ProxCode: Efficient Biometric Proximity Searchable Encryption from Error Correcting Codes

Maryam Rezapour* Benjamin Fuller†

June 20, 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 oblivious 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. We evaluate the scheme 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]. See prior reviews [BHJP14, FVY+17, KKM+22, RW23a, 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]. Security biometric databases [BBOH96, Dau14, Fou] is a major application of this type of search.

Let $\mathcal{DB} = x_1, ..., x_M$ be a collection of records. We focus on the Hamming distance metric. That is, for x, y, the distance between x and y is the number of positions that are not equal, denoted $\mathcal{D}(x, y) = |\{i | x_i \neq y_i\}|$. The Hamming distance metric is frequently used in iris recognition [Dau09].[1] For a distance parameter t and query y, the goal of search is to find the set $\texttt{Res} = \{x_i \in \mathcal{DB} | \mathcal{D}(x_i, y) \leq t\}$.[2] For biometric databases, one assumes for all y there exists at most one $x \in \mathcal{DB}$ such that $\mathcal{D}(y, x) \leq t$.

Prior Approaches Prior work [KIK12, BT21, HCD+23] combines locality sensitive hashes or LSHs [IM98] and oblivious/encrypted (multi)maps [WNL+14]. LSHs map close items to the same value more frequently than they map far items to the same value. Multimaps, MM, allow association of keywords with values x_i. A multimap has two operations:

1. $\mathsf{MM}.\texttt{add}(\texttt{keyword}, \texttt{value})$ that associates `value` with `keyword`, and
2. $\mathsf{MM}[\texttt{keyword}]$ which returns all values previously associated with `value`.[3]

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

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

[1] Our techniques apply to any metric with locality sensitive hashes [IM98].

[2] A related goal is (approximate) k-nearest neighbors where the goal is to retrieve the k closest records [BT21]. There have been leakage abuse attacks against k-nearest neighbor systems that reveal access pattern [KPT19, KPT20, LMWY20, CCD+20].

[3] We use the notation of dynamic multimaps [MM17, AKM19a, AKM19b, GPPW24, GKM21, APP+23] for simplicity in the Introduction, but our techniques are applicable to the static setting.

For some n number of LSHs, a multimap MM, and LSH family LSH, consider the following **Baseline** construction [KIK12, BT21, HCD+23]:

1. $\mathsf{Setup}(x_1, ..., x_M)$
   (a) Sample n LSHs, $\mathsf{LSH}_1, ..., \mathsf{LSH}_n \leftarrow \mathsf{LSH}$.
   (b) For $j = 1, ..., n$ & $i = 1, ..., M$,
   $$\mathsf{MM}.\mathtt{add}(\mathtt{keyword} = (j, \mathsf{LSH}_j(x_i)), \mathtt{value} = x_i).$$
2. $\mathtt{Search}(y)$:
   (a) Compute $\mathsf{LSH}_1(y), ..., \mathsf{LSH}_n(y)$.
   (b) Lookup $\cup_{j=1}^{n} \mathsf{MM}[(j, \mathsf{LSH}_j(y))]$.

Leakage of Multimaps Constructions of (dynamic) multimaps [SWP00, KMO18, GKM21, GPP23, AG22, RW23b, APP+23, WSL+22, PPYY19][4] have nonzero leakage including query equality. Patel et al. [PPYY19] showed that avoiding query equality requires higher overhead techniques similar to oblivious RAM (for map leakage suppression, see [GKM21]).

The need for accurate search All LSH-based solutions have imperfect accuracy. The two accuracy parameters are:

1. $\delta_{\mathtt{Close}}$ measures how frequently the close record is not returned, and
2. $\delta_{\mathtt{Far}}$ 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 1.1.1).

1.1 Our Contribution

This work introduces a data preprocessing method for accurate proximity search. Our approach is to transform the query from a disjunction to a k-out-of-n query. We call the system `ProxCode` for efficient proximity search from error correcting codes. We switch to using notation of a map as our system only associates one value with each keyword. The high-level approach proceeds in two stages:

1. Secret sharing x_i and inserting shares into the map and
2. Using the coding properties of secret sharing to ensure that only a single value is associated with each keyword in the map.

At search time, one collects $\geq k$ results from the map and uses these shares to reconstruct the relevant record (or record position).

Moving to shares Consider some fixed record x_i and compute the LSH values $\mathsf{LSH}_1(x_i)$, ..., $\mathsf{LSH}_n(x_i)$. Instead of directly associating x_i with these keywords, we create a linear secret sharing of x_i.[5] Let c_i be some codeword such that $\mathsf{c}_{i,1} = i$. We assume codewords are drawn from a linear code that is also a μ-out-of-n secret sharing [Sha79]. Then one adds to the M the pairs $\mathsf{M}.\mathtt{add}\left((j, \mathsf{LSH}_j(x_i)), \mathsf{c}_{i,j+1}\right)$. If there are enough matches, the client retrieves enough points on the codeword c_i and can reconstruct c_i and thus x_i.

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

[5] 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 4, associating x_i prevents a second lookup with associated leakage [GPPW24, GPP23].

Error	Improvement ($\log_{10}(n)$)		
Rate	$\delta_{\texttt{Far}} = 10^{-3}$	$\delta_{\texttt{Far}} = 10^{-4}$	$\delta_{\texttt{Far}} = 10^{-6}$
.10	-0.5	-0.2	0
.15	-0.2	0.1	0.6
.20	0.3	0.7	1.4
.25	2.1	1.6	2.4
.30	1.3	2.8	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 A. We note the substantial improvement for the high error rate regime. Dataset of size $M = 10^4$.

Dealing with LSH collisions The second step of our approach is associating each LSH output to a single value. Consider two values x_α and x_β such that $\mathsf{LSH}_j(x_\alpha) = \mathsf{LSH}_j(x_\beta)$ for some index j. One samples the codewords $\mathsf{c}_\alpha, \mathsf{c}_\beta$ uniformly under the constraint that $\mathsf{c}_{\alpha,j} = \mathsf{c}_{\beta,j}$.

The ability to perform this sampling is guaranteed by the fact that the code is a good secret sharing meaning that $\mathsf{c}_{\alpha,1}$ has a uniform distribution conditioned on $\mu - 1$ other symbols.

Each LSH collision between records x_i, x_j of the database causes two codewords to share a single symbol. There is a set of good codewords as long as no record x_i has no more than $\mu - 1$ LSH collisions with other x_j (More precisely, more than $\mu - 1$ distinct symbols have collisions). Our analysis shows for the error regimes present in biometrics one can sample a set of M codewords under this constraint using fewer LSHs than the Baseline *disjunctive search*'s requirement.

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 $\approx 20\%$. Their analysis suggests $n = 80$ LSHs suffices to capture the distribution mean. However, $n \approx 1000$ 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 $|\mathsf{M}|$ that must be stored. For an error rates of 25%, our improvement of at least $10^{1.6} \approx 39$. Ha et al.'s recent construction [HCD$^+$23] required 26 rounds of communication, 1571 seconds of server computation, and 35GB of storage for 5000 records. Private-eyes [HCD$^+$23] have indexing overhead of 22 and cryptographic of 291. Our work is reducing 22, this is orthogonal to cryptographic improvements. Efficiency improvements are crucial for scaling secure and private biometric databases.

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. As accuracy degrades two phenomena occur:

1. Overall fewer LSHs are required and the baseline scheme outperforms ours for low noise rates, and
2. It is harder to find a set of codewords satisfying the above constraints as there are more LSH collisions. This makes it more difficult for setup to complete.

One expects many erasures with LSH values matching nothing in the map. Throughout, we use Reed-Solomon codes which are a good secret sharing and naturally handle a mix of erasures and errors.

This approach can be secured using any map. In this work, we consider security when instantiated with 1) a map that leaks search [LZWaT14, OK21] & access pattern [SWP00, CGKO06] and 2) an oblivious map such as [BT21] which uses oblivious data structures [WNL$^+$14, Mic97]. 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].

Further Prior Work Ha et. al [HCD$^+$23] presented the first zero-leakage iris proximity search system called private-eyes. Private-eyes is built using the zero-leakage k-nearest-neighbor system [BT21]. Our scheme sits on top of any secure map implementation such as [HCD+23, BT21]. This is why we focus our comparison on the number of LSHs for a given accuracy level. Our scheme provides a drop in improvement for those mechanisms.

1.1.1 Implementation

We present a prototype implementation of the above `ProxCode` scheme integrated into an unprotected map (Github). We evaluate 1) the accuracy on 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.

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 means are, this is shown in Table 2. 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 2), at this setting $\delta_{\texttt{Close}} = .2$. The mean error rate is used to set parameters.

For real data, we set data empirically, for ≈ 200 irises achieving $\delta_{\texttt{Close}} \approx .1$ with $n = 1000$ LSHs (see Table 4). As we discuss real data has more variance than random data requiring larger values of α and more careful parameter tuning. For both real and random data, we observed $\delta_{\texttt{Far}} \approx .01$.

Since we use maps, our subquery and subaccess leakage is 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.

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 5.3). At a high level, `ProxCode` requires a persistent client [GRS17] with many queries to learn anything about any stored biometric.

Organization Section 2 introduces preliminary notation including the definition of APSS. In Section 3, we formalize the baseline construction. Section 4 presents `ProxCode` and proves that it is an APSS.Section 5 presents accuracy for real and random data. Section 6 concludes and discusses future work. Appendix A explains the methodology of parameters evaluation for 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 linear 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 5.
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. 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. For a positive integer x, let $[x]$ denote the set $\{1, ..., x\}$. For a prime power p, we use $\mathbb{F}_p$ to denote the field over $[p]$. 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. *A map $\mathsf{M} = (\mathsf{M}.\mathit{insert}, \mathsf{M}.\mathit{retrieve})$ is a pair of algorithms where*

1. *$\mathsf{M}.\mathit{insert}(L, R)$: Adds (L, R) where L is the key and R is its associated value.*
2. *$\mathsf{M}.\mathit{retrieve}(L)$: 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, keywords 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 Coding Theory

Definition 3 (Linear Codes [GRS22])**.** *For prime power p, a set $\mathtt{C} \subseteq \mathbb{F}_p^n$ is a $(n, k, k_{correct})$-error correcting code, if $|\mathtt{C}| = p^k$ and $\forall x_1, x_2 \in \mathtt{C}$ it is true that*

$$\mathcal{D}(x_1, x_2) < n - k_{correct}.$$

$\mathtt{C}$ is a linear code *if $\mathtt{C}$ is a linear subspace (of dimension k) of $\mathbb{F}_p^n$.*

We use $\mathbf{A}_{\mathtt{C}}$ to refer to a generating matrix of a linear error-correcting code, one such matrix always exists. We will need our codes to satisfy a slightly non-standard condition that we call μ-wise independence. This condition designates that minors of $\mathbf{A}_{\mathtt{C}}$ with at most μ rows have full rank.

Definition 4 (μ-wise independence)**.** *Let $\mathtt{C}$ be a $(n, k, k_{correct})$-linear error correcting code. For $\mu \leq k$, $\mathtt{C}$ is μ-wise independent if $\forall i \leq k$ for all $\mathbf{A}' \in \mathbb{F}_p^{i \times k}$ minors of $\mathbf{A}_{\mathtt{C}}$, it is true that*

$$\mathtt{rank}(\mathbf{A}') \geq \min\{i, \mu\}.$$

We use the abbreviate this condition as a $(n, k, k_{correct}, \mu)$-linear independent code.

A μ-wise independent code is a linear secret sharing [Sha79] against an adversary that sees μ shares: one selects a random codeword with a first symbol determined by a message and distributes symbols of the codeword as shares.

In the setting that $\mu = k$ this condition requires all $k \times k$ minors to be full rank which implies that the code is a maximum distance separable code. Reed-Solomon codes satisfy this condition. However, many codes satisfy μ-wise independence when $\mu < k$.

Claim 1. *Let* $\mathtt{C}$ *be a* $(n, k, \mathsf{k}_{correct}, \mu)$*-linear independent error correcting code. Then there exists an efficient procedure* Inv *such that given* $(i_1, y_{i_1}, ..., i_\mu, y_{i_\mu})$ *one can sample a uniform codeword* $\tilde{y}$ *such that* $\tilde{y}_{i_j} = y_{i_j}$ *for* $j = 1, ..., \mu$.

Proof. Let $\mathbf{A}'$ be the $\mu \times k$ minor of $\mathbf{A}$ with rows $i_1, ..., i_\mu$. By the independence condition, $\mathbf{A}'$ has row rank μ. There must exist some square minor of dimension $\mu \times \mu$ of $\mathbf{A}'$ that has rank μ. Without loss of generality suppose that this minor contains the first μ columns. Let $\mathbf{A}_{\mathtt{sq}}$ denote this square minor and let $\mathbf{A}_{\mathtt{Remain}}$ denote the last $k - \mu$ columns. Then we define Inv as below:

1. Sample $x_{\mu+1}, ..., x_k$ uniformly randomly.
2. Solve
$$\begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_\mu \end{pmatrix} = \mathbf{A}_{\mathtt{sq}}^{-1} \left(\begin{pmatrix} y_{i_1} \\ ... \\ y_{i_\mu} \end{pmatrix} - \mathbf{A}_{\mathtt{Remain}} \begin{pmatrix} x_{\mu+1} \\ ... \\ x_k \end{pmatrix} \right)$$
3. Output
$$\mathbf{A} \begin{pmatrix} x_1 \\ ... \\ x_k \end{pmatrix}.$$

Efficiency and consistency of Inv can be easily verified. □

Definition 5. *For prime power* p*, let* $\mathtt{C}$ *be a* $(n, k, \mathsf{k}_{correct}, \mu)$*-linear error correcting code over* $\mathbb{F}_p^n$*. Let* $y \in (\mathbb{F}_p \cup \perp)^n$ *and let* $\mathcal{I}_{\mathtt{Valid}}$ *denote the locations of* y *that are not* $\perp$*. For all* $x \in \mathtt{C}$ *define the set* Decode_x *as the set of all* y *such that*

1. $|\{i | x_i = y_i\}| \geq \mathsf{k}_{correct}$.
2. $|\{i | x_i \neq y_i \wedge y_i \neq \perp\}| \leq \mathsf{k}_{error}$.

Let $\mathsf{k}_{error} < \mathsf{k}_{correct}$ *be a parameter.* $\mathtt{C}$ *is a* k_{error}*-code with erasures if there exists an efficient procedure* Decode *such that*

1. *For all* $x \in \mathtt{C}$ *and for all* $y \in \mathtt{Decode}_x$ *it holds that* $\Pr[x \leftarrow \mathsf{Decode}(y)] = 1$.
2. *For all* y *if* $|\{i | y_i \neq \perp\}| \leq \mathsf{k}_{error}$, $\Pr[x \leftarrow \mathsf{Decode}(y) \wedge x = \perp] = 1$.

We abbreviate this as a $(n, k, \mathsf{k}_{correct}, \mathsf{k}_{error}, \mu)$*-linear error correcting code.*

Definition 5 implicitly bounds the number of erasures, it may be up to $n - \mathsf{k}_{correct} - \mathsf{k}_{error}$. The second condition is slightly nonstandard, it simply there to ensure that the code doesn't attempt to decode when it doesn't have enough information to determine the codeword (even without errors), any code can be modified to satisfy this condition.

Definition 6 (Reed-Solomon Codes [GRS22])**.** *The* (n, k) *Reed-Solomon code over* $\mathbb{F}_p$ *is the*

$$\{\mathtt{C} | \mathtt{C}_i = P(i) \textit{ for some } k-1 \textit{ degree } P\}$$

Such codes are linear with the Vandermonde matrix representing one $\mathbf{A}$ *(and* enc *algorithm).*

A (n, k) *Reed-Solomon code is an* $(n, k, \mathsf{k}_{correct} = 2k, \mathsf{k}_{error} = k, \mu = k)$ *code with erasures with the Berlekamp-Welch algorithm [WB86] representing one* Decode*.*

2.2 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 7 (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*

$$\begin{aligned}\texttt{Close}(y, \mathcal{DB}) &= \{x_i \mid x_i \in \mathcal{DB} \;\&\; \mathcal{D}(x_i, y) \le t\},\\ \texttt{Far}(y, \mathcal{DB}) &= \{x_i \mid x_i \in \mathcal{DB} \;\&\; \mathcal{D}(x_i, y) \ge ct\}.\end{aligned}$$

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

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

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

As discussed in the Introduction, Definition 7 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 5 consider real data.

Definition 8 (Approximate Proximity Search Scheme). *Consider Definition 9 security, let $\mathsf{APSS} = (\mathsf{APSS.Init}, \mathsf{APSS.Setup}, \mathsf{APSS.Search})$. For $c > 1, t \in \mathbb{Z}^+$ APSS is a $(t, c, q, \delta_{\texttt{Far}}, \delta_{\texttt{Close}})$-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, $\forall i, 1 \le i \le q$,

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

where $\texttt{Far}$, $\texttt{Close}$ *are defined as in Definition 7.*

Definition 9 (Adaptive Security for Search Protocol). *Let $\mathsf{SSE} = (\mathsf{Init}, \mathsf{Setup}, \mathsf{Search})$ be a triple of algorithms with associated leakage functions $(\mathcal{L}^{\mathsf{Setup}}, \mathcal{L}^{\mathsf{Search}})$. Let λ be a security parameter.*

For an adversary $\mathcal{A}$ and simulator $\mathcal{S}$ define $Exp_{\mathsf{SSE},\mathcal{A}}(\cdot)$ and $Exp_{\mathcal{S},\mathcal{A}}(\mathcal{L} = (\mathcal{L}^{\mathsf{Setup}}, \mathcal{L}^{\mathsf{Search}}))$ 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

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

We use Definition 9 for maps, multimaps, and approximate proximity schemes, which we denote at M, MM and APSS respectively. We consider the following leakage functions:

1. $\mathcal{L}^{\mathsf{Setup}}_{\mathsf{Size}}$ 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. $\mathcal{L}^{\mathsf{Search}}_0$ which leaks the occurrence of a query [BIPW17, BT21], and

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

1. $\mathsf{ts}_{\mathsf{Init}} \leftarrow \mathcal{S}(1^\lambda)$.
2. $D \leftarrow \mathcal{A}(\mathsf{ts}_{\mathsf{Init}})$.
3. $\mathsf{ts}_0 \leftarrow \mathcal{S}(\mathcal{L}^{\mathsf{Setup}}(D)))$.
4. For $i = 1$ to q:
   (a) $y_i \leftarrow \mathcal{A}(tk_{i-1})$.
   (b) $\mathsf{ts}_i \leftarrow \mathcal{S}(\mathcal{L}^{\mathsf{Search}}(y_i))$.
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. $\mathcal{L}^{\mathsf{Search}}_{\mathsf{AccPatt}}$ which leaks identifiers returned with a query [SWP00, CGKO06]. These identifiers are consistent across queries.
4. $\mathcal{L}^{\mathsf{Search}}_{\mathsf{QueryEq}}$ 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` APSS system. They are 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. $\mathcal{L}^{\mathsf{Search}}_{\mathsf{SubAccPatt}}$ 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. $\mathcal{L}^{\mathsf{Search}}_{\mathsf{SubQueryEq}}$ for an integer n, leaks query equality over subqueries.

3 Baseline Construction

The goal when to searching for a value y is to retrieve of `Close` without receiving any indices in `Far`. We informally present the baseline LSH scheme 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 1 (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 $(t, ct, \epsilon_{\mathtt{t}}, \epsilon_{\mathtt{f}})$ be a LSH family. Let* MM *be a multimap. Define $\mathsf{APSS} = (\mathsf{APSS.Init}$,* *APSS.Setup, APSS.Search) as in Algorithm 1. Then following notation from Definition 8, 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

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

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 1 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

$$\alpha \geq \frac{\log(M)}{\log(\epsilon'_{\texttt{t}}/\epsilon'_{\texttt{f}})}. \tag{2}$$

suffices for

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

This means that

$$n \approx \frac{M \log(\delta_{\texttt{Close}})}{\log(\delta_{\texttt{Far}})}. \tag{3}$$

Note that $\delta_{\texttt{Close}}$ exactly corresponds with `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 `FAR`. In Appendix A, we analyze the `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 $\epsilon_{\texttt{t}}$ of colliding LSH with the stored reading of the biometric. In that appendix, for the baseline scheme we use report FAR as $\delta_{\texttt{FAR}}$ 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 1:

1. The use of a multimap. Constructing oblivious multimaps is a difficult prospect (see discussion in [KMO18, GKM21, GPP23, AG22, RW23b]), and
2. The use of a disjunctive query requires $\epsilon_{\texttt{f}}$ to be very small and n to be very large to support reasonable $\delta_{\texttt{Close}}, \delta_{\texttt{Far}}$. Tables 3 and 5 highlight this comparison. Further discussion on parameter analysis can be found in Appendix A.
3. In the three party searchable encryption scenario, unintended biometrics are (occasionally) leaked to clients.

4 `ProxCode`

This section formally introduces `ProxCode`, proves it is secure, and proves it is accurate under the well-spread condition (Definition 7). This condition is used for analysis but *not* assumed in our evaluation in Section 5.

Our construction combines LSHs and a secure map. Instead of associating LSH outputs with records, we associate LSH outputs with shares of a linear secret sharing. One then collects multiple shares and decodes,

Algorithm 1 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 $J_i = \cup_{j=1}^n x_j$.

---

our search only reveals a matching record x_i when there are *enough* LSH matches. To do this, instead of directly associating $((j, \mathsf{LSH}_j(x_i)), x_i)$ we encode x_i onto a linear error correcting code. That is, we associate x_i with a random codeword c_i such that $\mathsf{c}_{i,1} = x_i$.

To handle *LSH collisions*, when $z = \mathsf{LSH}_j(x_i) = \mathsf{LSH}(x_k)$, we constrain the two codewords to have the same value at position j. That is, that $\mathsf{c}_{i,j} = \mathsf{c}_{k,j}$. We can do this without impacting either $\mathsf{c}_{i,1}$ or $\mathsf{c}_{k,1}$ because of the independence property of the code, which says the code is a good secret sharing (Definition 4). We assume the existence of an Inv algorithm that maps a set of codeword symbols to a uniform codeword with those symbols. Assuming one can sample a set $\mathsf{c}_1, ..., \mathsf{c}_M$ then one can effectively perform an k-out-of-n search in place of the pure 1-out-of-n search used in the Baseline scheme. We present this scheme formally in Algorithm 2 and Construction 2. 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 2. *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 $\mu \leq k < n$.*

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 $p \geq M$. Let $\mathbf{A} \in \mathbb{F}_p^{n\times k}$ be a generating matrix of $(n+1, k, k_{correct}, k_{error}, \mu)$-linear code with associated algorithms $\mathsf{Decode}_{\mathbf{A}}, \mathsf{Inv}_{\mathbf{A}}$.*
3. *Let $\mathsf{M} = (\mathsf{M}.\mathsf{insert}, \mathsf{M}.\mathsf{retrieve})$ be a map.*

Define APSS as in Algorithm 2.

Algorithm 2 `ProxCode`: APSS from maps and linear (secret-sharing) codes.
Procedures are run by Client unless calling an underlying interactive protocol.

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

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

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 $\texttt{Eq}$ with more than $\mu - 1$ 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 $\mathbf{C}_i = \mathsf{Inv}(\texttt{NEmpty}(\mathbf{C}_i))$. $\texttt{NEmpty}$ 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. $\begin{pmatrix}\perp\\ \mathcal{I}_0\end{pmatrix} \leftarrow \mathsf{M.Setup}\begin{pmatrix}\mathcal{DB},\mathsf{sk}\\ \mathsf{pp}\end{pmatrix}$.

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

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 $\begin{pmatrix}\mathsf{c}_{j+1}\\ \mathcal{I}_{(i-1)\cdot n+j}\end{pmatrix} = \mathsf{M.Search}\begin{pmatrix}(j,L_j),\mathsf{sk}\\ \mathcal{I}_{(i-1)\cdot n+(j-1)},\mathsf{pp}\end{pmatrix}$.
   (b) If $\mathsf{c}_{j+1} \neq \perp$, $\mathsf{e}_{erase} := \mathsf{e}_{erase} - 1$.
4. If $n - \mathsf{e}_{erase} > \mathsf{k}_{correct}$ output $\perp$.
5. Compute $\mathsf{c}_1,...,\mathsf{c}_\ell \leftarrow \mathsf{Decode}_{\mathbf{A}}(\perp||\mathsf{c}_2||...||\mathsf{c}_n)$, output c_1

Definition 5 on the definition of a linear code is in supplemental material, the parameters are the length, dimension, the number of needed correct symbols, the maximum number of errors, and the number of independent symbols of the code. A (n,k)-Reed-Solomon code is an $(n,k,\mathsf{k}_{correct} = 2k, \mathsf{k}_{error} = k, \mu = k)$ code with erasures with the Berlekamp-Welch algorithm [WB86] representing one Decode (Definition 6).

We provide some intuition for the scheme before presenting our formal results. There are two main ideas in Construction 2:

Shares in the Map First, we replace x_i as the value inserted into the map with a codeword whose first symbol 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 $\mathtt{Eq}$ the set of all LSH collisions in the database. If two values x_i, x_k share some $\mathsf{LSH}_j(x_i) = \mathsf{LSH}_j(x_k)$ then we will fix $\mathsf{c}_{i,j} = \mathsf{c}_{k,j}$. We rely on the μ independence of the linear code to ensure that we can describe a set of codewords under these constraints (Definition 4). 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 4.2.

4.1 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. Let C be a $(n + 1, k, \mathsf{k}_{correct}, \mathsf{k}_{error}, \mu)$-code with erasures over $\mathbb{F}_p$ where $p \geq M$ and let M be a map (with perfect correctness). Suppose the following are true:*

$$\epsilon_{\mathtt{t}} > \frac{\mathsf{k}_{correct}}{(1 - c_2)n}, \tag{4}$$

$$\epsilon_{\mathtt{f}} \leq \frac{\mathsf{k}_{error}}{Mn(1 + c_1)}, \tag{5}$$

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 2 instantiated with C and n LSHs from LSH is an $(t, c, \delta_{\mathtt{Far}}, \delta_{\mathtt{Close}}) - \mathsf{APSS}$. Furthermore,

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

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

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

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

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

by Equation 4. By independence of the LSHs, Match_{x,x^*} is bounded below by a $(n, \frac{\mathsf{k}_{correct}}{(1-c_2)n})$ binomial distribution with $\mathsf{Exp}[\mathsf{Match}_{x,x^*}] = \frac{\mathsf{k}_{correct}}{(1-c_2)}$. Then by a standard Chernoff bound, it is true that

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

This completes the proof of Lemma 1. $\square$

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 w' 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

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

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

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

This completes the proof of Lemma 2. $\square$

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

$$\Pr[\mathsf{Setup} \textit{ outputs } \perp] \leq \left(1 - \prod_{i=2}^{M}(1 - \delta_{\mathtt{Far},i})\right)^{\ell} \tag{6}$$

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 exists by Definition 7. 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 $\mathsf{k}_{correct}$ 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 $1 - (\delta_{\mathtt{Close},1} + \delta_{\mathtt{Close},2})$ Thus, by Definition 5, conditioned on both of these events occurring Decode outputs c_i with probability 1.

Far Items not in Result Set By Lemma 2 the probability that c' has more than k_{error} symbols other than $\perp$ is at most $\delta_{\mathtt{Far}}$. The fact that Decode outputs $\perp$ is by Step 4 of Search in Algorithm 2.

This completes the proof of Theorem 1. □

4.2 Security and Leakage

This section shows that when the M in Construction 2 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 $\mathsf{M} = (\mathsf{M}.\mathit{Setup}, \mathsf{M}.\mathit{Search})$ be a map that is secure according to Definition 9 for $\mathcal{L}_{\mathsf{M}} = (\mathcal{L}^{\mathsf{Setup}} = |\mathsf{M}|, \mathcal{L}^{\mathsf{Search}} = (\mathcal{L}^{\mathsf{Search}}_{\mathit{AccPatt}}, \mathcal{L}^{\mathsf{Search}}_{\mathit{QueryEq}}))$. Then the $\mathit{APSS} = (\mathit{APSS.Init}, \mathit{APSS.Setup}, \mathit{APSS.Search})$ scheme defined in Construction 2 is secure according to Definition 9 for $(\mathcal{L}^{\mathsf{Setup}} = n \cdot |\mathsf{M}|, \mathcal{L}^{\mathsf{Search}} = (\mathcal{L}^{\mathsf{Search}}_{\mathit{SubAccPatt}}, \mathcal{L}^{\mathsf{Search}}_{\mathit{SubQueryEq}}))$.*

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

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

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

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

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

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 $\mathcal{A}_{\mathsf{M}}$ that is a valid M adversary. Let $\mathcal{S}_{\mathsf{M}}$ be one such simulator for $\mathcal{A}_{\mathsf{M}}$. Note that setup leakage is the same in both settings. For the search leakage, $(\mathcal{L}^{\mathsf{Search}}_{\mathsf{SubAccPatt}}, \mathcal{L}^{\mathsf{Search}}_{\mathsf{SubQueryEq}})$ this allows $\mathcal{S}_{\mathsf{APSS}}$ to expand the q queries into qn subqueries which is the required leakage for $\mathcal{S}_{\mathsf{M}}$. Then

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

This completes the proof of Lemma 4. □

4.3 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.

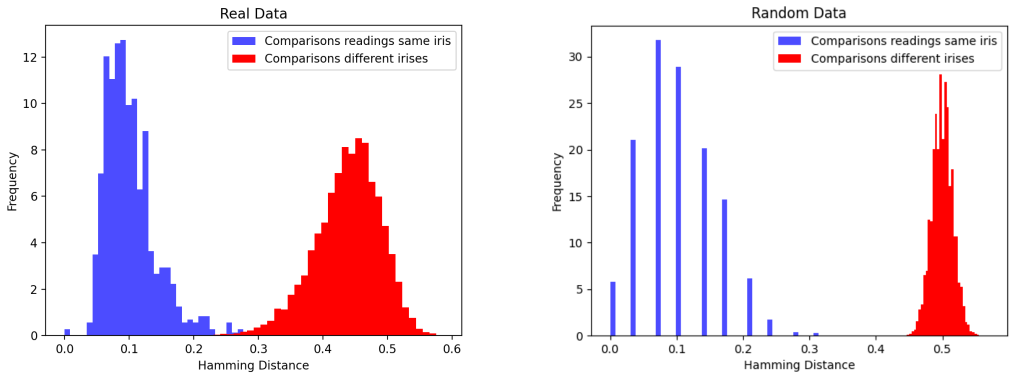
5 Implementation and Accuracy

We implemented `ProxCode` using Python 3.9. 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.

5.1 Random Data

We generate parameters algorithmically in Appendix A, 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



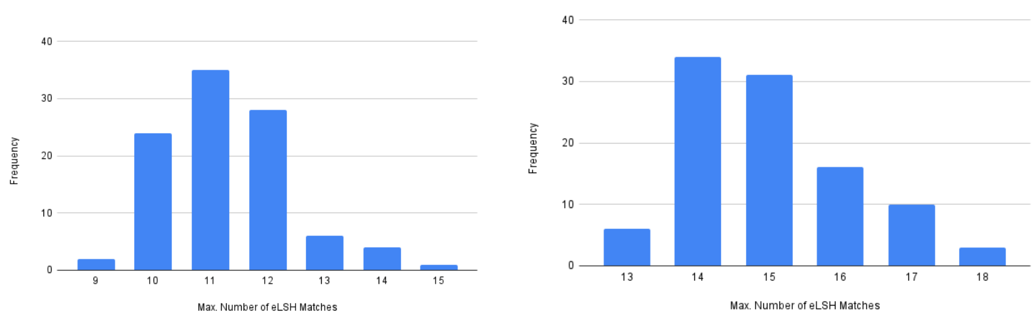
(a) IITD dataset

(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.

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.



(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. $M = 10^4$, $\epsilon_{\mathtt{f}} = .5$.

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

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

Table 2: 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.

$M = 10^4$															
$\delta_{\mathtt{Far}} = 10^{-4}$		Baseline		`ProxCode`				$\delta_{\mathtt{Far}} = 10^{-6}$		Baseline		`ProxCode`			
$\epsilon'_{\mathtt{f}}$	$\epsilon'_{\mathtt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\mathtt{Close}}$	$\epsilon'_{\mathtt{f}}$	$\epsilon'_{\mathtt{t}}$	α	$\log n$	α	$\log n$	k	$\delta_{\mathtt{Close}}$
.5	.7	61	10.3	36	7.5	27	6×10^{-4}	.5	.7	69	11.5	37	7.6	25	10^{-3}
.5	.75	51	7.3	30	5.7	28	4×10^{-4}	.5	.75	58	8.2	31	5.8	28	4×10^{-4}
.5	.8	44	5.2	26	4.5	30	2×10^{-4}	.5	.8	50	5.7	26	4.3	20	4×10^{-3}
.5	.85	39	3.7	**23**	**3.6**	**29**	$\mathbf{3 \times 10^{-4}}$	.5	.85	45	4.3	**24**	**3.7**	**33**	$\mathbf{10^{-4}}$
.5	.9	36	2.8	**21**	**3.0**	**34**	$\mathbf{10^{-4}}$	.5	.9	40	2.8	**21**	**2.8**	**22**	$\mathbf{2 \times 10^{-3}}$

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

according to the above statistics.

For each set of chosen parameters, accuracy results are shown in Table 2 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.

5.2 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.

We used the same methodology that we used to compute Table 5, 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.

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.

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, 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.

5.3 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 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 $2 \le \mathbf{C}_{i,j} \le n+1$ 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].

5.4 Time Efficiency

Our savings in n will translate to any encrypted map, see discussion in the Introduction. For time, we only measure time to run Algorithm 2 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 $M = 10^4$ and $n = 631, 1000, 3982, 5012$ `Setup` completed in $237, 434, 7415$ and 9939 seconds respectively.

6 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 International Conference on Biometrics: Theory, Applications and Systems*, 2019.

[AF19b] Sohaib Ahmad and Benjamin Fuller. Unconstrained iris segmentation using convolutional neural networks. In *Computer Vision–ACCV 2018 Workshops: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers 14*, 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.

[AKM19a] Ghous Amjad, Senny Kamara, and Tarik Moataz. Breach-resistant structured encryption. In *Proceedings on Privacy Enhancing Technologies, 2019(1)*, page 245–265, 2019.

[AKM19b] Ghous Amjad, Senny Kamara, and Tarik Moataz. Forward and backward private searchable encryption with sgx. In *In Proceedings of the 12th European Workshop on Systems Security - EuroSec '19*, pages 1–6, 2019.

[APP+23] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proceedings on Privacy Enhancing Technologies*, 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 Proceedings on Privacy Enhancing Technologies*, 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 *29th USENIX Security Symposium (USENIX Security 20)*, pages 2111–2128, 2020.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, 2006.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 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 *In 30th USENIX Security Symposium (USENIX Security 21)*, 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 *2015 IEEE Symposium on Security and Privacy*, 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 *2017 IEEE Symposium on Security and Privacy (SP)*, 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 Transactions on Information Forensics and Security*, 11(12):2706–2716, 2016.

[GJW19] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, page 361–378, 2019.

[GKM21] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III*, 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 *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 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 Security and Privacy*, 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 *Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I 21*, 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 *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 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 *2012 IEEE 28th International Conference on 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 *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340, 2016.

[KMO18] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. Structured encryption and leakage suppression. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*, 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 *2019 IEEE Symposium on Security and Privacy (SP)*, 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 *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1223–1240. IEEE, 2020.

[LMWY20] Kasper Green Larsen, Tal Malkin, Omri Weinstein, and Kevin Yeo. Lower bounds for oblivious near-neighbor search. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 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 *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 456–464, New York, NY, USA, 1997. Association for Computing Machinery.

[MM17] Ian Miers and Payman Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. In *In Proceedings 2017 Network and Distributed System Security Symposium, San Diego, CA, 2017.*, 2017.

[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 *In 30th USENIX Security Symposium (USENIX Security 21)*, 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 *In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, 2019.

[RW23a] Kui Ren and Cong Wang. *Searchable Encryption: From Concepts to Systems*. Springer Nature, 2023.

[RW23b] 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.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

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

[WB86] L Welch and ER Berlekamp. Error correction for algebraic block codes. *US Patent*, 4633470:16–40, 1986.

[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 *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226, 2014.

[WP21] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption with optimal search time. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 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 *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, 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 2014-IEEE conference on computer communications*, 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 *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5953–5970, 2023.

A 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 5. 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.

A.1 Evaluation Methodology

We take the smallest values for α, n, and k that satisfy equations 4 and 5 simultaneously . Our parameter finding was done in Python 3.9.

Recall that for a (n, k)-Reed-Solomon Code to decode successfully (Definition 6) 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 $\epsilon_{\mathtt{t}} = \epsilon_{\mathtt{t}}'^{\alpha}$ and $\epsilon_{\mathtt{f}} = \epsilon_{\mathtt{f}}'^{\alpha}$.

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.

A.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 6.

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

$$\mathtt{FAR} = Mn\epsilon'^{\alpha}_{\mathtt{f}},$$

$$\delta_{\mathtt{Close}} = (1-\epsilon'^{\alpha}_{\mathtt{t}})^n$$

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

A.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 $\delta_{\mathtt{Far}}$ 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 5 presents full results. Both tables report the base 10 log of n.

		$M=10^6$						$M=10^4$						$M=10^3$					
$\delta_{\texttt{Far}}=10^{-3}$		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**	$\mathbf{3\times10^{-4}}$	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**	$\mathbf{2\times10^{-3}}$	36	2.6	17	2.6	21	2×10^{-3}

Table 5: 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 `ProxCode` parameters are computed as in Section A. 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 $\delta_{\texttt{Far}}\approx10^{-3}$ from $c_1=2$ and $c_2=.4$. Accuracy $\delta_{\texttt{Far}}\approx10^{-4}$ from $c_1=3,c_2=.4$ and accuracy $\delta_{\texttt{Far}}\approx10^{-6}$ from $c_1=5,c_2=.4$. Logarithms are base 10.

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

$\epsilon'_{\mathtt{f}} = .6$ and $\epsilon'_{\mathtt{t}} = .75$ (represented in table 5), 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 ($\epsilon'_{\mathtt{t}} = .9, \epsilon'_{\mathtt{f}} = .5$) 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 $\epsilon'_{\mathtt{f}} = .6$ and $\epsilon'_{\mathtt{t}} = .80$ 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 $\delta_{\mathtt{Far}}$, one has

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

where η is the probability of failure. Requires that

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

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

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

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 5, we do observe parameters where setup takes a multiple ≤ 10 iterations to succeed.