathsf{QueryEq}}$ which leaks when queries repeat in a sequence [LZWaT14, OK21].

During our construction we make n calls to the underlying map, in the case of $\mathcal{L}^{\mathtt{Search}}_0$ this creates a straightforward leakage function as there are n calls to that map. Below we define two modifications of the above leakage functions. These leakage functions when one uses multiple LSHs in conjunction with a map. That is, they apply for the baseline or `ProxCode` system. They are a function of making multiple calls to the underlying map [KIK12, BT21, HCD+23]. For a query y and an integer n, we consider subqueries of the form $y_1, ..., y_n$.

1. $\mathcal{L}^{\mathtt{Search}}_{\mathsf{SubAccPatt}}$ for an integer n, for each returned identifier ι leaks the pair (i, ι) of each subquery y_i that caused the identifier ι to be returned where $1 \leq i \leq n$.
2. $\mathcal{L}^{\mathtt{Search}}_{\mathsf{SubQueryEq}}$ for an integer n, leaks query equality over subqueries.

Experiment $Exp_{\mathsf{SSE},\mathcal{A}}(\cdot)$:

1. $\begin{pmatrix}\mathsf{sk}\\ \mathsf{pp}\end{pmatrix} \leftarrow \mathsf{SSE.Init}\begin{pmatrix}1^\lambda\\ 1^\lambda\end{pmatrix}$.
   Let $\mathsf{ts}_{\mathsf{Init}}$ be the server's view.
2. $D \leftarrow \mathcal{A}(\mathsf{ts}_{\mathsf{Init}})$.
3. $\begin{pmatrix}\perp\\ \mathcal{I}_0\end{pmatrix} \leftarrow \mathsf{SSE.Setup}\begin{pmatrix}D, \mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$.
   Let ts_0 be the server's view.
4. For $i = 1$ to q:
   (a) $y_i \leftarrow \mathcal{A}(\mathsf{ts}_{i-1})$.
   (b) $\begin{pmatrix}J_i\\ \mathcal{I}_i\end{pmatrix} \leftarrow \mathsf{SSE.Search}\begin{pmatrix}y_i, \mathsf{sk}\\ \mathcal{I}_{i-1}, \mathsf{pp}\end{pmatrix}$. Let ts_i be the server's view.
5. Output $b \leftarrow \mathcal{A}(\mathsf{ts}_q)$.

Experiment $Exp_{\mathcal{S},\mathcal{A}}(\mathcal{L}^{\mathtt{Setup}}, \mathcal{L}^{\mathtt{Search}})$:

1. $\mathsf{ts}_{\mathsf{Init}} \leftarrow \mathcal{S}(1^\lambda)$.
2. $D \leftarrow \mathcal{A}(\mathsf{ts}_{\mathsf{Init}})$.
3. $\mathsf{ts}_0 \leftarrow \mathcal{S}(\mathcal{L}^{\mathtt{Setup}}(D)))$.
4. For $i = 1$ to q:
   (a) $y_i \leftarrow \mathcal{A}(tk_{i-1})$.
   (b) $\mathsf{ts}_i \leftarrow \mathcal{S}(\mathcal{L}^{\mathtt{Search}}(y_i))$.
5. Output $b \leftarrow \mathcal{A}(\mathsf{ts}_q)$.

Figure 1: Adaptive Experiments for search protocols and Adversary interacting with the Simulator in the ideal world using $\mathcal{L}$. The $\mathcal{A}$ and $\mathcal{S}$ keep state between stages but these is omitted for notational clarity.

3 ProxCode

This section formally introduces ProxCode, proves it is secure, and proves it is accurate under the well-spread condition (Definition 4). This condition is used for analysis but not assumed in our evaluation in Section 4.

Our construction combines LSHs and a secure map. Instead of associating LSH outputs with records, we associate LSH outputs with points on a random polynomial whose intercept is the record. One then collects multiple shares and decodes, our search only reveals a matching record x_i when there are *enough* LSH matches.

To handle LSH collisions, when $z = \mathsf{LSH}_\beta(x_i) = \mathsf{LSH}_\beta(x_j)$, we constrain the two polynomials to have the same value at position β. That is, that $\mathsf{c}_{i,\beta} = \mathsf{c}_{j,\beta}$. We can do this without impacting either $\mathsf{c}_{i,1}$ or $\mathsf{c}_{k,1}$ as long as this occurs in no more than $k-1$ positions for each codeword. We present this scheme formally in Algorithm 1 and Construction 1.

As mentioned in the Introduction, one can consider the goal to retrieve the indices, i, or the actual values, x_i (see discussion in Gui et al. [GPPW24, GPP23]). Usually in encrypted search, one focuses on building an index data structure with the actual records being obtained through a second oblivious structure. Our system works equally well in both settings assuming the map can hold entire records (as long as they are distinct) since our encoding technique does not increase the size of values inserted in the map (beyond the additional space to encode them in a field). A separate lookup of the value x_i from i often has leakage, so we associate x_i to prevent the second lookup.

Construction 1. *Let t be a distance and let $c > 1$ be a distance parameter. Let $\mathcal{DB} \in \{0,1\}^{\gamma \times M}$ be a (c,t)-well-spread database. Let $n \in \mathbb{Z}^+$ and $\mu, k \in \mathbb{Z}^+$ such that $\mu \le k < n$.*

1. *Let LSH be a family of $(t, ct, \epsilon_{\mathtt{t}}, \epsilon_{\mathtt{f}})$-LSHs with domain of $\{0,1\}^r$.*
2. *Let p be a prime power such that $p \ge M$. Let $\mathsf{RS}_{(n+1,k)}$ the Reed-Solomon code with associated algorithms $\mathsf{Decode}_{\mathsf{RS}}, \mathsf{Inv}_{\mathsf{RS}}$.*
3. *Let $\mathsf{Map} = (\mathsf{Map}.\mathit{insert}, \mathsf{Map}.\mathit{retrieve})$ be a map.*

Define APSS as in Algorithm 1.

Algorithm 1 `ProxCode`: APSS from maps and linear (secret-sharing) codes.
Procedures are run by Client unless calling an underlying interactive protocol.

$\texttt{Init}\begin{pmatrix}1^\lambda\\1^\lambda\end{pmatrix} = \mathsf{Map.Init}\begin{pmatrix}1^\lambda\\1^\lambda\end{pmatrix}$

$\begin{pmatrix}\mathsf{LSH}_1,...,\mathsf{LSH}_n\\ \mathcal{I}\end{pmatrix} \leftarrow \texttt{Setup}\begin{pmatrix}\mathcal{DB},\mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$:

1. Sample $\mathsf{LSH}_1,...,\mathsf{LSH}_n \leftarrow \mathsf{LSH}$. Define $L \in (\{0,1\}^r)^{M\times n}$ where $L_{i,j} = \mathsf{LSH}_j(x_i)$.
2. Define $\texttt{Eq} \in [M]^{M\times n}$ where $\texttt{Eq}_{i,j} = \arg\min_{i'<i}(L_{i',j} = L_{i,j})$ where $\texttt{Eq}_{i,j} = 0$ if no such i' exists.
3. If there exists a row of Eq with more than $k-1$ nonzero coordinates go to Step 1 (up to ℓ times, then output $\perp$).
4. Initialize $\mathbf{C} \in (\mathbb{F}_p \cup \perp)^{M\times(n+1)} = \perp^{M\times(n+1)}$
5. For $i = 1,...,M$:
   (a) $\mathbf{C}_{i,1} = x_i$.
   (b) For $j = 1,...,n$, let $i' = \texttt{Eq}_{i,j}$ if $i' \neq 0$, set $\mathbf{C}_{i,j+1} = \mathbf{C}_{i',j+1}$.
   (c) Set $\mathbf{C}_i = \mathsf{Inv}_{\mathsf{RS}}(\texttt{NEmpty}(\mathbf{C}_i))$. `NEmpty` outputs the indices and values of positions that are not $\perp$.
6. $\mathcal{DB} = (j||\mathsf{LSH}_j(x_i), \mathbf{C}_{i,j+1})$ for $i = 1,...,M, j = 1,...,n$.
7. $\begin{pmatrix}\perp\\ \mathcal{I}_0\end{pmatrix} \leftarrow \mathsf{Map.Setup}\begin{pmatrix}\mathcal{DB},\mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$.

$\texttt{Search}\begin{pmatrix}y,\mathsf{LSH}_1,...,\mathsf{LSH}_n,\mathsf{sk}\\ \mathcal{I}_{(i-1)\cdot n},\mathsf{pp}\end{pmatrix}$:

1. Compute $L_j = \mathsf{LSH}_j(y)$ for all $j = 1,...,n$.
2. Initialize $\mathsf{e}_{erase} = n$.
3. For $j = 1,..,n$,
   (a) Client retrieves $\begin{pmatrix}\mathsf{c}_{j+1}\\ \mathcal{I}_{(i-1)\cdot n+j}\end{pmatrix} = \mathsf{Map.Search}\begin{pmatrix}(j,L_j),\mathsf{sk}\\ \mathcal{I}_{(i-1)\cdot n+(j-1)},\mathsf{pp}\end{pmatrix}$.
   (b) If $\mathsf{c}_{j+1} \neq \perp$, $\mathsf{e}_{erase} := \mathsf{e}_{erase} - 1$.
4. If $\mathsf{e}_{erase} > n-k$ output $\perp$.
5. Compute $\mathsf{c}_1,...,\mathsf{c}_k \leftarrow \mathsf{Decode}_{\mathsf{RS}}(\perp||\mathsf{c}_2||...||\mathsf{c}_n)$, output c_1

We provide some intuition for the scheme before presenting our formal results. There are two main ideas in Construction 1:

Polynomial evaluations in the Map. First, we replace x_i as the value inserted into the map with a polynomial whose intercept is x_i. That is, $\mathsf{c}_{i,1} = x_i$. The idea behind this change is that if one can reconstruct c_i then one can easily recover the value x_i. We then insert pairs $((j,\mathsf{LSH}_j(x_i)),\mathsf{c}_{i,j})$ into the map.

Align codewords with LSH collisions. We add a preprocessing step so that codewords are chosen in a correlated maner. We precompute using Eq the set of all LSH collisions in the database. If two values x_i, x_k share some $\mathsf{LSH}_\beta(x_i) = \mathsf{LSH}_\beta(x_j)$ then we will fix $\mathsf{c}_{i,\beta} = \mathsf{c}_{j,\beta}$. We rely on fact that one can interpolate a degree $k-1$ polynomial for any k points to ensure this is possible. Theorem 1 bounds the

probability that such sampling cannot complete over the choice of $\mathsf{LSH}_1, ..., \mathsf{LSH}_n$. Importantly, this probability holds for every well-spread $\mathcal{DB}$ and does not depend on the chosen codewords. We check this condition by examining $\mathtt{Eq}$.

Once Setup completes there is now a one-to-one correspondence between LSH outputs and codeword symbols. Let $x_i \in \mathcal{DB}$, if one searched for the value x_i one would retrieve $\mathsf{c}_{i,2}, ..., \mathsf{c}_{i,n+1}$ which would determine $\mathsf{c}_{i,1}$ and allow retrieval. If one searches for a value y then the returned values will be a mix of different codewords and $\perp$ where nothing in the database matched the LSH value. We first consider the correctness of this scheme deferring security until Section 3.2.

3.1 Correctness

Theorem 1. *Let $c, c_1, c_2 > 0$ be constants. Let LSH be a family of $(t, ct, \epsilon_{\mathtt{t}}, \epsilon_{\mathtt{f}})$-locality sensitive hashes. Let $\mathcal{DB}$ be a (c, t)-well-spread database where $|\mathcal{DB}| = M$. Let $n \in \mathbb{Z}^+, k \in \mathbb{Z}^+$ be parameters. Suppose the following are true:*

$$\epsilon_{\mathtt{t}} > \frac{2k}{(1-c_2)n}, \tag{1}$$

$$\epsilon_{\mathtt{f}} < \frac{k}{Mn(1+c_1)}, \tag{2}$$

and define

$$\forall i \leq M, \delta_{\mathtt{Far}_i} = exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon_{\mathtt{f}} \cdot n \cdot (i-1)\right),$$

$$\delta_{\mathtt{Far}} = \delta_{\mathtt{Far}_M} = exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon_{\mathtt{f}} \cdot n \cdot (M-1)\right),$$

$$\delta_{\mathtt{Close}} = exp\left(\frac{-c_2^2 \epsilon_{\mathtt{t}} n}{2}\right) + \delta_{\mathtt{Far}_{M-1}}.$$

Construction 1 instantiated with $\mathsf{RS}_{(n+1,k)}$ and n LSHs from LSH is an $(t, c, \delta_{\mathtt{Far}}, \delta_{\mathtt{Close}})-\mathsf{APSS}$. Furthermore,

$$\begin{aligned}\Pr[\mathsf{Setup}\ \textit{outputs}\ \perp] &\leq \left(1 - \prod_{i=2}^{M}(1-\delta_{\mathtt{Far},i})\right)^{\ell} \\ &\leq \left(1-(1-\delta_{\mathtt{Far}})^{M-1}\right)^{\ell}\end{aligned}$$

Theorem 1 is proved through Lemmas 1, 2, and 3 which focus on the number of LSH matches between close records, far records, and the ability of setup to complete. Roughly, each of these lemmas is proved using a Chernoff bound since LSH outputs are independent (if data is fixed before sampling). The constants c_1, c_2 represent the constant of the Binomial deviating from its expectation.

Lemma 1. *Let all parameters be as in Theorem 1. Define*

$$\mathsf{Match}_{j,x,x^*} = \begin{cases} 1 & LSH_j(x) = LSH_j(x^*) \\ 0 & otherwise \end{cases}.$$

And define $\mathsf{Match}_{x,x^*} = \sum_{j=1}^{n} \mathsf{Match}_{j,x,x^*}$. If $\mathcal{D}(x, x^*) \leq t$ then

$$\Pr[\mathsf{Match}_{x,x^*} \leq 2k] < exp\left(\frac{-c_2^2}{2} \cdot \epsilon_{\mathtt{t}} \cdot n\right).$$

Proof. Let x, x^* be two values where $\mathcal{D}(x, x^*) \leq t$. One has

$$\forall j, \mathsf{Exp}[\mathsf{Match}_{j,x,x^*}] > \frac{2k}{(1-c_2)n}$$

by Equation 1. By independence of the LSHs, Match_{x,x^*} is bounded below by a $(n, \frac{2k}{(1-c_2)n})$ binomial distribution with $\mathsf{Exp}[\mathsf{Match}_{x,x^*}] = \frac{2k}{(1-c_2)}$. Then by a standard Chernoff bound, it is true that

$$\begin{aligned}
&\Pr[\mathsf{Match}_{x,x^*} \leq 2k] \\
&\quad = \Pr[\mathsf{Match}_{x,x^*} < (1-c_2)\mathsf{Exp}[\mathsf{Match}_{x,x^*}]] \\
&\quad < exp\left(\frac{-c_2^2}{2} \cdot \epsilon_{\mathtt{t}} \cdot n\right).
\end{aligned}$$

This completes the proof of Lemma 1. □

Lemma 2. *Let all parameters be as in Theorem 1. Define random variable $\mathsf{Match}_{j,\mathcal{DB},x}$ as follows for $j \in \{1, ..., n\}$:*

$$\mathsf{Match}_{j,\mathcal{DB},x} = |\{x_i \in \mathcal{DB} | \mathsf{LSH}_j(x_i) = \mathsf{LSH}_j(x)\}|.$$

and $\mathsf{Match}_{\mathcal{DB},x} = \sum_{j=1}^{n} \mathsf{Match}_{j,\mathcal{DB},x}$, denoting the number of LSH's where there exists some collision between the value x' and some record in the $\mathcal{DB}$. For all x such that $\forall x_i \in \mathcal{DB}$ it is true that $\mathcal{D}(x, x_i) \geq ct$ it is true that

$$\Pr[\mathsf{Match}_{\mathcal{DB},x} \geq k] \leq exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon_{\mathtt{f}} \cdot n \cdot M\right).$$

Proof. For each pair x, x' such that $\mathcal{D}(x, x') \geq ct$ it is true that

$$\Pr_{\mathsf{LSH} \leftarrow \mathrm{H}_{\mathtt{lsh}}}[\mathsf{LSH}(x) = \mathsf{LSH}(x')] \leq \epsilon_{\mathtt{f}}.$$

This means that $\mathsf{Match}_{\mathcal{DB},x}$ is bounded above by a $(nM, \epsilon_{\mathtt{f}})$ binomial distribution. By a standard Chernoff bound one has

$$\begin{aligned}
&\Pr[\mathsf{Match}_{\mathcal{DB},x} > k] = \\
&\Pr[\mathsf{Match}_{\mathcal{DB},x} > (1+c_1)\mathsf{Exp}[\mathsf{Match}_{x,\mathcal{DB}}]] \leq \\
&\quad exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon_{\mathtt{f}} \cdot n \cdot M\right)
\end{aligned}$$

This completes the proof of Lemma 2. □

Lemma 3. *Let all parameters be as in Lemma 2 and Theorem 1 letting*

$$\delta_{\mathtt{Far},i} = exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon_{\mathtt{f}} \cdot n \cdot (i-1)\right).$$

Then the probability that Setup outputs $\perp$ is at most

$$\Pr[\mathsf{Setup}\ outputs\ \perp] \leq \left(1 - \prod_{i=2}^{M}(1 - \delta_{\mathtt{Far},i})\right)^{\ell} \tag{3}$$

Proof. Let $(x_1, ..., x_M) = \mathcal{DB}$ For all $x_i \in \mathcal{DB}$ define $\mathcal{DB}_{x_i} = x_1, ..., x_{i-1}$. By the (c,t)-well-spread condition of $\mathcal{DB}$ and Lemma 2 it is true that

$$\Pr\left[\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} \geq k\right] \leq \delta_{\mathtt{Far},i}.$$

Setup succeeds in an iteration if it is true for all $x_i \in \mathcal{DB}$ that $\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} < k$. Let 1_{x_i} be an indicator random variable where $1_{x_i} = 1$ if $\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} < k$. Then

$$\Pr\left[\sum_{i=2}^{M} 1_{x_i} = 0\right] \geq \prod_{i=2}^{M}(1 - \Pr[\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} \geq k]).$$

So the chance that an iteration of setup fails is at most

$$\Pr\left[\sum_{x_i} 1_{x_i} > 0\right] \leq 1 - \prod_{i=2}^{M}(1 - \Pr[\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} \geq k]).$$

The chance that all ℓ iterations fail is then at most

$$\left(1 - \prod_{i=2}^{M}(1 - \Pr[\mathsf{Match}_{\mathcal{DB}_{x_i},x_i} \geq k])\right)^{\ell}.$$

This completes the proof of Lemma 3. □

Proof of Theorem 1. There are three parts to proving Theorem 1 that setup completes with high probability, that the close item is included in the result set and that far items are not included in the result set. The probability of setup completing follows directly from Lemma 3.

Close Item in Result Set. Let x be the search term where x is close to at most one item in $\mathcal{DB}$ denoted as x_i with corresponding codeword c_i. That is, $\mathcal{D}(x, x_i) \leq t$. If such a x_i exists, its uniqueness is guaranteed by Definition 4. Let c_i denote the corresponding codeword. Define the following parameters:

$$\delta_{\mathtt{Close},1} = exp\left(\frac{-c_2^2}{2} \cdot \epsilon_{\mathtt{t}} \cdot n\right),$$
$$\delta_{\mathtt{Close},2} = \delta_{\mathtt{Far},(M-1)}.$$

Let c' denote the recovered symbols (including symbols that are $\perp$). By Lemma 1 there are at least $2k$ symbols from c_i with probability $1 - \delta_{\mathtt{Close},1}$. By Lemma 2 there are at most k_{error} symbols from the other LSH values $M - 1$. By union bound, both of these conditions hold with probability $1 - (\delta_{\mathtt{Close},1} + \delta_{\mathtt{Close},2})$ Conditioned on both of these events occurring $\mathsf{Decode}_{\mathsf{RS}}$ outputs c_i with probability 1.

Far Items not in Result Set. By Lemma 2 the probability that c' has more than $k - 1$ symbols other than $\perp$ is at most $\delta_{\mathtt{Far}}$.
This completes the proof of Theorem 1. □

3.2 Security and Leakage

This section shows that when the Map in Construction 1 is an appropriate encrypted map one achieves a secure APSS. The proofs in this section are straightforward. We consider two leakage patterns frequently used in secure maps: 1) the zero-leakage setting where the server learns the dataset size M and when a query occurs such as [BIPW17, BT21] and 2) access and search pattern where the server learns identifiers associated with each query response and whether queries have repeated [SWP00, CGKO06]. Of course, if one uses a zero-leakage map [WNL+14, BT21], the resulting APSS is zero-leakage as well (treating n as a public system parameter). Since each query of the APSS translates to n queries to the underlying map, we additionally leak when subqueries repeat and the subquery associated with a returned identifier.

Lemma 4. *Let λ be a security parameter. Let $\mathsf{Map} = (\mathsf{Map}.\mathit{Setup}, \mathsf{Map}.\mathit{Search})$ be a map that is secure according to Definition 6 for $\mathcal{L}_{\mathsf{Map}} = (\mathcal{L}^{\mathtt{Setup}} = |\mathsf{Map}|, \mathcal{L}^{\mathtt{Search}} = (\mathcal{L}^{\mathtt{Search}}_{\mathit{AccPatt}}, \mathcal{L}^{\mathtt{Search}}_{\mathit{QueryEq}}))$. Then the $\mathit{APSS} = (\mathit{APSS.Init}, \mathit{APSS.Setup}, \mathit{APSS.Search})$ scheme defined in Construction 1 is secure according to Definition 6 for $(\mathcal{L}^{\mathtt{Setup}} = n \cdot |\mathsf{Map}|, \mathcal{L}^{\mathtt{Search}} = (\mathcal{L}^{\mathtt{Search}}_{\mathit{SubAccPatt}}, \mathcal{L}^{\mathtt{Search}}_{\mathit{SubQueryEq}}))$.*

Proof. Let $\mathcal{A}_{\mathsf{APSS}}$ denote some PPT adversary for the APSS scheme. Our goal is to construct a $\mathcal{S}_{\mathsf{APSS}}$. As noted in Figure 2 for any valid $\mathcal{A}_{\mathsf{APSS}}$ adversary there exists some $\mathcal{A}_{\mathsf{Map}}$ that is a valid Map adversary. Let $\mathcal{S}_{\mathsf{Map}}$ be one such simulator for $\mathcal{A}_{\mathsf{Map}}$. Note that setup leakage is the same in both settings. For the search leakage, $(\mathcal{L}^{\mathtt{Search}}_{\mathsf{SubAccPatt}}, \mathcal{L}^{\mathtt{Search}}_{\mathsf{SubQueryEq}})$ this allows $\mathcal{S}_{\mathsf{APSS}}$ to expand the q queries into qn subqueries which is the required leakage for $\mathcal{S}_{\mathsf{Map}}$. Then

$\mathcal{A}_{\mathsf{Map}}.\mathsf{Setup}(1^\lambda)$:

1. Initialize $\mathcal{A}^{\mathtt{APSS}}$ and receive $\mathcal{DB} \in (\{0,1\}^\gamma)^M$.
2. Run steps 1-6 of $\mathsf{APSS.Setup}(\mathcal{DB})$ from algorithm 1 to receive vector $\vec{\mathsf{LSH}}$ and matrices L and $\mathtt{C}$ as described in steps 1, 2 and 6 respectively. If Step 4 outputs $\perp$ output $\perp$.
3. Output $\mathcal{DB}_{\mathsf{Map}} = \{(L_{i,j}, \mathtt{C}_{i,j+1})\}_{j=1,...,n}^{i=1,...,M}$

$\mathcal{A}_{\mathsf{Map}}.\mathsf{Search}(\vec{\mathsf{LSH}}, L, \mathtt{C})$:

1. Receive $q \in \{0,1\}^\gamma$ from $\mathcal{A}^{\mathtt{APSS}}$.
2. Compute $q_1, ..., q_n = \vec{\mathsf{LSH}}_1(q), ..., \vec{\mathsf{LSH}}_n(q)$.
3. Output $q_1, ..., q_n$.
4. Receive tk and send to $\mathcal{A}^{\mathtt{APSS}}$.

Figure 2: Construction of $\mathcal{A}_{\mathsf{Map}}$ from $\mathcal{A}_{\mathsf{APSS}}$.

$$|\Pr[Exp_{\mathsf{APSS},\mathcal{A}_{\mathsf{APSS}}}(\cdot)) = 1] - \Pr[Exp_{\mathcal{S}_{\mathsf{APSS}},\mathcal{A}_{\mathsf{APSS}}}(\mathcal{L}_0^{\mathtt{init}}, \mathcal{L}^{\mathtt{Search}}_{\mathsf{SubQueryEq},\mathsf{SubAccPatt}}]) = 1]| =$$
$$|\Pr[Exp_{\mathsf{Map},\mathcal{A}_{\mathsf{Map}}}(\cdot)) = 1] - \Pr[Exp_{\mathcal{S}_{\mathsf{Map}},\mathcal{A}_{\mathsf{Map}}}(\mathcal{L}_0^{\mathtt{init}}, \mathcal{L}^{\mathtt{Search}}_{\mathsf{QueryEq},\mathsf{AccPatt}}) = 1]|$$

This completes the proof of Lemma 4. □

3.3 Discussion

Handling Dynamic Data. Assuming a dynamic map, one can naturally handle new data x^* being added to the database by searching for x^* and retrieving codeword symbols that x^*'s codeword should be consistent with. Then one can sample the codeword (under the constraints described above) and add the missing codeword symbols to the corresponding maps. Handling data deletion and updates requires care; maps values have information about multiple biometrics. One way to handle deletes is to maintain a counter with each value indicating how many records are using this value, this counter could be decremented with each delete. The leakage and efficiency of the above depends strongly on the underlying map, and further study is required to understand viability.

4 Implementation and Accuracy

We implemented `ProxCode` using Python 3.9. Our implementation uses the tree-based oblivious map of Wang et al. [WNL+14]. The source code can be found in (Github). We perform tests on iris data to indicate viability and on random data to scale. Iris datasets are not available for large M. Figure 3 shows histograms for the two datasets. There are two main differences between random and real data:

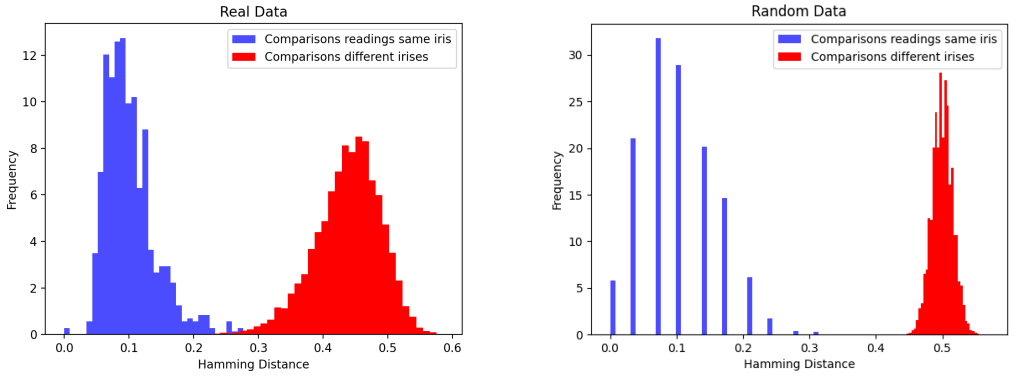
1. In real data there is a large variance on the Hamming distance both between readings of the same iris and readings of different irises.
2. In real data there is an overlap between distance comparisons of readings of the same iris and readings of different irises. This means it is not possible to achieve a system with perfect accuracy.

4.1 Real Data

The IITD dataset [KP10] consists of 224 persons and 2240 images. We process this data using the feature extractor ThirdEye [AF19a] and segmentation system of Ahmad and Fuller [AF19b]. We use their regime of left irises for training and right irises for testing. After removing the right irises without two readings, there are $M = 208$ right irises suitable for testing. The only statistics we use to set parameters from the combination of the dataset and feature extractor are the means and variances of distances between readings of the same iris and readings of different irises.

$M = 10^4$													
$\delta_{\texttt{Far}} = 10^{-4}$	Baseline		ProxCode				$\delta_{\texttt{Far}} = 10^{-6}$	Baseline		ProxCode			
$\epsilon'_{\texttt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	$\epsilon'_{\texttt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$
.7	61	10.3	36	7.5	27	6×10^{-4}	.7	69	11.5	37	7.6	25	10^{-3}
.75	51	7.3	30	5.7	28	4×10^{-4}	.75	58	8.2	31	5.8	28	4×10^{-4}
.8	44	5.2	26	4.5	30	2×10^{-4}	.8	50	5.7	26	4.3	20	4×10^{-3}
.85	39	3.7	**23**	**3.6**	**29**	$\mathbf{3 \times 10^{-4}}$	.85	45	4.3	**24**	**3.7**	**33**	$\mathbf{10^{-4}}$
.9	36	2.8	**21**	**3.0**	**34**	$\mathbf{10^{-4}}$	.9	40	2.8	**21**	**2.8**	**22**	$\mathbf{2 \times 10^{-3}}$

Table 3: Parameters Comparison between our ProxCode and Baseline scheme where $M \cdot \epsilon_{\texttt{f}}'^{\alpha}.n = \delta_{\texttt{Far}}$ and $(1-\epsilon_{\texttt{t}}'^{\alpha})^n = \delta_{\texttt{Close}}$. In all rows $\epsilon'_{\texttt{f}} = .5$. In ProxCode parameters are computed as in Appendix B. The numbers for α, n, k are the first found solutions. For the baseline scheme we measure FAR while in ProxCode we measure $\delta_{\texttt{Far}}$ this allows the baseline scheme to have more errors for the same accuracy. Accuracy $\delta_{\texttt{Far}} \approx 10^{-3}$ from $c_1 = 2$ and $c_2 = .4$. Accuracy $\delta_{\texttt{Far}} \approx 10^{-4}$ from $c_1 = 3, c_2 = .4$ and accuracy $\delta_{\texttt{Far}} \approx 10^{-6}$ from $c_1 = 5, c_2 = .4$. Logarithms are base 10. Appendix B describes methodology and presents more parameter ranges in Table 6. Bold numbers are the parameters that were used in testing the implementation.



(a) IITD dataset

(b) Random data with $M = 1000$ and mean same error rate of .10

Figure 3: FHD histogram for real data and random data loss. Comparisons between readings of the same biometric are in blue. Comparisons between readings of different biometrics are in red. The x-axis differs. Discontinuities in the same histogram for random data are due to using a Binomial distribution to generate errors.

We used the same methodology that we used to compute Table 3, to find the right parameters for the dataset of size $M = 208$. Table 3 parameters are computed assuming a well-spread database. Figure 3 shows that biometrics' distribution satisfy this condition except for the tails. Since the variance in real iris datasets in higher than random dataset, Setup with Table 3 parameters did not succeed.

Thus, we then manually tuned the parameters in a way that Setup succeeds with reasonable TAR during Search. We considered $\alpha \in [15, 30]$, $k \in [1, 30]$, and $n = [1000, 3000]$. For most of these trials, Setup could not sample a good set of LSHs. Table 4 list parameters that both Setup succeeded, and Search had reasonable TAR. For many parameter regimes, Setup succeeded with a TAR of 0 (when one picked a large α, k and small n). The most promising parameters are for $\alpha = 23$, $n = 1000$, and $k = 30$ with a TAR of .87 a single iteration for Setup to succeed.

Results are shown in Table 4. In addition, Table 4 shows the number of times subquery equality was observed over the 208 iris query set. This number is usually about the number of LSHs. For the first row in Table 4, the matrix of subquery equality has 416000 entries with .5% of them being nonzero.

α	n	k	Iterations	TAR	$\delta_{\texttt{Far}}$	Subquery Equality
25	2000	30	1	.89	0	2336
30	2000	15	1	.89	.004	2074
25	1500	25	2	.88	0	1775
23	1000	25	1	.88	.004	1355
23	1000	30	1	.87	0	1338
25	1500	30	1	.87	.004	1762
25	1000	25	1	.85	.004	1192
25	1000	30	1	.82	.004	1176
30	1000	30	1	.67	.004	1043

Table 4: Correctness on IITD dataset [KP10] processed with ThirdEye feature extractor [AF19a]. Iterations columns reports on the number of iterations needed to complete Setup. Subquery Equality reports the total number of subquery matches.

4.2 Time Efficiency

Our savings in n will translate to any encrypted map, see discussion in the Introduction. For our cryptographic implementation, we use a tree-based oblivious map due to Wang et al. [WNL+14]. At a high level, the construction is a tree of oblivious arrays where one stores the current logical position of all children for a node in addition to the node value. One can traverse the tree using PathORAM [SDS+18] using $\text{depth} + 1$ rounds of communication. The depth of the tree is the $1 + \log$ of the number of items in the map.

Usually one queries the map on all n LSH values of the search term. Ha et al. [HCD+23] build a preprocessing step using private set intersection where one first finds out which terms exist in the map and then uses the map to search only a constant number of non-null terms. They find the cryptographic overhead of this first stage is small compared to overhead of the second stage.

We use κ to denote the required number of parallel accesses to ORAM that doesn't impact accuracy. Let $\max_{\texttt{error}}$ be the maximum number of errors observed in the query set. Then $\kappa = \max\{2k + \max_{\texttt{error}}, 3\max_{\texttt{error}}\}$. We exclude queries where the number of errors is larger than $2k$ as these would not be corrected. By making κ queries in this way, one always retains enough codeword symbols to reconstruct the value without having to query all n terms. Often $\kappa/n \approx .1$ improving performance by an order of magnitude. Furthermore, κ is usually of the scale that queries could be processed in parallel by a modern server.

ProxCode preprocessing time. We first measure time to run Algorithm 1 ignoring the map, note this algorithm has complexity $\Theta(n \times M)$, assuming constant cost to evaluate each LSH. Timings are on a commodity laptop with 2.6 GHz 6-Core Intel Core i7 CPU, and 32 GB 2667 MHz DDR4 RAM. We timed setup for Random datasets of size 10^4, and Real dataset of size 208. For the real dataset where we set $\alpha = 23$, and $n = 1000$, the setup time is 16 seconds. The highest measured setup time for real data, with higher α and n, was 20 seconds. For the Random dataset, we tested on with $M = 10,000$ and $n = 631, 1000, 3982, 5012$ Setup completed in $237, 434, 7415$ and 9939 seconds respectively.

Cryptographic Efficiency. We then benchmark using the full system including the oblivious map implementation. These evaluations are on a Ryzen Threadripper PRO 7995WX with 384MB L3 cache and 768GB of DDR5 RAM. Here we report the time to set up all the oblivious RAM arrays, the number of rounds, the number of ORAM reads, the overall search time, and the parallel search time which is the average (across queries) of the maximum time required for each subquery. Results are in Table 5.

4.3 Implications of ProxCode for Client Security

Throughout this work, we defined and considered the two party SSE setting for notational simplicity. Since ProxCode is primarily a preprocessing technique it naturally extends to a three party SSE setting. A three party SSE consider a data owner, server, and a client [FVK+15]. In this setting, in addition to limiting leakage to the server, the data owner wishes to limit unintended data learned by a client. The three party setting highlights the importance of an accurate system. To demonstrate the difference between the baseline

ℓ	Dataset type	# Queries	n	k	α	Map K Size	Par Rounds	Setup Time	Setup GB	ORAM Reads	Search Time Seq.	Search Time Par.	κ
208	IITD	208	1000	25	23	208	20	200	.3	17	6.7	.0027	99
208	IITD	208	2000	30	25	416	21	424	.6	38	24.3	.0034	135
1000	random	50	398	21	17	398	21	406	.6	7.6	2.0	.0029	56
1000	random	50	631	32	17	631	22	685	1.2	12.6	4.1	.0034	75
1000	random	50	1258	21	18	1258	23	1462	2.3	26.4	11.5	.0038	55
10000	random	50	631	30	22	6310	25	8123	9.2	14.5	4.0	.0045	67
50000	random	50	1119	21	24	55950	28	83487	72.6	29.1	10.5	.0056	47

Table 5: Cryptographic Efficiency results. Map sizes and number of Reads (number of ORAM accesses per search) are in thousands. Times are in seconds.

and `ProxCode` we demonstrate the difference in information available between map outputs to the client. This information is available even if one uses a fully oblivious map.

- **Baseline:** For fixed value $n = 2000$ we found the α for Baseline with comparable TAR. This value is $\alpha = 44$, producing a TAR of .923. For these values across the query set, this produced 3 false positives across the 208 queries. Each false positive represents a biometric of a non-relevant person incidentally exposed to the client.
- **ProxCode:** For parameters $\alpha = 25, n = 2000, k = 30$ (first row in Table 4), we fixed a single value x_i and issued queries corresponding to the other irises in the dataset. We then measured how many values $\mathbf{C}_{i,j}$ for $2 \leq \mathbf{C}_{i,j} \leq n + 1$ are returned by some other query. That is, we measure how many "shares" of $\mathbf{C}_i$ are obtained after 207 queries for each of the other irises. This produced the average of 2.8 shares across 207 queries, with variance = 13.2, and the maximum of 31 shares. Only a single iris had at least k codeword symbols returned after issuing 207 queries. Obtaining k symbols is necessary for decoding (ignoring the "errors" received by the client).

To summarize, in the Baseline system a client who issues a small number of queries has a small probability of learning some non-relevant iris. In `ProxCode` a client may be able to decode a single iris after 200 queries. Both of these analysis assume client queries come from irises in the dataset. In both settings, one expects higher success with specifically crafted queries [ZKP16, ZWX+23].

5 Conclusion and Future Work

In this work, we consider approximate proximity searchable encryption in both zero and access pattern leakage setting. Our scheme allows use of a map and reduces leakage over the baseline scheme.

This work consider Reed-Solomon codes which correct arbitrary errors. However, observed errors are not arbitrary. Assume that the $\epsilon_{\mathtt{t}}$ is tuned so that $k' > k$ LSH matches occur for a nearby value with good probability. The actual errors are defined by the following process:

1. Sample M codewords $\mathtt{c}_1, ..., \mathtt{c}_M$ (under the collision constraint defined above).
2. Consider codeword $\mathtt{c}_i$ corresponding to a search for a value x^* that is close to x_i. Consider a fixed a symbol j. With probability at least $1 - \epsilon'_{\mathtt{t}}$ the symbol $\mathtt{c}_{i,j}$ is correctly transmitted. Otherwise there are two cases:
   (a) With probability at most $(M-1)\epsilon'_{\mathtt{f}}$ is replaced by $\mathtt{c}_{i,k}$ for some $k \neq j$. Each of these replacements occur with probability at most $\epsilon'_{\mathtt{f}}$.
   (b) Otherwise, the symbol is converted to $\perp$.

There are two important aspects of the above error model: 1) that errors come from the symbols of other codewords and 2) that these codewords are not independently sampled.

It is an open problem to design codes that correct more of such errors than is possible in traditional error models. Furthermore, it seems possible to argue some independence and randomness of the errors (Shannon

model [GRS22]) using the secret sharing properties of the code. We were not able to prove this or find a counterexample. The sticking point was the coupled sampling of the codewords.

`ProxCode` has a dramatic impact on the efficiency of LSH based proximity searchable encryption. In natural parameter settings it improves the size of the index structure by a factor of 30 leading to major improvements in all aspects of system efficiency. We present an open-source prototype implementation that confirms all presented analysis for random data. `ProxCode` also demonstrates high accuracy on real iris data after careful parameter tuning.

Acknowledgements

The authors are grateful to the reviewers for their important comments in improving this work. The work of B.F. and M.R. is supported by NSF grants #2141033 and #2232813. This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via Contract No. 2019-19020700008. This material is based upon work supported by the Defense Advanced Research Projects Agency, DARPA, under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

[ACD+22] Sohaib Ahmad, Chloe Cachet, Luke Demarest, Benjamin Fuller, and Ariel Hamlin. Proximity searchable encryption for the iris biometric. In AsiaCCS, 2022.

[AF19a] Sohaib Ahmad and Benjamin Fuller. Thirdeye: Triplet-based iris recognition without normalization. In IEEE BTAS, 2019.

[AF19b] Sohaib Ahmad and Benjamin Fuller. Unconstrained iris segmentation using convolutional neural networks. In ACCV 2018 Workshops, pages 450–466. Springer, 2019.

[AG22] Megumi Ando and Marilyn George. On the cost of suppressing volume for encrypted multi-maps. Proceedings on Privacy Enhancing Technologies, 4:44–65, 2022.

[APP+23] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. PoPETS, 1:417–436, 2023.

[BBOH96] Christopher M Brislawn, Jonathan N Bradley, Remigius J Onyshczak, and Tom Hopper. The FBI compression standard for digitized fingerprint images. In Proc. SPIE, volume 2847, pages 344–355, 1996.

[BHJP14] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR), 47(2):1–51, 2014.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In TCC (2), pages 662–693. Springer, 2017.

[BT21] Alexandra Boldyreva and Tianxin Tang. Privacy-preserving approximate k-nearest-neighbors search that hides access, query and volume patterns. PoPETS, 2021.

[CCD+20] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M Sadegh Riazi. SANNS: Scaling up secure approximate k-Nearest neighbors search. In USENIX Security, pages 2111–2128, 2020.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In CCS, pages 79–88, 2006.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In CCS, pages 668–679, 2015.

[Dau09] John Daugman. How iris recognition works. In The essential guide to image processing, pages 715–739. Elsevier, 2009.

[Dau14] John Daugman. 600 million citizens of India are now enrolled with biometric id,". SPIE newsroom, 7, 2014.

[DHP21] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In USENIX Security, page 143–160, 2021.

[FMC+15] Benjamin Fuller, Darby Mitchell, Robert Cunningham, Uri Blumenthal, Patrick Cable, Ariel Hamlin, Lauren Milechin, Mark Rabe, Nabil Schear, Richard Shay, et al. Security and privacy assurance research (SPAR) pilot final report. Technical report, MIT Lincoln Laboratory, 2015.

[FMC+20] Francesca Falzon, Evangelia Anna Markatou, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. Full database reconstruction in two dimensions. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 443–460, 2020.

[FMET22] Francesca Falzon, Evangelia Anna Markatou, Zachary Espiritu, and Roberto Tamassia. Range search over encrypted multi-attribute data. Proceedings of the VLDB Endowment, 16(4):587–600, 2022.

[Fou] Electronic Frontier Foundation. Mandatory national ids and biometric databases.

[FP22] Francesca Falzon and Kenneth G. Paterson. An efficient query recovery attack against a graph encryption scheme. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, Computer Security – ESORICS 2022, pages 325–345, Cham, 2022. Springer International Publishing.

[FVK+15] Ben A Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M Bellovin. Malicious-client security in blind seer: a scalable private dbms. In IEEE S&P, pages 395–410. IEEE, 2015.

[FVY+17] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. SoK: Cryptographically protected database search. In IEEE S&P, pages 172–191. IEEE, 2017.

[FWG+16] Zhangjie Fu, Xinle Wu, Chaowen Guan, Xingming Sun, and Kui Ren. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE TIFS, 11(12):2706–2716, 2016.

[GJW19] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In CCS, page 361–378, 2019.

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Eurocrypt, pages 370–396. Springer, 2021.

[GLMP18] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In CCS, pages 315–331, 2018.

[GLMP19] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In In 2019 IEEE Symposium on Security and Privacy (SP), page 1067–1083, 2019.

[GPP23] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. In IEEE S&P, 2023.

[GPPW24] Zichen Gui, Kenneth G Paterson, Sikhar Patranabis, and Bogdan Warinschi. Swissse: System-wide security for searchable symmetric encryption. Proceedings on Privacy Enhancing Technologies, 2024.

[GRS17] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. In Proceedings of the 16th workshop on hot topics in operating systems, pages 162–168, 2017.

[GRS22] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential Coding Theory. 2022.

[GSB+17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In Security and Privacy (SP), 2017 IEEE Symposium on, pages 655–672. IEEE, 2017.

[HCD+23] Julie Ha, Chloe Cachet, Luke Demarest, Sohaib Ahmad, and Benjamin Fuller. Private eyes: Zero-leakage iris searchable encryption. Cryptology ePrint Archive, Paper 2023/736, 2023. https://eprint.iacr.org/2023/736.

[HKR+24] Mahdieh Heidaripour, Ladan Kian, Maryam Rezapour, Mark Holcomb, Benjamin Fuller, Gagan Agrawal, and Hoda Maleki. Organizing records for retrieval in multi-dimensional range searchable encryption. Cryptology ePrint Archive, 2024.

[HSWW18] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. Multi-key searchable encryption, revisited. In PKC, pages 95–124. Springer, 2018.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In NDSS, volume 20, page 12. Citeseer, 2012.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In STOC, pages 604–613, 1998.

[KE19] Evgenios M Kornaropoulos and Petros Efstathopoulos. The case of adversarial inputs for secure similarity approximation protocols. In 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pages 247–262. IEEE, 2019.

[KIK12] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. Efficient similarity search over encrypted data. In IEEE Data Engineering, pages 1156–1167. IEEE, 2012.

[KKM+22] Seny Kamara, Abdelkarim Kati, Tarik Moataz, Thomas Schneider, Amos Treiber, and Michael Yonli. Sok: Cryptanalysis of encrypted search with leaker - a framework for leakage attack evaluation on real-world data. In Euro S&P, 2022.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In CCS, pages 1329–1340, 2016.

[KM19] Senny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In EUROCRYPT 2019, volume 11477. Springer, 2019.

[KMO18] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakage suppression. In Crypto, pages 339–370. Springer, 2018.

[KP10] Ajay Kumar and Arun Passi. Comparison and combination of iris matchers for reliable personal authentication. Pattern recognition, 43(3):1016–1026, 2010.

[KPT19] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In IEEE S&P, pages 1033–1050. IEEE, 2019.

[KPT20] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: attacks on encrypted databases beyond the uniform query distribution. In IEEE S&P, pages 1223–1240, 2020.

[LMWY20] Kasper Green Larsen, Tal Malkin, Omri Weinstein, and Kevin Yeo. Lower bounds for oblivious near-neighbor search. In SODA, pages 1116–1134. SIAM, 2020.

[LPW+20] Qin Liu, Yu Peng, Jie Wu, Tian Wang, and Guojun Wang. Secure multi-keyword fuzzy searches with enhanced service quality in cloud computing. IEEE Transactions on Network and Service Management, 18(2):2046–2062, 2020.

[LZWaT14] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu an Tan. Search pattern leakage in searchable encryption: Attacks and new construction. In Information Sciences, volume 265, pages 176–188, 2014.

[Mic97] Daniele Micciancio. Oblivious data structures: applications to cryptography. In STOC, page 456–464, New York, NY, USA, 1997. Association for Computing Machinery.

[MT19] Evangelia Anna Markatou and Roberto Tamassia. Full database reconstruction with access and search pattern leakage. In International Conference on Information Security, pages 25–43. Springer, 2019.

[OK21] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In USENIX Security, 2021.

[PPYY19] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In CCS, 2019.

[RW23] Kui Ren and Cong Wang. Security impact of leakage profiles: Threats and countermeasures. In Searchable Encryption: From Concepts to Systems, pages 77–105. Springer, 2023.

[SDS+18] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. Journal of the ACM (JACM), 65(4):1–26, 2018.

[SWP00] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In IEEE S&P, pages 44–55. IEEE, 2000.

[WHL16] Boyang Wang, Yantian Hou, and Ming Li. Practical and secure nearest neighbor search on encrypted large-scale data. In INFOCOM, pages 1–9. IEEE, 2016.

[WHL20] Boyang Wang, Yantian Hou, and Ming Li. Quickn: Practical and secure nearest neighbor search on encrypted large-scale data. IEEE Transactions on Cloud Computing, 10(3):2066–2078, 2020.

[WLD+17] Guofeng Wang, Chuanyi Liu, Yingfei Dong, Hezhong Pan, Peiyi Han, and Binxing Fang. Query recovery attacks on searchable encryption based on partial knowledge. In International Conference on Security and Privacy in Communication Systems, pages 530–549. Springer, 2017.

[WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In CCS, pages 215–226, 2014.

[WP21] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption with optimal search time. In Asia CCS, pages 252–264, 2021.

[WSL+22] Jianfeng Wang, Shi-Feng Sun, Tianci Li, Saiyu Qi, and Xiaofeng Chen. Practical volume-hiding encrypted multi-maps with optimal overhead and beyond. In CCS, page 2825–2839, New York, NY, USA, 2022. Association for Computing Machinery.

[WYLH14] Bing Wang, Shucheng Yu, Wenjing Lou, and Y Thomas Hou. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In IEEE INFOCOM, pages 2112–2120. IEEE, 2014.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In 25th USENIX Security Symposium, pages 707–720, 2016.

[ZWX+23] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T Yang, and Kaitai Liang. High recovery with fewer injections: practical binary volumetric injection attacks against dynamic searchable encryption. In USENIX Security, pages 5953–5970, 2023.

A Baseline Construction

The goal when to searching for a value y is to retrieve the set $\texttt{Close}$ without receiving any indices in $\texttt{Far}$. We informally present the baseline LSH scheme, as described in prior work [KIK12, FWG+16, WYLH14, LPW+20, BT21, HCD+23], to introduce the relevant accuracy parameters. Let $\mathsf{LSH}_1, ..., \mathsf{LSH}_n$ be a sampled set of LSHs and treat a record a relevant for a value y if they agree on a single LSH value. The output is the set for $1 \leq j \leq n$:

$$\{x_i | \{(j, \mathsf{LSH}_j(x_i))\} \cap \{(j, \mathsf{LSH}_j(y))\} \neq \emptyset\}.$$

The construction is as follows:

Construction 2 (LSH & Multimap based APSS)**.** *Let t be a distance parameter, $c > 1$ and let $\mathcal{DB} \in \{0,1\}^{\gamma \times M}$. Let LSH be a $(t, ct, \epsilon_{\texttt{t}}, \epsilon_{\texttt{f}})$ be a LSH family. Let MM be a multimap. Define $\mathsf{APSS} = (\mathsf{APSS.Init}, \mathsf{APSS.Setup}, \mathsf{APSS.Search})$ as in Algorithm 2. Then following notation from Definition 5, for all $y_1, ..., y_q$, and all well-spread $\mathcal{DB}$, $\forall i$*

$$\Pr\left[\texttt{Far}(y_i, \mathcal{DB}) \cap J_i = \emptyset\right] \geq 1 - \delta_{\texttt{Far}},$$
$$\Pr\left[\texttt{Close}(y_i, \mathcal{DB}) \subseteq J_i\right] \geq 1 - \delta_{\texttt{Close}},$$

For

$$\delta_{\texttt{Far}} = 1 - (1 - \epsilon_{\texttt{f}})^{nM},$$
$$\delta_{\texttt{Close}} = (1 - \epsilon_{\texttt{t}})^{n}.$$

That is, APSS is a $(t, c, \delta_{\texttt{Far}}, \delta_{\texttt{Close}})$-approximate proximity search scheme.

Finding n for baseline construction For fixed $\epsilon_{\texttt{t}}, \epsilon_{\texttt{f}}$ it suffices to set

$$\frac{\log(\delta_{\texttt{Close}})}{\log(1 - \epsilon_{\texttt{t}})} \leq n \leq \frac{\log(\delta_{\texttt{Far}})}{M \log(1 - \epsilon_{\texttt{f}})}. \tag{4}$$

In particular, in the setting when $\delta_{\texttt{Close}} \approx \delta_{\texttt{Far}}$ and for small $\epsilon_{\texttt{t}}, \epsilon_{\texttt{f}}$ where $\log(1 - x) \approx -x$ for n to exist in Equation 4 it must be the case that $\epsilon_{\texttt{t}} \geq M\epsilon_{\texttt{f}}$.

In the case when the LSH is an extended LSH with underlying error rates of $\epsilon'_{\texttt{t}}, \epsilon'_{\texttt{f}}$ with α concatenated copies and $\epsilon'_{\texttt{t}} > \epsilon'_{\texttt{f}}$ then setting

$$\alpha \geq \frac{\log(M)}{\log(\epsilon'_{\texttt{t}}/\epsilon'_{\texttt{f}})}. \tag{5}$$

suffices for

$$\left(\frac{\epsilon'_{\texttt{t}}}{\epsilon'_{\texttt{f}}}\right)^{\alpha} = \frac{\epsilon_{\texttt{t}}}{\epsilon_{\texttt{f}}} \geq M.$$

This means that

$$n \approx \frac{M \log(\delta_{\texttt{Close}})}{\log(\delta_{\texttt{Far}})}. \tag{6}$$

Note that $\delta_{\texttt{Close}}$ exactly corresponds with $\texttt{TAR}$ for a well-spread database. However, $\delta_{\texttt{Far}}$ controls the overall probability of a false accept and is a much stronger condition than controlling the $\texttt{FAR}$. In Appendix B, we analyze the $\texttt{FAR}$ of the baseline scheme for random data where each record in the database has exactly $\epsilon_{\texttt{f}}$ probability of matching an LSH and each query that is a noisy version of a stored x_i has probability exactly

Algorithm 2 Baseline construction of APSS from LSH and MM.All procedures from perspective of Client with calls to underlying interactive protocols.

$\texttt{Init}\begin{pmatrix}1^\lambda\\1^\lambda\end{pmatrix} = \mathsf{MM}.\texttt{Init}\begin{pmatrix}1^\lambda\\1^\lambda\end{pmatrix}$

$\mathsf{APSS.Setup}_n\begin{pmatrix}\mathcal{DB}=(x_1,...,x_M),\mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$:

1. Sample n LSHs $\mathsf{LSH}_1,...,\mathsf{LSH}_n \leftarrow \mathsf{LSH}$.
2. Set $\mathcal{DB}_{\mathsf{MM}} = \{(j,\mathsf{LSH}_j(x_i)),x_i\}_{j=1,...,M,i=1,...,n}$.
3. Execute $\begin{pmatrix}\perp\\ \mathcal{I}_0\end{pmatrix} \leftarrow \mathsf{MM.Setup}_n\begin{pmatrix}\mathcal{DB}_{\mathsf{MM}},\mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$.
4. Output $(\mathsf{LSH}_1,...,\mathsf{LSH}_n)$ to Client.

$\mathsf{APSS.Search}_n\begin{pmatrix}y_i,\mathsf{LSH}_1,...,\mathsf{LSH}_n\\ \mathcal{I}_{(i-1)\cdot n}\end{pmatrix}$:

1. For $j=1$ to n, compute $\begin{pmatrix}x_j\\ \mathcal{I}_{(i-1)\cdot n+j}\end{pmatrix} \leftarrow \mathsf{MM.Search}_n\begin{pmatrix}(j,\mathsf{LSH}_j(y))\\ \mathcal{I}_{(i-1)\cdot n+(j-1)}\end{pmatrix}$.
2. Output $J_i = \cup_{j=1}^n x_j$.

$\epsilon_{\mathtt{t}}$ of colliding LSH with the stored reading of the biometric. In that appendix, for the baseline scheme we use report FAR as $\delta_{\mathtt{FAR}}$ for consistency with ProxCode, which is presented shortly. This means we are comparing ProxCode against a baseline scheme with a weaker correctness guarantee.

As described in the Introduction, there are three main issues with Construction 2:

1. The use of a multimap. Constructing oblivious multimaps is a difficult prospect (see discussion in [KMO18, GKM21, GPP23, AG22, RW23]), and
2. The use of a disjunctive query requires $\epsilon_{\mathtt{f}}$ to be very small and n to be very large to support reasonable $\delta_{\mathtt{Close}}, \delta_{\mathtt{Far}}$. Tables 3 and 6 highlight this comparison. Further discussion on parameter analysis can be found in Appendix B.
3. In the three party searchable encryption scenario, unintended biometrics are (occasionally) leaked to clients.

B Algorithmic Parameter Analysis for Random Data

This section compares the efficiency of the baseline scheme with ProxCode. During this discussion, we assume that all biometrics in the database are far yielding probability $\epsilon_{\mathtt{f}}$ of their LSHs matching, and that all queries are closing yielding probability of $\epsilon_{\mathtt{t}}$ of matching an LSH with the relevant stored record. These assumptions are useful for analysis but not true in practice, see discussion in Section 4. Our evaluation focuses on the number of required LSHs. We first compute accuracy parameters for ProxCode and then find corresponding parameters for the baseline scheme for the same accuracy.

B.1 Evaluation Methodology

We take the smallest values for α, n, and k that satisfy equations 1 and 2 simultaneously . Our parameter finding was done in Python 3.9.

Recall that for a (n,k)-Reed-Solomon Code to decode successfully (Definition 2) it suffices that $\mathsf{k}_{correct} > 2k$ and $\mathsf{k}_{error} \leq k$. Our evaluation uses an augmented LSH so we assume for some $\alpha \in \mathbb{Z}^+$ that $\epsilon_{\mathtt{t}} = \epsilon'^{\alpha}_{\mathtt{t}}$ and $\epsilon_{\mathtt{f}} = \epsilon'^{\alpha}_{\mathtt{f}}$.

We assume the bit selection LSH $\mathsf{LSH}_i(x) = x_i$ which has the property that

$$\Pr[\mathsf{LSH}(x) = \mathsf{LSH}(y)] = \frac{\gamma - \mathcal{D}(x,y)}{\gamma} = 1 - \mathcal{D}(x,y)/\gamma.$$

We test with different parameters $\epsilon'_{\mathtt{t}}, \epsilon'_{\mathtt{f}}$ which represent the noise between different readings of the same biometric and readings of different biometrics respectively. Errors between readings of the same biometric and differences between readings of different biometrics both come from distributions. So for two different values $x_i, x_j \in \mathcal{DB}$, one will frequently observe $\mathcal{D}(x,y) < .5\gamma$. Even if the average FHD between readings of the same biometric is .1 one observes errors of at least .2. See Figure 3. This is why we test for values of $\epsilon'_{\mathtt{f}} \in \{.5, .6, .7\}$. We consider $\epsilon'_{\mathtt{t}} \in \{.7, .75, .80, .85, .9\}$. Our results exclude values where no solutions could be found with $\log_{10}(n) \leq 20$. We provide a full methodology next.

B.1.1 Detailed Methodology

For input constants c_1, c_2 we search for settings of α, n, k such that

$$(\epsilon'_{\mathtt{t}})^{\alpha} > \frac{2k}{(1-c_2)n}, \tag{7}$$

$$(\epsilon'_{\mathtt{f}})^{\alpha} \leq \frac{k}{Mn(1+c_1)}. \tag{8}$$

Increasing α exponentially decreases both the true accept rate and false accept rate. Thus, we first find the minimum α that produces a solution for n, k. Combining the Equations 7 and 8 one has that:

$$M(1+c_1)(\epsilon'_{\mathtt{f}})^{\alpha} \leq \frac{k}{n} < \frac{1}{2}(1-c_2)(\epsilon'_{\mathtt{t}})^{\alpha}.$$

We compute the minimum α such that

$$M \cdot (1+c_1)(\epsilon'_{\mathtt{f}})^{\alpha} \leq \frac{1}{2}(1-c_2)(\epsilon'_{\mathtt{t}})^{\alpha}$$

Now using the computed α, we find the first integer n that satisfies the following inequality:

$$M \cdot (1+c_1)(\epsilon'_{\mathtt{f}})^{\alpha} \cdot n \leq \left(\frac{1}{2}(1-c_2)(\epsilon'_{\mathtt{t}})^{\alpha} + 1\right) \cdot n$$

With α, and n we can easily find the set of possible as solutions to:

$$M \cdot (1+c_1)(\epsilon'_{\mathtt{f}})^{\alpha} \cdot n \leq k < \frac{1}{2}(1-c_2)(\epsilon'_{\mathtt{t}})^{\alpha} \cdot n. \tag{9}$$

As we show next the value of k, is strongly connected to the error probabilities in Lemma 1 and Lemma 2. Thus, we exclude solutions where $k < 20$ or k is not an integer.

Lastly, we check the probability that setup fails according to Lemma 3. We compute

$$\delta_{\mathtt{Far}} \leq exp\left(\frac{-c_1^2}{2+c_1} \cdot \epsilon'^{\alpha}_{\mathtt{f}} \cdot n \cdot M\right) \approx exp\left(\frac{-c_1^2 k}{(2+c_1)(1+c_1)}\right)$$

$$\delta_{\mathtt{Close}} \leq exp\left(\frac{-c_2^2}{2} \cdot \epsilon'^{\alpha}_{\mathtt{t}} \cdot n\right) + \delta_{\mathtt{Far}} \approx exp\left(\frac{-c_2^2 k}{1-c_2}\right) + \delta_{\mathtt{Far}}$$

We estimated the minimum value of ℓ such that Setup has probability of at least .99 of completing within ℓ iterations using Equation 3.

Computing parameters for baseline scheme Recall for the baseline scheme described in Construction 2 one has $\delta_{\texttt{Far}} = (1-\epsilon_{\texttt{f}}'^{\alpha})^{nM}$ and $\delta_{\texttt{Close}} = (1-\epsilon_{\texttt{t}}'^{\alpha})^{n}$. We solve the following two equations to compute α and n in the Baseline scheme.

$$\texttt{FAR} = Mn\epsilon_{\texttt{f}}'^{\alpha},$$
$$\delta_{\texttt{Close}} = (1-\epsilon_{\texttt{t}}'^{\alpha})^{n}$$

As mentioned above, we require the baseline scheme to have the same FAR as our $\delta_{\texttt{Far}}$. This is a much weaker condition. For example, for a dataset of size $M = 10^6$ and $\delta_{\texttt{Far}} = 10^{-4}$ corresponds to a $\texttt{FAR} \approx 10^{-10}$.

B.2 Required Number of LSHs

Our parameter analysis focuses on three different database sizes when $M = 10^6, 10^4$ and $M = 10^3$ representing a country wide specialized database, a large organization, and a medium size organization.

Table 3 compares the bounds on the number and size of the LSHs (n, and α), as well as the number of needed matches (k) in ProxCode with the same parameters in the Baseline scheme. (Recall, we allow the baseline scheme to have FAR equal to our $\delta_{\texttt{Far}}$ so we allow the baseline scheme false matches with every query.) Table 3 computes these parameters over multiple values of the constants c_1, c_2 which control the accuracy of the system. In the body, Table 3 gives a summary and Table 6 presents full results. Both tables report the base 10 log of n.

Discussion The smaller the value of $\delta_{\texttt{Far}}$ the more ProxCode improves over the baseline scheme. Furthermore, the more noise is present, represented by a decrease in $\epsilon_{\texttt{t}}$ the more ProxCode improves over the baseline scheme.

For $\epsilon_{\texttt{t}}' = .9$ the difference in $\log n$ between ProxCode and in baseline scheme is negative (across all three accuracy regimes). As error increases, for example, $\epsilon_{\texttt{t}}'$ to .7, ProxCode presents major improvement. This improvement is largest in higher accuracy regime with $\delta_{\texttt{Far}} = 10^{-6}$ and smaller with $\delta_{\texttt{Far}} = 10^{-3}$. The gap between $\log n$ is similar across sizes of databases M though the absolute size has a strong dependence on M. The summary comparison is presented in Table 1.

For instance, looking at Table 3, setting $c_1 = 3$, and $c_2 = .4$ results in a high accuracy setting yielding:

- $\delta_{\texttt{Far}} = 10^{-4}$.
- $\delta_{\texttt{Close}}$ varies in size between the order of 10^{-4} and 10^{-3}.

The improvements are most pronounced when the gap between $\epsilon_{\texttt{t}}'$ and $\epsilon_{\texttt{f}}'$ is smallest. As an example, when $\epsilon_{\texttt{f}}' = .6$ and $\epsilon_{\texttt{t}}' = .75$ (represented in table 6), for $M = 10^4$ records in the Baseline scheme we need $n = 10^{12.1}$ (with LSHs of size $\alpha = 91$), ProxCode requires $n = 10^{8.4}$ (with $\alpha = 53$). There are some cases where there is a large difference between underlying LSH error rates ($\epsilon_{\texttt{t}}' = .9, \epsilon_{\texttt{f}}' = .5$) where ProxCode performs worse requiring approximately 60% more LSHs. These are "easy cases" when few LSHs are required. However, ProxCode often makes drastic improvements: when $\epsilon_{\texttt{f}}' = .6$ and $\epsilon_{\texttt{t}}' = .80$ one moves from almost a 100 million LSHs to a million. Improvements follow the same pattern for the setting of $M = 10^3$ and $M = 10^6$.

Impact of reducing α Although our prime interest is to decrease n, we can see that we also have improvement in the value of α. This improvement is in all testing parameters. Current oblivious maps [BT21, HCD+23] build trees and obliviously traverse them, the LSH values are used to decide which child to visit. Decreasing α allows one to use a tree with a larger branching factor. This in turn decreases the number of communication rounds. So decreasing α improves efficiency even if n remains the same.

Probability of setup completing Assuming ℓ to be the number of iterations for the Setup to succeed. For the setting of $\delta_{\texttt{Far}}$, one has

$$\left(1 - \prod_{i=2}^{M} (1 - \delta_{\texttt{Far},i})\right)^{\ell} \leq \eta$$

$\delta_{\texttt{Far}} = 10^{-3}$		$M = 10^6$						$M = 10^4$						$M = 10^3$					
		Baseline		ProxCode				Baseline		ProxCode				Baseline		ProxCode			
$\epsilon'_{\texttt{f}}$	$\epsilon'_{\texttt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$
.7	.85	117	9.1	84	7.8	24	10^{-3}	93	7.4	60	6.1	22	2×10^{-3}	81	6.5	48	5.2	22	2×10^{-3}
.7	.9	91	5.0	65	4.8	24	10^{-3}	72	4.1	46	3.9	20	3×10^{-3}	63	3.7	37	3.5	21	2×10^{-3}
.6	.75	102	13.6	73	11.0	22	10^{-3}	81	10.9	52	8.3	21	2×10^{-3}	71	9.7	42	7.1	23	10^{-3}
.6	.8	79	8.5	57	7.5	26	8×10^{-4}	63	6.9	41	5.9	26	8×10^{-4}	55	6.2	33	5.1	26	8×10^{-4}
.6	.85	66	5.6	47	5.3	25	10^{-3}	53	4.7	34	4.3	27	6×10^{-4}	46	4.2	27	3.8	24	10^{-3}
.6	.9	56	3.4	40	3.7	22	2×10^{-3}	45	2.9	26	3.2	25	10^{-3}	39	2.6	23	2.9	22	2×10^{-3}
.5	.7	67	11.1	48	9.3	20	4×10^{-3}	49	8.4	34	7.1	27	2×10^{-3}	42	7.3	27	6	26	3×10^{-3}
.5	.75	56	7.8	40	6.9	22	2×10^{-3}	41	6.0	20	3.9	36	4×10^{-4}	35	5.2	23	4.8	33	7×10^{-4}
.5	.8	49	5.7	35	5.4	27	5×10^{-4}	38	4.7	25	4.4	37	2×10^{-4}	31	4.0	20	3.9	36	4×10^{-4}
.5	.85	43	3.9	31	4.2	27	5×10^{-4}	32	3.3	22	3.5	35	5×10^{-4}	28	3.1	18	3.3	41	10^{-4}
.5	.9	39	2.7	28	3.2	28	5×10^{-4}	29	2.4	20	2.9	38	2×10^{-4}	25	2.2	16	2.7	36	4×10^{-4}
$\delta_{\texttt{Far}} = 10^{-4}$		Baseline		ProxCode				Baseline		ProxCode				Baseline		ProxCode			
$\epsilon'_{\texttt{f}}$	$\epsilon'_{\texttt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$
.7	.85	129	9.9	85	7.8	21	2×10^{-3}	104	8.1	61	6.1	20	3×10^{-3}	92	7.2	49	5.2	20	4×10^{-3}
.7	.9	100	5.4	66	4.9	21	10^{-3}	81	4.5	47	3.9	20	4×10^{-3}	72	4.1	38	3.5	21	3×10^{-3}
.6	.75	112	14.8	74	11.1	21	2×10^{-3}	91	12.1	53	8.4	20	4×10^{-3}	81	10.9	43	7.2	22	2×10^{-3}
.6	.8	87	9.3	58	7.5	21	8×10^{-4}	71	7.7	42	6.0	26	8×10^{-4}	63	6.9	34	5.2	26	8×10^{-4}
.6	.85	72	5.9	48	5.3	21	6×10^{-4}	58	4.8	34	4.2	20	3×10^{-3}	52	4.5	28	3.9	25	10^{-3}
.6	.9	62	3.7	41	3.7	21	10^{-3}	51	3.3	30	3.3	28	4×10^{-4}	45	2.9	24	3.0	25	10^{-3}
.5	.7	74	12.2	49	9.4	26	3×10^{-3}	61	10.3	36	7.5	27	6×10^{-4}	54	9.2	29	6.4	25	9×10^{-4}
.5	.75	62	8.6	41	7.0	30	10^{-3}	51	7.3	30	5.7	28	4×10^{-4}	45	6.5	24	4.9	25	10^{-3}
.5	.8	53	5.9	35	5.2	25	3×10^{-3}	44	5.2	26	4.5	30	2×10^{-4}	39	4.7	21	4.0	29	4×10^{-4}
.5	.85	47	4.1	31	4.0	25	3×10^{-3}	39	3.7	**23**	**3.6**	**29**	$\mathbf{3 \times 10^{-4}}$	34	3.2	18	3.1	21	3×10^{-3}
.5	.9	43	2.9	28	3.1	26	3×10^{-3}	36	2.8	**21**	**3.0**	**34**	$\mathbf{10^{-4}}$	32	2.6	17	2.8	32	10^{-4}
$\delta_{\texttt{Far}} = 10^{-6}$		Baseline		ProxCode				Baseline		ProxCode				Baseline		ProxCode			
$\epsilon'_{\texttt{f}}$	$\epsilon'_{\texttt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$	α	$\log n$	α	$\log n$	k	$\delta_{\texttt{Close}}$
.7	.85	142	10.7	87	7.9	21	4×10^{-3}	119	9.2	63	6.2	20	4×10^{-3}	108	8.5	52	5.5	24	10^{-3}
.7	.9	110	5.8	67	4.9	20	3×10^{-3}	92	5.0	49	4.1	22	2×10^{-3}	83	4.6	40	3.7	23	2×10^{-3}
.6	.75	124	16.2	76	11.3	23	2×10^{-3}	104	13.8	55	8.7	21	3×10^{-3}	94	12.6	45	7.5	22	2×10^{-3}
.6	.8	96	10.0	59	7.6	23	10^{-3}	81	8.7	43	6.0	23	10^{-3}	73	7.9	35	5.2	23	10^{-3}
.6	.85	79	6.3	49	5.3	25	10^{-3}	67	5.6	36	4.5	27	5×10^{-4}	60	5.0	29	3.9	24	10^{-3}
.6	.9	68	3.8	42	3.8	24	10^{-3}	58	3.6	31	3.4	28	4×10^{-4}	52	3.3	25	3.0	25	10^{-3}
.5	.7	83	13.6	50	9.5	20	4×10^{-3}	69	11.5	37	7.6	25	10^{-3}	63	10.7	30	6.5	24	10^{-3}
.5	.75	69	9.5	42	7.1	24	10^{-3}	58	8.2	31	5.8	28	4×10^{-4}	52	7.4	25	5.0	25	10^{-3}
.5	.8	59	6.5	36	5.3	22	2×10^{-3}	50	5.7	26	4.3	20	4×10^{-3}	45	5.3	22	4.1	30	2×10^{-4}
.5	.85	53	4.7	30	3.5	23	2×10^{-3}	45	4.3	**24**	**3.7**	**33**	$\mathbf{10^{-4}}$	40	3.8	19	3.2	23	10^{-3}
.5	.9	48	3.2	29	3.2	25	10^{-3}	40	2.8	**21**	**2.8**	**22**	$\mathbf{2 \times 10^{-3}}$	36	2.6	17	2.6	21	2×10^{-3}

Table 6: Parameters Comparison between our ProxCode and Baseline scheme where $M \cdot \epsilon'^{\alpha}_{\texttt{f}}.n = \delta_{\texttt{Far}}$ and $(1 - \epsilon'^{\alpha}_{\texttt{t}})^n = \delta_{\texttt{Close}}$. In ProxCode parameters are computed as in Section B. The numbers for α, n, k are the first found solutions. For the baseline scheme we measure FAR while in ProxCode we measure $\delta_{\texttt{Far}}$ this allows the baseline scheme to have more errors for the same accuracy. Accuracy $\delta_{\texttt{Far}} \approx 10^{-3}$ from $c_1 = 2$ and $c_2 = .4$. Accuracy $\delta_{\texttt{Far}} \approx 10^{-4}$ from $c_1 = 3, c_2 = .4$ and accuracy $\delta_{\texttt{Far}} \approx 10^{-6}$ from $c_1 = 5, c_2 = .4$. Logarithms are base 10.

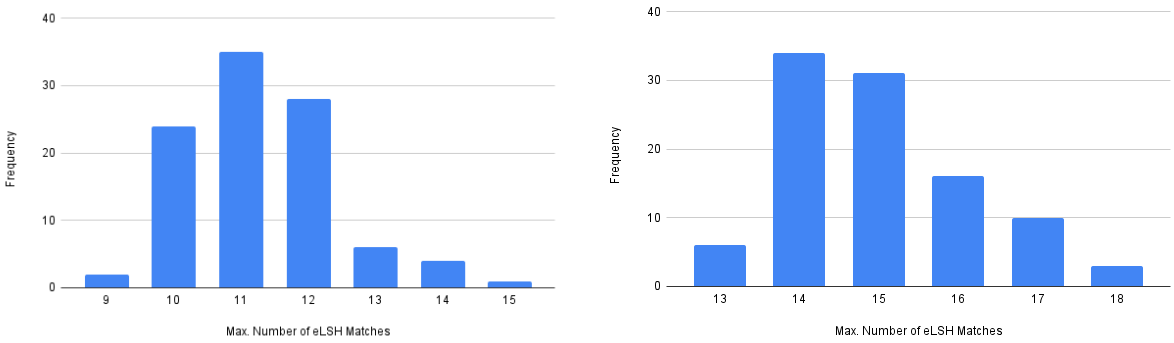
where η is the probability of failure. Requires that

$$\ell \geq \frac{\log(\eta)}{\log\left(1 - \prod_{i=2}^{M}(1 - \delta_{\texttt{Far},i})\right)}.$$

For our choice of parameters $\delta_{\texttt{Far}}$, and n, we always have,

$$1 - \prod_{i=2}^{M}(1 - \delta_{\texttt{Far},i}) \approx 0$$

This behavior held true regardless of the size of the dataset, giving evidence that $\ell = 1$ suffices. We note that we performed this computation with floating point arithmetic and its known inaccuracies. For our implementation, Section 4, we do observe parameters where setup takes a multiple ≤ 10 iterations to succeed.



(a) Histogram of maximum number of matches for $c_1 = 5$, and $\epsilon_{\mathtt{t}} = .9$

(b) Histogram of maximum number of matches for $c_1 = 3$, and $\epsilon_{\mathtt{t}} = .9$

Figure 4: Histograms of the maximum number of eLSH matches between dataset records in the Setup phase. Results are from 100 runs of Setup. $M = 10^4$, $\epsilon_{\mathtt{f}} = .5$.

C Implementation on Random Data

We generate parameters algorithmically in Appendix B, we include a summary in Table 3. For all tested parameters, random data only required a single iteration for setup to complete. We choose instances from parameters regime in Table 3 to evaluate accuracy and how close Setup came to failing. The parameters chosen for testing are in **bold** in Table 3. For all tests the datasets chosen were i.i.d. samples from $\{0,1\}^{1024}$, the output length of the ThirdEye feature extractor [AF19a, ACD+22] used for real data in the next subsection.

For $c_1 = 5$, Figure 4(a) shows the maximum number of eLSH matches that a record (across all M records) shares with its predecessors is in the range of $[9, 15]$, which is less than problematic threshold of $k = 22$. For $c_1 = 3$, Figure 4(b) shows the maximum number of matches between two records lies between 13 to 18. This is dramatically less than the upper bound of $k = 34$. This confirms our analysis that Setup has a high probability of succeeding. Both histograms are from 100 runs of Setup.

We also tested Search, over a dataset of size 10^4 and two sets of queries with mean error rate of 0.1 and 0.15 from a value stored in the database. Errors were drawn from a binomial distribution the above listed fractional mean and standard deviation $\sigma = .056$. This standard deviation is drawn from recent work on proximity search for the iris [HCD+23, Section 4]. Use of this technique means that some numbers of errors are not possible, yielding discontinuities in the same histogram (Figure 3). Table 3 is used to set the number of LSHs and needed matches. We randomly chose 100 database entries and generated a corresponding query according to the above statistics.

For each set of chosen parameters, accuracy results are shown in Table 7 separated by the actual fraction of errors between the query and the value stored in the database. We only observed incomplete or incorrect results when the fraction of errors was over 120% of the target error rate. In the computation of the observed $\delta_{\mathtt{Far}}$, we counted all recovered responses that are not the correct value whether or not it is a value in the dataset.

	$\epsilon_{\mathtt{t}} = .9$				$\epsilon_{\mathtt{t}} = .85$			
c_1	FHD	TAR	$\delta_{\mathtt{Far}}$	Avg Erasures	FHD	TAR	$\delta_{\mathtt{Far}}$	Avg Erasures
3	$\leq .10$	1	0	.61	$\leq .15$	1	0	.90
	$\leq .12$	1	0	.73	$\leq .18$	1	0	.92
	$\leq .14$	.95	.01	.77	$\leq .21$	84	.01	.93
5	$\leq .10$	1	0	.59	$\leq .15$	1	0	.90
	$\leq .12$	1	0	.66	$\leq .18$	1	0	.91
	$\leq .14$	.90	0	.71	$\leq .21$	.93	0	.92

Table 7: Correctness on Random Data. Experiments are performed for $M = 10^4$ records and $\epsilon_{\mathtt{f}} = .5$ with setting $c_1 = 3$ or $c_1 = 5$. FHD is the actual fractional Hamming distance between the query and the relevant database item. Avg Erasures is the average fraction of codewords that contain erasures.