Private Eyes: Zero-Leakage Iris Searchable Encryption

Julie Ha* Chloe Cachet† Luke Demarest‡ Sohaib Ahmad§ Benjamin Fuller¶

May 25, 2023

Abstract

This work introduces *Private Eyes* the first zero-leakage biometric database. The only leakage of the system is unavoidable: 1) the log of the dataset size and 2) the fact that a query occurred. Private Eyes is built from symmetric searchable encryption. Proximity queries are the required functionality: given a noisy reading of a biometric, the goal is to retrieve all stored records that are close enough according to a distance metric.

Private Eyes combines locality sensitive-hashing or LSHs (Indyk and Motwani, STOC 1998) and oblvious maps. One searches for the disjunction of the LSHs of a noisy biometric reading. The underlying encrypted map needs to efficiently answer disjunction queries.

We focus on the iris biometric. Iris biometric data requires a large number of LSHs, approximately 1000. The most relevant prior work is in zero-leakage k-nearest-neighbor search (Boldyreva and Tang, PoPETS 2021), but that work is designed for a small number of LSHs.

Our main cryptographic tool is a zero-leakage disjunctive map designed for the setting when most clauses do not match any records. For the iris, on average at most 6% of LSHs match any stored value. Our scheme is implemented and open-sourced. For the largest tested parameters of a 5000 iris database, search requires 26 rounds of communication and 26 minutes of single-threaded computation.

To evaluate we use a simple generative adversarial network to produce synthetic irises. Accurate statistics on sizes beyond available datasets is crucial to optimizing the cryptographic primitives. This tool may be of independent interest.

Keywords: Searchable Encryption, Biometrics, Proximity Search

1 Introduction

Biometrics are collected into large databases for search [BBOH96, Dau14, Fou]. Learning stored biometric values enables an attacker to break authentication and privacy for a user's lifetime [GRGB+12, MCYJ18, AF20, VS11, HWKL18, SDDN19]. To reduce this risk, this article develops new searchable encryption techniques for biometric databases [SWP00, CGKO11].[1]

Consider the setting where a client outsources a database $\mathcal{DB}$ to an honest but curious server. To be efficient, the server may learn some information known as *leakage*. Prior work exploits such leakage to reveal sensitive information about the database or queries [IKK12, CGPR15, KKNO16, WLD+17, GSB+17, GLMP18, KPT19a, MT19, KE19, KPT20, FMC+20, FP22, GPP23]. Since biometrics cannot be replaced or revoked, we focus on a *zero-leakage* system. A zero-leakage system leaks only unavoidable information: 1) that a query occurs and 2) $|\mathcal{DB}|$ (which can be padded to a power of 2).

We present `PrivateEyes`**, the first zero-leakage iris proximity search system.** Our system is implemented and tested on datasets of up to 5000 records with a search time of at most 26 minutes and 26 round trips.[2] Throughout,

*Boston University. Email `hajulie@bu.edu`
†University of Connecticut. Email `chloe.cachet@uconn.edu`
‡University of Connecticut. Email `luke.h.demarest@gmail.com`
§University of Connecticut. Email `sohaib.ahmad@uconn.edu`
¶University of Connecticut. Email `benjamin.fuller@uconn.edu`

[1] A full review of searchable encryption is out of scope. See previous reviews [BHJP14, FVY+17, KKM+22].

[2] We report on the single threaded execution time and parallelized number of round trips for our system. Our approach and relevant prior work are embarrassingly parallel.

we focus on the iris (we briefly discuss the face in Section 2.1). Recently, Boldyreva and Tang [BT21] built a zero-leakage k-nearest-neighbor system. Boldyreva and Tang did not build proximity search; our comparison is based on adapting their scheme. On the 5000 record dataset, this adaptation has a search time of approximately 7.9 hours.

Like prior secure proximity search systems (starting with [KIK12]), our system combines locality-sensitive hashes [IM98], LSHs, and (a variant of) encrypted maps. Our design results from improving the efficiency gaps present in databases over actual biometrics. As such, the Introduction first reviews 1) how to use LSHs to build proximity search, 2) describes the critical efficiency gaps, and then 3) presents the new cryptography and implementation [CHF22].

Proximity Search from LSHs A database is a list of biometrics $\mathcal{DB} = w_1, ..., w_\ell$ where each $w_i \in \{0,1\}^n$. The goal of a biometric database is given some w^* to find all values $w_i \in \mathcal{DB}$ that are similar enough to w^*. For the Hamming metric $\mathcal{D}$ and distance threshold t, the goal is to find all w_i such that $\mathcal{D}(w_i, w^*) \leq t$.[3] A LSH maps similar items to the same value more frequently than it maps far items to the same value. Let $\mathcal{H}$ be a family of LSHs then

$$\Pr_{\mathsf{LSH} \leftarrow \mathcal{H}}[\mathsf{LSH}(w_i) = \mathsf{LSH}(w^*) | w_i, w^* \text{ are near}] \geq 1 - p_1,$$
$$\Pr_{\mathsf{LSH} \leftarrow \mathcal{H}}[\mathsf{LSH}(w_j) = \mathsf{LSH}(w^*) | w_j, w^* \text{ are far}] \leq 1 - p_2.$$

where $p_1 < p_2$. Maps associate keys to a value and are used to build inverted indices. For a database of size ℓ, parameter $\beta \in \mathbb{Z}^+$, maps $\mathsf{M}_1, ..., \mathsf{M}_\beta$, and LSH family $\mathcal{H}$, one achieves proximity search as follows:

1. Sample β LSHs, $\mathsf{LSH}_1, ..., \mathsf{LSH}_\beta \leftarrow \mathcal{H}$.
2. For $j = 1, ..., \beta$, set $\mathsf{M}_j[v] = \{w_i | \mathsf{LSH}_j(w_i) = v\}$.
3. To search for value w^*, compute $\mathsf{LSH}_1(w^*), ..., \mathsf{LSH}_\beta(w^*)$ and retrieve $\cup_{j=1}^{\beta} \mathsf{M}_j[\mathsf{LSH}_j(w^*)]$.

Notice that the query is a disjunction.

If multiple records share the same LSH value our implementation concatenates the matching values. This allows us to handle a constant number of values associated to each key. This condition is satisfied for the accuracy regimes discussed in this work. Boldyreva and Tang [BT21] constructed a zero-leakage encrypted map scheme called an *oblivious map with encryption* or OMapE. Each clause is submitted to the relevant OMapE, the results are concatenated as in Step 3 above.

Proximity search on biometrics is tricky using the above construction. Due to their large noise, **biometrics require one to sample hundreds or thousands of LSHs** to achieve reasonable accuracy (see analysis in Section 2.1 and Section 6). At the same time, very **few of these LSHs will match anything** in the corresponding map. Most of the time the construction above performs heavy oblivious operations to hide the null value.

1.1 Our Design

Our design is a two-part approach:

1. Find a small number, δ, of LSHs values that exist in some map,
2. Query only those δ maps in a way that hides which δ maps are being queried.

For the above design to be successful, one needs to demonstrate:

Obliviousness Sec. 3 One can hide the queried δ maps,

Accuracy Sec. 6.1 High accuracy with $\delta < \beta$, and

Speed Sec. 6.2 The two-stage approach results in a faster overall system.

Below we present a more formal description of these two components. Our evaluation focuses on showing the above three properties.

[3] This functionality differs from 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 [KPT19a, KPT19b, LMWY20] and resulting systems [CCD$^+$20]. These attacks do not apply to our leakage profile.

Oblivious Membership Check An object to check which of the LSHs have matches. For an encrypted stored set X the *oblivious membership check* or OMC takes in a set W and returns $W' \subseteq W \cap X$ where $|W'| \leq \delta$. Our analysis finds a parameter δ that is unlikely to remove hurt accuracy. One can easily build OMC using private set intersection and pseudorandom permutations (there are many approaches), see discussion in Sections 2.2 and 3.3. We benchmark this design using VolePSI [RS21]; the resulting implementation is orders of magnitude faster than our oblivious map implementation.

Disjunctive Oblivious Map An object that directly searches for the disjunction of exactly δ items (if $|W'| < \delta$ we search for dummy values to search for exactly δ values). This object has the same functionality as an oblivious map that takes multiple clauses but the fact that all clauses are presented together is crucial for security. We call this object a DOMapE for *disjunctive oblivious map with encryption.* Our focus is on designing a DOMapE.

Building DOMapE Our focus is on the efficient design of **Disjunctive Oblivious Maps** for $\delta < \beta$. Recall Boldyreva and Tang [BT21] build β separate maps and search each one for an LSH clause. This approach does not work if one only queries δ clauses because it reveals which LSHs matched. We present the following contributions:

1. A DOMapE based on oblivious tree traversal building on the design principles of Wang et al. [WNL+14]. We show how to convert a DOMapE into proximity search.
2. A thorough parameter analysis on real, synthetic, and random data. As part of this analysis, we present a novel tool for generating larger synthetic iris datasets based on generative adversarial networks or GANs [CWD+18, GPAM+20].
3. A prototype implementation of proximity search and benchmarking on real, synthetic, and random datasets of up to 5 thousand irises.

Organization The rest of this paper is organized as follows: Section 2 introduces preliminaries, Section 3 presents our designs for DOMapE, Section 4 presents the datasets, Section 5 describes our implementation, Section 6 our evaluation methodology, and Section 7 concludes. Appendix A describes the architecture used to generate synthetic irises.

2 Preliminaries

Let λ be the security parameter throughout the paper. We use $\texttt{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to denote unspecified functions that are polynomial and negligible in λ, respectively. All definitions are indexed by λ but this indexing is omitted for notational clarity. For some $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \cdots, n\}$. Let $x \xleftarrow{\$} S$ denote sampling x uniformly at random from the finite set S. Hamming distance is defined as the distance between the bit vectors x and y of length n: $\mathcal{D}(x, y) = |\{i \mid x_i \neq y_i\}|$ and the fractional Hamming distance is $\mathcal{D}(x, y)/n$. For a map M, let $\mathsf{M.Keywords}$ output the set of all stored keywords. *Locality sensitive hashes* (LSH) frequently map similar values to the same hash.

Definition 1 (Locality-sensitive Hashing (LSH))**.** *Let $t \in \mathbb{N}$, $c > 0$ and $p_1, p_2 \in [0, 1]$. $\mathcal{H}$ defines a (t, ct, p_1, p_2)-sensitive hash family if for any $x, y \in \{0, 1\}^n$, we have:*

1. *If $\mathcal{D}(x, y) \leq t$ then $\Pr_{\mathsf{LSH} \in \mathcal{H}} [\mathsf{LSH}(x) = \mathsf{LSH}(y)] \geq p_1$ and*
2. *If $\mathcal{D}(x, y) \geq ct$ then $\Pr_{\mathsf{LSH} \in \mathcal{H}} [\mathsf{LSH}(x) = \mathsf{LSH}(y)] \leq p_2$,*

where $\mathcal{D}(x, y)$ denotes the Hamming distance between binary vectors x and y. For x, y if $\mathcal{D}(x, y) \leq t$ they are said to be near, if $\mathcal{D}(x, y) \geq ct$ they are said to be far.

Composing LSHs The error rates p_1, p_2 can be increased by randomly sampling several LSHs and checking that they all match. That is, taking the logical AND of LSHs. Similarly, p_1, p_2 can be decreased by randomly sampling several LSHs and checking that at least one of them matches. That is, the logical OR of LSHs. We use α to indicate the number of LSHs in a logical AND. We will separately search for each LSH in the logical OR, we use β for this parameter.

If one takes the logical AND of α LSHs the probability of them all matching is at least $p_1' \geq p_1^\alpha$ for near things and at most $p_2' \leq p_2^\alpha$ for far things. For the β OR of LSHs the bounds are $p_1' \geq 1 - (1 - p_1)^\beta$ and $p_2' \leq 1 - (1 - p_2)^\beta$. **We use the LSH that selects α bits of the input, a common approach for Hamming data.**

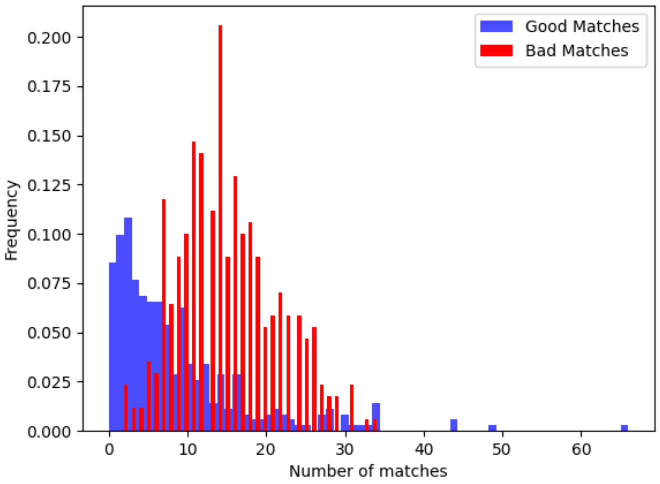


Figure 1: The number of maps returning a value when searching similar values with $\alpha = 15$, $\beta = 225$ trees and 356 records from the ND-0405 dataset.

2.1 The need for many LSHs in biometric proximity search

We focus on the iris biometric using the ND-0405 dataset [PSO$^+$09, BF16] which has an average distance $t/n \approx .21$ using a state-of-the-art feature extractor [AF19]. There are 712 different irises in the ND0405 dataset which is a superset of the NIST Iris Evaluation Challenge [PBF$^+$08]. Half of these records are used for training the used feature extractor [AF18, Ahm20, AF19, ACD$^+$22] which produces features of length 1024. The remaining 356 right irises are suitable for experimentation. Section 4 describes other datasets used in this work. Our discussion applies to other biometrics with substantive noise such as the face. See face noise histograms on a modern feature extractor in [DGXZ19, Figure 6].

Let w_i' be a noisy reading of w_i. When using w_i' as input to search, a true accept is when w_i is returned and the true accept rate (TAR) is the fraction of queries where this happens. The false accept rate (FAR) is the fraction of $\mathcal{DB} \setminus \{w_i\}$ that is returned on average. If one assumes that all readings of the same biometric have $p_1 = .79$ (corresponding to distance exactly $t/n = .21$) then achieving TAR= $p_1' = .95$ and a FAR of $p_2' = .01$ requires the number of LSHs $65 \leq \beta \leq 80$ for the minimum $\alpha = 13$. For a dataset of size 10^6 if one seeks at most 100 false accepts this requires $680 \leq \beta \leq 835$ at the minimum $\alpha = 23$. Even though the mean distance between readings of the same biometric is $t/n = .21$ there is substantial variance in this distance (see Figure 13), requiring β to be larger as we show in Table 1. Boldyreva and Tang [BT21] tested on datasets with $\beta \leq 10$.

Furthermore, most of the LSHs will return $\perp$. We call an LSH match *good* if it ensures the query results in a true accept and *bad* otherwise. For the ND-0405 dataseta histogram of the number of good and bad LSH matches is in Figure 1.[4] The average number of total LSH matches is 23.4. Boldyreva and Tang's design queries all β OMapE despite the fact that most will return no records.

2.2 Cryptographic Definitions

This work also relies on *oblivious RAM*, to achieve zero-leakage.

Definition 2 (Oblivious RAM [GMP16])**.** *An Oblivious RAM (ORAM) scheme is two protocols,* Setup *and* OracleAccess*:*

[4] This uses the following experiment:

1. Storage of a single feature extracted reading for the right eye for each of the 356 persons in the ND-0405 dataset. Sample $\beta = 225$ LSHs of size $\alpha = 15$.
2. Use the second stored template in the ND-0405 dataset to create a search corpus $w_1', ..., w_{356}'$.
3. Search for each record w_i'. Record the number of good and bad LSH matches.

$\underline{\mathsf{Real}_{\mathcal{A},q}(1^\lambda)}$:

1. $\mathcal{A}$ chooses Mem.
2. Run $(\sigma_0, \mathsf{EMem}_0) \leftarrow \mathsf{Setup}((1^\lambda, \mathsf{Mem}), \perp)$.
3. For $1 \le i \le q$:
   (a) $y_i \leftarrow \mathcal{A}(\mathsf{EMem}_{i-1})$.
   (b) Run $((v_i, \sigma_i), \mathsf{EMem}_i) \leftarrow \mathsf{OracleAccess}((\sigma_{i-1}, y_i), \mathsf{EMem}_{i-1})$.
   (c) Set ts_i, as the server's view of the above computation.
4. Output $(\mathsf{EMem}_q, \mathsf{ts}_1, \ldots, \mathsf{ts}_q)$.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}(1^\lambda)}$:

1. Output $(\mathsf{EMem}_q, \mathsf{ts}_1, \ldots, \mathsf{ts}_q) \leftarrow \mathsf{Sim}(q, |\mathsf{Mem}|, 1^\lambda)$.

Figure 2: Definition of ORAM security.

- $(\sigma, \mathsf{EMem}) \leftarrow \mathsf{Setup}((1^\lambda, \mathsf{Mem}), \perp)$*:* Setup *takes the security parameter, and an array* Mem *and outputs a secret state for the client* σ *and an encrypted memory object* EMem *for the server.*
- $((v, \sigma'), \mathsf{EMem}') \leftarrow \mathsf{OracleAccess}((\sigma, i), \mathsf{EMem})$*:* $\mathsf{OracleAccess}$ *is a protocol between the client and the server, where the client provides as input* σ *and an index* i*. The server provides as input* EMem*. The client receives back* v *and an updated* σ' *and the server receives an updated* EMem'*.*

Correctness *Consider the following correctness experiment:*

1. *An adversary* $\mathcal{A}$ *chooses memory* Mem*.*
2. *Consider* EMem_0 *and* σ_0 *generated from* $\mathsf{Setup}((1^\lambda, \mathsf{Mem}), \perp)$*.*
3. *For* $1 \le i \le q$*:*
   (a) Run $y_i \leftarrow \mathcal{A}(\mathsf{EMem}_{i-1})$*.*
   (b) Run $(v_i, \sigma_i, \mathsf{EMem}_i) \leftarrow \mathsf{OracleAccess}((\sigma_{i-1}, y_i), \mathsf{EMem}_{i-1})$*.*

The adversary wins if for some $i, v_i \neq \mathsf{Mem}[y_i]$*. The ORAM scheme is correct if the probability of* $\mathcal{A}$ *winning the game is* $\mathit{negl}(\lambda)$*.*

Security *An ORAM scheme is secure in the semi-honest model if for any PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that for any PPT distinguisher* D *we have*

$$\left|\Pr[D(\mathsf{Real}_{\mathcal{A},q}(1^\lambda)) = 1] - \Pr[D(\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}(1^\lambda)) = 1]\right| \le \mathit{negl}(\lambda)$$

with $\mathsf{Real}_{\mathcal{A},q}$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}$ *as described in Figure 2.*

The above is an adaptive simulation definition of ORAM, all of our proofs work naturally for the standard non-adaptive definition as well. This would yield a non-adaptive searchable encryption scheme, Definition 3 below is adaptive.

Definition 3 (Oblivious Searchable Encryption (OSE))**.** *Let* $\mathcal{DB} = (w_1, ..., w_\ell)$ *be a database where each* $w_i \in \{0,1\}^n$*. Let* $t \in \mathbb{N}$ *be a parameter. For an input* $y \in \{0,1\}^n$ *The algorithms* $\mathsf{OSE} = (\mathsf{Setup}, \mathsf{BuildIndex}, \mathsf{Search}_{\mathit{client}}, \mathsf{Search}_{\mathit{server}})$ *define an interactive searchable encryption scheme:*

- $(\mathsf{sk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$*,*
- $I_{\mathcal{DB}} = (I_{\mathit{server}}, I_{\mathit{client}}) \leftarrow \mathsf{BuildIndex}(\mathsf{sk}, \mathcal{DB})$*,*
- $((J, I'_{\mathit{client}}), I'_{\mathit{server}}) \leftarrow [\mathsf{Search}_{\mathit{client}}(\mathsf{pp}, \mathsf{sk}, y, I_{\mathit{client}}), \mathsf{Search}_{\mathit{server}}(\mathsf{pp}, I_{\mathit{server}})]$*.*

We require the scheme to satisfy the following:

$\underline{\mathsf{Real}_{\mathcal{A},q}(1^\lambda)\text{:}}$

1. Compute $(\mathsf{sk}, \mathsf{pp}) \leftarrow \mathsf{OSE.Setup}(1^\lambda)$.
2. $\mathcal{A}(\mathsf{pp})$ outputs $\mathcal{DB}$.
3. Compute $(I^1_{\mathsf{client}}, I^1_{\mathsf{server}}) \leftarrow \mathsf{OSE.BuildIndex}(\mathsf{sk}, \mathcal{DB})$ and give I^1_{server} to $\mathcal{A}$.
4. For $1 \leq j \leq q$:
   (a) $\mathcal{A}$ sends query $y^j \in (\mathcal{X} \times [1, \ell])^\nu$.
   (b) Run
   $$\left((J^j, I^{j+1}_{\mathsf{client}}), I^{j+1}_{\mathsf{server}}\right) \leftarrow \left[\mathsf{OSE.Search}_{\mathsf{client}}(\mathsf{pp}, \mathsf{sk}, y_j, I^j_{\mathsf{client}}), \mathsf{OSE.Search}_{\mathsf{server}}(\mathsf{pp}, I^j_{\mathsf{server}})\right].$$
   (c) Send search transcript ts^j and updated index $I^{j+1}_{\mathsf{server}}$ to $\mathcal{A}$.
5. Output $(\mathsf{ts}^1, ..., \mathsf{ts}^q, I^1_{\mathsf{server}}, ..., I^q_{\mathsf{server}})$.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}(1^\lambda)\text{:}}$

1. Compute $\mathsf{pp} \leftarrow \mathsf{Sim}(1^\lambda, q)$.
2. $\mathcal{A}(\mathsf{pp})$ outputs $\mathcal{DB}$.
3. Compute and output $(\mathsf{ts}^1, ..., \mathsf{ts}^q, I^1_{\mathsf{server}}, ..., I^q_{\mathsf{server}}) \leftarrow \mathsf{Sim}(\mathcal{L}_{\mathsf{BuildIndex}}(\mathcal{DB}))$.

Figure 3: Definition of $\mathsf{Exp}^{\mathsf{OSE}}_{\mathsf{ADA\text{-}SIM}}$.

Security: *Let* $q = \mathtt{poly}(\lambda)$ *and* $\mathcal{L}_{\mathsf{OSE}} = \{\mathcal{L}_{\mathsf{BuildIndex}}, \mathcal{L}_{\mathsf{Search}} = \perp\}$ *be the leakage profile describing the leakage of* OSE*'s algorithms. For any PPT adversary* $\mathcal{A}$*, there exists a simulator* Sim *such that for any PPT distinguisher* D *we have*

$$\left|\Pr[D(\mathsf{Real}_{\mathcal{A},q}(1^\lambda)) = 1] - \Pr[D(\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}(1^\lambda)) = 1]\right| \leq \textsf{\textit{negl}}(\lambda)$$

with $\mathsf{Real}_{\mathcal{A},q}$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}$ *as described in Figure 3.*

We additionally say that OSE is *approximately correct* if:

(ϵ, t)-Approximate Correctness: For all $\mathcal{DB}, y$ define

$$J_{\mathcal{DB},\mathtt{near},y} := \{w_i | \mathcal{D}(w_i, y) \leq t\}.$$

Let $q = \mathtt{poly}(\lambda)$ and $\epsilon > 0$. We say that OSE is ϵ-approximately correct if for all $\mathcal{DB}$ for all $y_1, ..., y_q$:

$$\Pr\left[J^j \supseteq J_{\mathcal{DB},\mathtt{near},y} \;\middle|\; \begin{array}{c} (\mathsf{sk},\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda) \\ I^1_{\mathcal{DB}} \leftarrow \mathsf{BuildIndex}(\mathsf{sk}, \mathcal{DB}) \\ (J^j, I^{j+1}_{\mathcal{DB}}) \leftarrow \left[\mathsf{Search}_{\mathsf{client}}(\mathsf{pp}, \mathsf{sk}, y_j, I^j_{\mathsf{client}}), \mathsf{Search}_{\mathsf{server}}(\mathsf{pp}, I^j_{\mathsf{server}})\right] \end{array}\right] \geq 1 - \epsilon.$$

In the above definition, we don't explicitly limit the number of false matches. In Section 3, we never show that our construction satisfies approximate correctness. Proving formal bounds requires many assumptions about the data. Instead, we evaluate approximate correctness including the number of false matches empirically in Section 6.

Definition 4 (Disjunctive Oblivious Map with Encryption)**.** *Let* $\lambda \in \mathbb{N}$ *be the security parameter, let* $\mathcal{X}$ *be a universe of keywords and let* $\mathsf{M} : \mathcal{X} \to \{0, 1\}^n$ *be a map. Let* $\beta, \nu, \delta \in \mathbb{N}, \delta \leq \nu \leq \beta$ *be parameters. A set of protocols* $\mathsf{DOMapE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Get}_{\mathit{client}}, \mathsf{Get}_{\mathit{server}})$ *has the following functionality:*

- $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, \beta, \nu, \delta)$.

$\underline{\mathsf{Real}_{\mathcal{A},q,\beta,\nu,\delta}(1^\lambda)}$:

1. Run $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{DOMapE.Setup}(1^\lambda, \beta, \delta)$.
2. $\mathcal{A}(\mathsf{pp})$ chooses maps $\mathsf{M}_1, ..., \mathsf{M}_\beta$. Compute $\mathsf{EM} \leftarrow \mathsf{DOMapE.Encrypt}(\mathsf{sk}, \mathsf{M}_1, ..., \mathsf{M}_\beta)$.
3. For $1 \leq i \leq q$:
   (a) $\mathcal{A}$ outputs query Y^i,
   (b) Run $(r^i, \mathsf{EM}^{i+1}) \leftarrow \left[\mathsf{DOMapE.Get_{client}}(\mathsf{pp}, \mathsf{sk}, Y^i), \mathsf{DOMapE.Get_{server}}(\mathsf{pp}, \mathsf{EM}^i)\right]$.
   (c) Set ts^i to be the transcript of the above computation.
4. Output $(\mathsf{ts}^1, \ldots, \mathsf{ts}^q, \mathsf{EM}^1, ..., \mathsf{EM}^q,)$.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\nu,\delta}(1^\lambda)}$:

Compute and output $(\mathsf{ts}^1, \ldots, \mathsf{ts}^q, \mathsf{EM}^1, ..., \mathsf{EM}^q) \leftarrow \mathsf{Sim}(q, \beta, \nu, \delta, \{\lceil \log |\mathsf{M}_i| \rceil\}_{i=1}^q, 1^\lambda)$.

Figure 4: Definition of DOMapE security.

- $\mathsf{EM} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathsf{M}_1, ..., \mathsf{M}_\beta)$*, takes the secret key* sk *and maps* $\mathsf{M}_1, ..., \mathsf{M}_\beta$ *as inputs and outputs an encrypted map* EM*.*
- $(r, \mathsf{EM}') \leftarrow [\mathsf{Get}_{\mathit{client}}(\mathsf{sk}, y = (x_1, k_1, ..., x_\nu, k_\nu)),\ \mathsf{Get}_{\mathit{server}}(\mathsf{pp}, \mathsf{EM})]$ *, an interactive protocol between a client and a server. The client takes the secret key* sk *and a query* $y \in (\mathcal{X} \times [1, \ell])^\nu$ *as inputs and the server takes the encrypted map* EM*. The client receives a response* $r \in (\{0,1\}^n \cup \perp)^\delta$ *and the server receives an updated encrypted map* EM'*.*

Security: *Let* λ *be a security parameter and let* $q, \beta, \nu, \delta = \mathtt{poly}(\lambda)$*. A* DOMapE *scheme is secure in the semi-honest model if for any PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that for any PPT distinguisher* D *we have*

$$\left|\Pr[D(\mathsf{Real}_{\mathcal{A},q,\beta,\nu,\delta}(1^\lambda)) = 1] - \Pr[D(\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\nu,\delta}(1^\lambda)) = 1]\right| \leq \textsf{\textit{negl}}(\lambda)$$

with $\mathsf{Real}_{\mathcal{A},q,\beta,\nu,\delta}$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\nu,\delta}$ *as described in Figure 4.*

Correctness: *Let* $\epsilon > 0$*,* $q, \beta, \delta, \nu = \mathtt{poly}(\lambda)$ *and* $\delta \leq \nu \leq \beta$*.* DOMapE *is* ϵ*-correct if for all* $(\{\mathsf{M}_i\}_{i=1}^\beta, \{y^j \in (\mathcal{X} \times [1, \ell])^\nu\}_{j=1}^q)$ *there exists a set* $\mathcal{I} \subseteq [\nu]$ *where* $|\mathcal{I}| \leq \delta$ *such that :*

$$\Pr\left[(\cup_i r_i^j) \setminus \emptyset \subseteq \cup_{i \in \mathcal{I}} \mathsf{M}_{k_i}\left[x_i^j\right] \;\middle|\; \begin{array}{c} (\mathsf{sk,pp}) \leftarrow \mathsf{Setup}(1^\lambda, \beta) \\ \mathsf{EM}^1 \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathsf{M}_1, ..., \mathsf{M}_\beta) \\ (r^j, \mathsf{EM}^{j+1}) \leftarrow \left[\mathsf{Get}_{\mathit{client}}(\mathsf{pp}, \mathsf{sk}, y^j), \right. \\ \left. \mathsf{Get}_{\mathit{server}}(\mathsf{pp}, \mathsf{EM}^j)\right] \end{array}\right] \geq 1 - \epsilon.$$

As described in the Introduction, our scheme follows a two stage approach: first find a list of candidate LSH matches, and then use an appropriate oblivious match to find the relevant records using the candidate LSH matches. We formalize this first stage as an *oblivious membership checking* or OMC object. The functionality of this object is to find the matches that exist (in the corresponding maps). There are many ways to implement this object including private set intersection (PSI), client storage, and full set retrieval (see Section 3.3). We are **not** offering constructions of OMC as a technical contribution. We benchmark separately using PSI, see discussion in Section 6. In our full implementation we use a local Bloom filter to simplify evaluation.

OMC only handles sets, that is, a collection of values *without repeats*. In our search system, these values will be LSH outputs. It is possible for two distinct LSHs to have the same output. To avoid this, we prepend the LSH id to each LSH output value. For LSH j, the corresponding values to use would then be $\{j \mid\mid \mathsf{LSH}_j(x)\}$.

Definition 5 ((β, δ, γ)-Oblivious Membership Checking Object)**.** *Let* $\beta, \delta \in \mathbb{N}$ *such that* $\delta \leq \beta$*. An oblivious membership checking object* OMC *is a set of two interactive algorithms between a client and server. Let* X *be a set of objects where* $|X| = \gamma$*.*

$\underline{\mathsf{Real}_{\mathcal{A},q,\beta,\delta}(1^\lambda)}$:

1. $\mathcal{A}$ receives 1^λ and outputs $X \subseteq \mathcal{X}$ where $|X| = \gamma$.
2. Compute $(EC, ES) \leftarrow \mathsf{OMC.Setup}(1^\lambda, X)$.
3. $\mathcal{A}$ adaptively makes q search queries, for $1 \leq j \leq q$:
   (a) $\mathcal{A}$ sends query set Y^j, abort if $|Y^j| \neq \beta$.
   (b) Run $(I_j, \perp) \leftarrow \mathsf{OMC.MemCheck}(ES, EC, Y^j)$.
   (c) Send the transcript ts_j of the check execution to $\mathcal{A}$.
4. Return $\mathsf{ts}^1, ..., \mathsf{ts}^q, ES$.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\delta}(1^\lambda)}$: Compute and output $\mathsf{ts}^1, ..., \mathsf{ts}^q, ES \leftarrow \mathsf{Sim}(1^\lambda, q, \gamma, \beta, \delta)$.

Figure 5: Definition of OMC security.

- $(EC, ES) \leftarrow \mathsf{OMC.Setup}(1^\lambda, X)$ *takes as input the set of objects to be stored X and returns EC and ES to the client and server, respectively.*
- $(I, \perp) \leftarrow \mathsf{OMC.MemCheck}(ES, EC, Y)$ *where $|Y| = \beta$. Outputs a set I to the client where $|I| \leq \delta$ and $I \subseteq X \cap Y$ and $\perp$ to the server.*

Correctness: *Correctness is one-sided. For $I \leftarrow \mathsf{OMC.MemCheck}(ES, EC, Y)$, for $(EC, ES) \leftarrow \mathsf{OMC.Setup}(1^\lambda, X)$:*

$$\Pr[i \in I \wedge i \notin X \cap Y] \leq \textsf{\textit{negl}}(\lambda).$$

Security: *Let parameters $\delta, \beta, q = \texttt{poly}(\lambda)$. OMC is secure if for any PPT adversary $\mathcal{A}$ there exists a simulator Sim such that for any PPT distinguisher D we have*

$$\left|\Pr[D(\mathsf{Real}_{\mathcal{A},q,\beta,\delta}(1^\lambda)) = 1] - \Pr[D(\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\delta}(1^\lambda)) = 1]\right| \leq \textsf{\textit{negl}}(\lambda)$$

with $\mathsf{Real}_{\mathcal{A},q,\beta,\delta}$ and $\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q,\beta,\delta}$ as described in Figure 5.

We defer discussion of constructions of OMC to Section 3.3

3 Oblivious Proximity Search for Biometrics

This section presents our technical designs, focusing on the design of DOMapE. We describe constructions of OMC in Section 3.3. The most relevant related work is by Boldyreva and Tang [BT21], whose construction is for the approximate k-nearest neighbors search problem. While Boldyreva and Tang discuss two ways of implementing OMapE, one using a tree and the other using a skip list [Pug90], we present only a tree based construction. Similar modifications can be made to the skip list construction. In this work, we consider static data. For static data, B-trees and skip lists are equivalent data structures [LN+96]. However, updates and the resulting performance differ.

3.1 Overview of DOMapE design

Recall the unprotected solution for proximity search from the Introduction (adapted from Boldyreva and Tang [BT21]):

1. Sample β LSHs, $\mathsf{LSH}_1, ..., \mathsf{LSH}_\beta \leftarrow \mathcal{H}$.
2. For $j = 1, ..., \beta$, set $\mathsf{M}_j[v] = \{w_i | \mathsf{LSH}_j(w_i) = v\}$.
3. To search w^*, compute $\mathsf{LSH}_1(w^*), ..., \mathsf{LSH}_\beta(w^*)$, and retrieve

$$\cup_{j=1}^{\beta} \mathsf{M}_j[\mathsf{LSH}_j(w^*)].$$

```
BuildIndex(M, $\mu$):
  1. Sort $\mathsf{M}_i$ using the comparator $\leq$.
     Let $\mathsf{Leaves} = (x_i, \mathsf{M}_i[x_i])$ be the sorted result.
  2. Pad Leaves to length $2^\mu$ with pairs $(\bot, \bot)$.
  3. Build balanced binary search tree Tree over the values of $x_i$ and for each internal node, attach pointers to
     its left and right child, LChild and RChild.
  4. Set $\mathsf{M}_i[x_i]$ as data associated with the leaf node with identifier $x_i$.
```

Figure 6: Build tree index algorithm

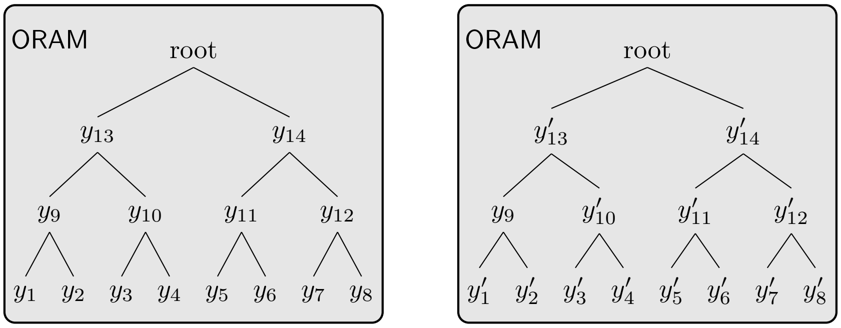


Figure 7: Basic access strategy of Boldyreva and Tang [BT21]. Each shaded region represents data stored together in a single oblivious RAM.

Boldyreva and Tang's design focuses on building M_j. The maps consists of $y_i, \{w_i\}$ pairs. The values placed into the map are sorted (lexographically) and used as nodes in a binary tree. Internal nodes are given the value of the minimum value in the right subtree and the location of the two children $\mathsf{LChild}, \mathsf{RChild}$. We show the design of this BuildIndex algorithm in Figure 6. Let $\mathsf{Tree}_1, ..., \mathsf{Tree}_\beta$ be the output of BuildIndex on maps $\mathsf{M}_1, ..., \mathsf{M}_\beta$ respectively. They then place each tree in an ORAM. Their construction fully traverses every tree Tree_i meaning that there is a constant number of accesses to each ORAM with every search. Let ν_i be the number of elements in M_i, define $\mu = \lceil \log \max_i \nu_i \rceil$, by padding each ORAM to length 2^μ each ORAM receives exactly μ accesses with each query ($\mu\beta$ across the β trees). The design is shown visually in Figure 7 with each shaded region representing a separate ORAM. Since the tree is fully traversed on each access one does not need to hide the level. As such one can use one ORAM per tree level. This is shown visually in Figure 8.

Our approach Recall that our goal is a two part construction: First one queries the OMC to find out which δ LSHs have matches. Then one queries the relevant δ M_i to find records. In this new design, one does not query every $\mathsf{M}_1, ..., \mathsf{M}_\beta$. As such, the set of queried Ms is potential leakage. We merge the ORAMs across maps to prevent this. However, we retain a separate ORAM for each level of the maps. This is shown visually in Figure 9. This means that each query now has δ accesses of each ORAM level. There are μ levels in total resulting in $\delta\mu$ ORAM accesses (in place of $\beta\mu$ accesses in Boldyreva and Tang [BT21]). Throughout our efficiency analysis, we view the number of ORAM accesses as our atomic unit.

3.2 Detailed design of DOMapE

3.2.1 $(\beta, \nu = \beta, \delta = \beta)$-DOMapE construction

First, we show how to build $(\beta, \nu = \beta, \delta = \beta)$-DOMapE from binary trees and ORAM.

Construction 1. *Let $\mathcal{X}$ and $\mathcal{M}$ be the domain and range of a map, such that elements in $\mathcal{X}$ are comparable with the $\leq$ operator. We define β maps* $\mathsf{M}_1, ..., \mathsf{M}_\beta$*. Let* ORAM = (ORAM.Setup, ORAM.OracleAccess) *be an oblivious RAM*

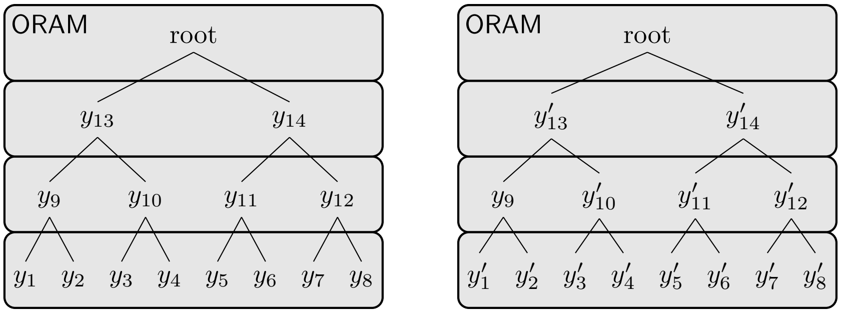

ORAM
root
y_{13} y_{14}
y_9 y_{10} y_{11} y_{12}
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8
ORAM
root
y'_{13} y'_{14}
y_9 y'_{10} y'_{11} y'_{12}
y'_1 y'_2 y'_3 y'_4 y'_5 y'_6 y'_7 y'_8

Figure 8: The first optimization to Boldyreva and Tang's construction, where each ORAM is applied per level.

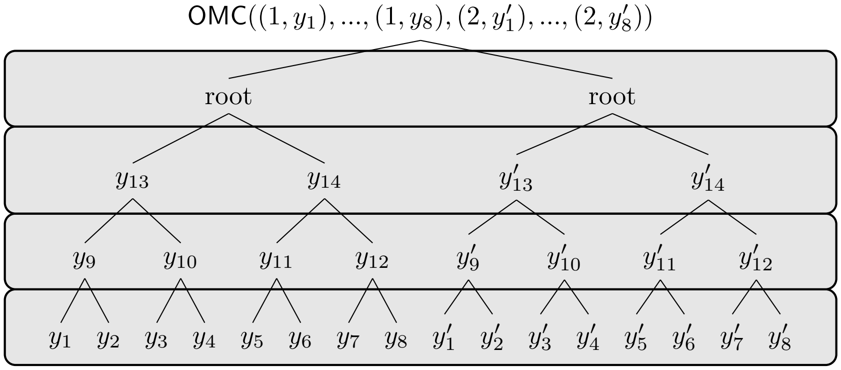

$\mathsf{OMC}((1,y_1),...,(1,y_8),(2,y'_1),...,(2,y'_8))$
root root
y_{13} y_{14} y'_{13} y'_{14}
y_9 y_{10} y_{11} y_{12} y'_9 y'_{10} y'_{11} y'_{12}
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y'_1 y'_2 y'_3 y'_4 y'_5 y'_6 y'_7 y'_8

Figure 9: Our design of DOMapE each level across binary trees is stored in a single ORAM.

as defined in Definition 2.2, and let ORAM_i *denote its instantiation for level* $1 \leq i \leq \mu$. *Consider the* DOMapE *construction shown in Figure 10.*

Theorem 1. *Construction 1 describes an* $(\beta, \nu = \beta, \delta = \beta)$-DOMapE *for* $\mu := \mathcal{L}_{\mathsf{BuildIndex}}(\mathsf{M}_1, ..., \mathsf{M}_\beta) = \max_i \lceil \log |\mathsf{M}_i| \rceil$.

Proof. We need to show that for every adversary $\mathcal{A}_{\mathsf{DOMapE}}$ there exists simulator $\mathsf{Sim}_{\mathsf{DOMapE}}$ for $(\beta, \nu = \beta, \delta = \beta)$-DOMapE such that

$$\mathsf{Real}_{\mathcal{A}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}(\lambda) \approx \mathsf{Ideal}_{\mathcal{A}, \mathsf{Sim}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}(\lambda)$$

where $\mathsf{Real}_{\mathcal{A}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$ and $\mathsf{Ideal}_{\mathcal{A}, \mathsf{Sim}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$ are defined as in Definition 4.

Let $\mathsf{Sim}_{\mathsf{ORAM}_j}$ denote the simulator for the j^{th} level ORAM, $1 \leq j \leq \mu$. We build $\mathsf{Sim}_{\mathsf{DOMapE}}$, the simulator for $(\beta, \nu = \beta, \delta = \beta)$-DOMapE, as follows:

1. Receive inputs $(q, \beta, \beta, \beta, \mu, 1^\lambda)$.
2. For $1 \leq i \leq q$:
   (a) For $1 \leq j \leq \mu$, run ORAM_j simulator $(\mathsf{EMem}^i_j, \mathsf{ts}^i_{\mathsf{ORAM},j,1}, \ldots, \mathsf{ts}^i_{\mathsf{ORAM},j,\beta}) \leftarrow \mathsf{Sim}_{\mathsf{ORAM}_j}(\beta, |\mathsf{Level}_j|, 1^\lambda)$.
   (b) Set $\mathsf{EM}^i = (\mathsf{EMem}^i_1, \cdots, \mathsf{EMem}^i_\mu)$ and $\mathsf{ts}^i = (\mathsf{ts}^i_{\mathsf{ORAM},1,1}, \cdots, \mathsf{ts}^i_{\mathsf{ORAM},\mu,\beta})$.
3. Return $(\mathsf{ts}^1, \cdots, \mathsf{ts}^q, \mathsf{EM}^1, \cdots, \mathsf{EM}^q)$.

We then use a hybrid argument where at each step, we replace an ORAM by its corresponding simulator. We obtain the following games

- *Game* 0: $\mathsf{ORAM}_1, \cdots, \mathsf{ORAM}_\mu$,
- *Game* j: $\mathsf{Sim}_{\mathsf{ORAM}_1}, \cdots, \mathsf{Sim}_{\mathsf{ORAM}_j}, \mathsf{ORAM}_{j+1}, \cdots, \mathsf{ORAM}_\mu$,

```
$\underline{\mathsf{DOMapE.Encrypt}(\mathsf{M}_1, \cdots, \mathsf{M}_\beta)}$:
  1. Let $\eta_i$ be the number of elements in $\mathsf{M}_i$, define $\mu = \lceil \log \max_i \eta_i \rceil$.
  2. For $i \in [\beta]$: set $\mathsf{Tree}_i \leftarrow \mathsf{BuildIndex}(\mathsf{M}_i, \mu)$.
  3. Output $(\mathsf{EMem}_1, ..., \mathsf{EMem}_\mu) \leftarrow \mathsf{ApplyORAM}(\mathsf{Tree}_1, ...., \mathsf{Tree}_\beta)$.
$\underline{\mathsf{ApplyORAM}(\mathsf{Tree}_1, ..., \mathsf{Tree}_\beta)}$:
  1. For $j \in [1, \mu]$:
     (a) $\mathsf{Level}_j = \perp$. For all $\mathsf{Tree}_{i \in [\beta]}$, $\mathsf{Level}_j = \mathsf{Level}_j || \mathsf{Level}(\mathsf{Tree}_i, j)$.
     (b) $\mathsf{EMem}_j = \mathsf{ORAM.Setup}(1^\lambda, \mathsf{Level}_j), \perp)$
  2. Return $\mathsf{EMem}_1, ..., \mathsf{EMem}_\mu$.
$\underline{\mathsf{Level}(\mathsf{Tree}, j)}$: Return all nodes at level $j$ in $\mathsf{Tree}$.
$\underline{\mathsf{DOMapE.Get}_{\mathsf{client}}(\mathsf{sk}, y \in (\mathcal{X} \times [1, \ell])^\nu)}$:
  1. Parse $y = (x_1, k_1, ..., x_\nu, k_\nu)$. Set $\mathsf{Nodes}_1 = ((k_1, 1), ..., (k_\nu, 1))$, $\mathsf{Res} = \perp$.
  2. For $j = [1, \mu - 1]$ and for $i$ in 1 to $\nu$:
     (a) Run $\mathsf{ORAM.OracleAccess}(\mathsf{Nodes}_j[i], \perp)$ with $\mathsf{server}$.
     (b) Let $x', \mathsf{LChild}, \mathsf{RChild}$ denote the result.
     (c) If $x' \leq x_i$, $\mathsf{Nodes}_{j+1} = \mathsf{Nodes}_{j+1} || (k_i, \mathsf{LChild})$.
     (d) Else $\mathsf{Nodes}_{j+1} = \mathsf{Nodes}_{j+1} || (k_i, \mathsf{RChild})$.
  3. For $i$ in 1 to $\nu$:
     (a) Initiate $\mathsf{ORAM.OracleAccess}(\mathsf{Nodes}_\mu[i], \perp)$ with server.
     (b) Let $x', \mathsf{M}[x']$ denote the result.
     (c) If $x' = x_i$, $\mathsf{Res} = \mathsf{Res} \cup \mathsf{M}[x']$.
  4. Return $\mathsf{Res}$.
```

Figure 10: $(\beta, \nu, \delta = \nu)$-DOMAPE Construction. The BuildIndex algorithm is shown in Figure 6.

- *Game* μ: $\mathsf{Sim}_{\mathsf{ORAM}_1}, \cdots, \mathsf{Sim}_{\mathsf{ORAM}_\mu}$,

with $0 < j < \mu$.

Note that Game 0 contains μ ORAM instantiations, which corresponds to the real world $\mathsf{Real}_{\mathcal{A}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$. Also note that Game μ, contains μ ORAM simulators, which is equivalent to $\mathsf{Sim}_{\mathsf{DOMapE}}$ and to the ideal world $\mathsf{Ideal}_{\mathcal{A}, \mathsf{Sim}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$. Then for each Game, we show indistinguishability with the previous one by relying on the security of the underlying ORAM_i.

Lemma 1. *For $1 \leq i \leq \mu$, Game i is indistinguishable from Game $i - 1$.*

Proof. By security of ORAM_i, we have $\mathsf{ORAM}_i \approx \mathsf{Sim}_{\mathsf{ORAM}_i}$. Since Game $i - 1$ and Game i only differ at index i, we conclude that these two games are indistinguishable. □

By applying Lemma 1 to each game, we obtain that Games 0 and μ are indistinguishable, which implies that $\mathsf{Real}_{\mathcal{A}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$ and $\mathsf{Ideal}_{\mathcal{A}, \mathsf{Sim}_{\mathsf{DOMapE}}, q, \beta, \beta, \beta}$ are also indistinguishable and concludes this proof. □

3.2.2 $(\beta, \nu = \beta, \delta)$-$\mathsf{DOMapE}$ construction

Recall that in Boldyreva and Tang's construction [BT21], each tree is in its own ORAM, and every tree is traversed during search. This works for their setting, which has a smaller number of trees. Our setting requires many more trees, traversing every one of them during search would dramatically decrease efficiency. Thus, we want to only traverse a fixed number of trees. To identify which trees to traverse we add an OMC layer on top of the tree-based construction. We then show that combining a (β, δ)-OMC scheme and a (β, δ, δ)-DOMapE yields a (β, β, δ)-DOMapE.

Construction 2. *Let* OMC *be an* (β, δ)*-oblivious membership check and let* DOMapE *be an* $(\beta, \nu = \delta, \delta)$-$\mathsf{DOMapE}$, *then we can build a* $(\beta, \nu = \beta, \delta)$-$\mathsf{DOMapE} = (\mathsf{Setup}^*, \mathsf{Encrypt}^*, \mathsf{Get}^*_{\mathit{client}}, \mathsf{Get}^*_{\mathit{server}})$ *as follows:*

- $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{Setup}^*(1^\lambda, \beta, \beta, \delta)$*: Run* $\mathsf{DOMapE.Setup}(1^\lambda, \beta, \delta, \delta)$ *and output* pp, sk.
- $\mathsf{EM}^* \leftarrow \mathsf{Encrypt}^*(\mathsf{sk}, \mathsf{M}_1, \cdots, \mathsf{M}_\beta)$*: Run*

$$ES \leftarrow \mathsf{OMC.Setup}(\mathsf{M}_1.\mathsf{Keywords}, \cdots, \mathsf{M}_\beta.\mathsf{Keywords})$$
$$\mathsf{EM} \leftarrow \mathsf{DOMapE.Encrypt}(\mathsf{sk}, \mathsf{M}_1, \cdots, \mathsf{M}_\beta).$$

  Output $\mathsf{EM}^* = (ES, \mathsf{EM})$.

- $(r, \mathsf{EM}'^*) \leftarrow [\mathsf{Get}^*_{\mathit{client}}(\mathsf{sk}, y),\ \mathsf{Get}^*_{\mathit{server}}(\mathsf{pp}, \mathsf{EM}^*)]$*:*
  1. *Run* $I \leftarrow \mathsf{OMC.MemCheck}(ES, y, \delta)$. *View* I *as a list.*
  2. *For* $j = 1$ *to* δ, *if* $|Y| < j$ *set* $z_j = (1, 1)$, *else set* $z_j = (y[i], I[i])$.
  3. *Run* $(r, \mathsf{EM}') \leftarrow [\mathsf{DOMapE.Get}_{\mathit{client}}(\mathsf{sk}, z),\ \mathsf{DOMapE.Get}_{\mathit{server}}(\mathsf{pp}, \mathsf{EM})]$.
  4. *Output* r *and* $\mathsf{EM}'^* = (ES, \mathsf{EM}')$.

Theorem 2. *Let* OMC *be a* (β, δ)*-oblivious membership check and* DOMapE *be an* $(\beta, \nu = \delta, \delta)$-$\mathsf{DOMapE}$ *scheme, then Construction 2 describes a* $(\beta, \nu = \beta, \delta)$-$\mathsf{DOMapE}$.

Proof. Let $\mathsf{Sim}_{(\beta,\delta,\delta)\text{-}\mathsf{DOMapE}}$ be the simulator for (β, δ, δ)-DOMapE. This simulator will run as follows:

1. Upon input $(q, \beta, \nu = \beta, \delta, \mu, 1^\lambda)$, run the simulator for the (β, δ)-oblivious membership check, $(ES, \mathsf{ts}^1_{\mathsf{OMC}}, \cdots, \mathsf{ts}^q_{\mathsf{OMC}}) \leftarrow \mathsf{Sim}_{(\beta,\delta)\text{-}\mathsf{OMC}}(|\mathsf{M}_1|, \delta, \beta, q)$.
2. Run $(\mathsf{ts}^1_{\mathsf{DOMapE}}, \cdots, \mathsf{ts}^q_{\mathsf{DOMapE}}, \mathsf{EM}^1, \cdots, \mathsf{EM}^q) \leftarrow \mathsf{Sim}_{(\beta,\delta,\delta)\text{-}\mathsf{DOMapE}}(q, \beta, \delta, \delta, \mu, 1^\lambda)$.
3. For $1 \le i \le q$, set $\mathsf{EM}^{i,*} = (ES, \mathsf{EM}^i)$ and $\mathsf{ts}^*_i = (\mathsf{ts}_{\mathsf{OMC},i}, \mathsf{ts}_{\mathsf{DOMapE},i})$.
4. Output transcripts $\mathsf{ts}^{1,*}, \cdots, \mathsf{ts}^{q,*}$ and encrypted maps $\mathsf{EM}^{1,*}, \cdots, \mathsf{EM}^{q,*}$.

We use a hybrid argument to show security of $(\beta, \nu = \beta, \delta)$-$\mathsf{DOMapE}$. Consider the following games:

1. $\mathsf{Real}_{(\beta,\nu=\beta,\delta)\text{-}\mathsf{DOMapE}}$,
2. $\mathsf{Sim}^*_{(\beta,\nu=\beta,\delta)\text{-}\mathsf{DOMapE}}$, which runs $\mathsf{Sim}_{(\beta,\delta)\text{-}\mathsf{OMC}}$ and $\mathsf{Real}_{(\beta,\nu=\delta,\delta)\text{-}\mathsf{DOMapE}}$,
3. $\mathsf{Sim}_{(\beta,\nu=\beta,\delta)\text{-}\mathsf{DOMapE}}$, which runs $\mathsf{Sim}_{(\beta,\delta)\text{-}\mathsf{OMC}}$ and $\mathsf{Sim}_{(\beta,\nu=\delta,\delta)\text{-}\mathsf{DOMapE}}$ as described above.

We want to show that game 1 is indistinguishable from game 3.

Lemma 2. *Game 1 and Game 2 are indistinguishable.*

Proof. Between games 1 and 2, the only difference is $\mathsf{Sim}^*_{(\beta,\nu=\beta,\delta)\text{-}\mathsf{DOMapE}}$'s use of $\mathsf{Sim}_{\mathsf{OMC}}$ instead of $\mathsf{Real}_{\mathsf{OMC}}$. Then by security of the OMC scheme, game 1 and 2 are indistinguishable. □

Lemma 3. *Game 2 and Game 3 are indistinguishable.*

Proof. Between games 2 and 3, the only difference is $\mathsf{Sim}_{(\beta,\nu=\beta,\delta)\text{-}\mathsf{DOMapE}}$'s use of $\mathsf{Sim}_{(\beta,\nu=\delta,\delta)\text{-}\mathsf{DOMapE}}$ instead of $\mathsf{Real}_{(\beta,\nu=\delta,\delta)\text{-}\mathsf{DOMapE}}$. Then by security of the $(\beta, \nu = \delta, \delta)$-$\mathsf{DOMapE}$ scheme, game 2 and 3 are indistinguishable. □

Combining lemmas 2 and 3, we obtain that games 1 and 3 are indistinguishable, which conclude our proof. □

$(\beta, \nu = \beta, \delta)$-$\mathsf{DOMapE}$ can implement oblivious searchable encryption.

Construction 3. *Let* $\mathsf{DOMapE} = (\mathsf{DOMapE.Setup}, \mathsf{DOMapE.Encrypt}, \mathsf{DOMapE.Get}_{\mathit{client}}, \mathsf{DOMapE.Get}_{\mathit{server}})$ *be a* $(\beta, \nu = \beta, \delta)$*-disjunctive oblivious map with encryption, let* $\mathcal{H}$ *be a family of functions mapping records to arbitrary values. For a* $\mathcal{DB} = (w_1, ..., w_\ell)$*, define* $\mathsf{OSE} = (\mathsf{OSE.Setup}, \mathsf{OSE.BuildIndex}, \mathsf{OSE.Search}_{\mathit{client}}, \mathsf{OSE.Search}_{\mathit{server}})$ *as*

- $\mathsf{OSE.Setup}(1^\lambda, \beta, \delta)$*:*
  1. *Run* $\mathsf{LSH}_1, ..., \mathsf{LSH}_\beta \leftarrow \mathcal{H}(1^\lambda)$.
  2. *Run* $(\mathsf{pp}_{\mathsf{DOMapE}}, \mathsf{sk}_{\mathsf{DOMapE}}) \leftarrow \mathsf{DOMapE.Setup}(1^\lambda)$.
  3. *Output* $\mathsf{pp}_{\mathsf{OSE}} = \mathsf{pp}_{\mathsf{DOMapE}}, \mathsf{sk}_{\mathsf{OSE}} = (\mathsf{sk}_{\mathsf{DOMapE}}, \mathsf{LSH}_1, ..., \mathsf{LSH}_\beta)$.
- $\mathsf{OSE.BuildIndex}(\mathsf{sk} = (\mathsf{sk}_{\mathsf{DOMapE}}, \mathsf{LSH}_1, ..., \mathsf{LSH}_\beta), w_1, ..., w_\ell)$*:*
  1. *For* $1 \leq i \leq \beta$*:*
     (a) *Initialize map* M_i.
     (b) *For* $1 \leq j \leq \ell$ *set* $\mathsf{M}_i[\mathsf{keyword}] = \{w_j | \mathsf{LSH}_i(w_j) = \mathsf{keyword}\}$.
     (c) *Add dummy values to* M_i *until it is of size* ℓ.
  2. *Set* $I_{\mathit{client},\mathsf{OSE}} = \perp, I_{\mathit{server},\mathsf{OSE}} = \mathsf{DOMapE.Encrypt}(\mathsf{sk}, \mathsf{M}_1, ..., \mathsf{M}_\beta)$.
- $\mathsf{OSE.Search}_{\mathit{client}}(\mathsf{sk} = (\mathsf{sk}_{\mathsf{DOMapE}}, \mathsf{LSH}_1, ..., \mathsf{LSH}_\beta), y)$*:*
  1. *Compute* $r \leftarrow (\mathsf{DOMapE.Get}_{\mathit{client}}(\mathsf{sk}_{\mathsf{DOMapE}}, \mathsf{LSH}_1(y), ..., \mathsf{LSH}_\beta(y)))$.
  2. *Output* $(\cup_{i=1}^{\delta} r_i) \setminus \perp$.
- $\mathsf{OSE.Search}_{\mathit{server}}(\mathsf{pp}) = \mathsf{DOMapE.Get}_{\mathit{server}}(\mathsf{pp})$.
- $((J, I'_{\mathit{client}}), I'_{\mathit{server}}) \leftarrow [\mathsf{Search}_{\mathit{client}}(\mathsf{pp}, \mathsf{sk}, y, I_{\mathit{client}}), \mathsf{Search}_{\mathit{server}}(\mathsf{pp}, I_{\mathit{server}})]$.

Theorem 3. *Let* $\mathsf{DOMapE} = (\mathsf{DOMapE.Setup}, \mathsf{DOMapE.Encrypt}, \mathsf{DOMapE.Get}_{\mathit{client}}, \mathsf{DOMapE.Get}_{\mathit{server}})$ *be a* $(\beta, \nu = \beta, \delta)$*-disjunctive oblivious map with encryption then Construction 3 is an oblivious searchable encryption scheme with leakage* $\mathcal{L}_{\mathsf{BuildIndex}}(\mathsf{M}_1, ...\mathsf{M}_\beta) = \lceil \log \ell \rceil = \max_{i=1}^{\beta}\{\lceil \log |\mathsf{M}_i| \rceil\}$.

Proof. The simulator for $(\beta, \nu = \beta, \delta)$-disjunctive oblivious map with encryption $\mathsf{Sim}_{\mathsf{DOMapE}}$ is a valid simulator. □

Theorem 3 does not handle correctness. Since there is an overlap between the histograms for real data in Figure 13 one cannot make strong correctness claims. We evaluate correctness empirically using our implementation in Section 6.

3.3 Oblivious membership check constructions

In this section, we discuss with more details a few options to implement OMC. We briefly cover approaches based on Bloom filter lookups. In Section 3.3.2, we describe how to build OMC from private set intersection. This is the tool that we use for microbenchmarks. In our implementation, we use a Bloom filter to emulate an OMC.

3.3.1 Oblivious Bloom Filter Lookups

The client's set can be stored in a Bloom filter [Blo70] which is then stored on the server in an ORAM. The client will request the relevant bits from the ORAM. This prevents the client from having to store the entire Bloom Filter on their side, but requires the client to request multiple ORAM accesses to query the relevant bits.

BlindSEER [PKV+14,FVK+15] built a tree of encrypted Bloom filters for general Boolean search. Search of each node uses Garbled circuits to decide whether to proceed to children. One can use a single level of their tree as an OMC as long as only the client learns the response. This requires some modification as their system was optimized for circuits that output a bit, we would need the set of matching locations. Their system was evaluated on datasets with 10^8 records [FMC+15].

$\underline{\mathsf{Real}_{\mathcal{A},n_X,n_Y}(1^\lambda)\text{:}}$

1. $\mathcal{A}$ receives 1^λ and outputs sets $X \subset \mathcal{X}$ and $Y \subseteq \mathcal{X}$.
2. If $|X| > n_X$ or $|Y| > n_Y$, return $\perp$.
3. Output transcript ts of $(Z, \perp) \leftarrow [\mathsf{PSI}_{\mathsf{client}}(X), \mathsf{PSI}_{\mathsf{server}}(Y)]$.

$\underline{\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},n_X,n_Y}(1^\lambda)\text{:}}$

1. $\mathcal{A}$ receives 1^λ and outputs sets $X \subset \mathcal{X}$ and $Y \subseteq \mathcal{X}$.
2. Output transcript $\mathsf{ts} \leftarrow \mathsf{Sim}(1^\lambda, |X|, |Y|)$.

Figure 11: Definition of PSI security.

3.3.2 Building OMC from PSI and pseudorandom permutations

Private set intersection (PSI) [FNP04] is a form of secure multi-party computation (MPC) where a client and server hold sets Y and X respectively. They run an interactive computation, at the end, the client learns $X \cap Y$. No other information is leaked. Current implementations of PSI depend on one of two tools: oblivious polynomial evaluations (OPE) and oblivious pseudorandom functions (OPRF).[5]

We show how to build OMC from honest-but-curious PSI as follows:

1. At initialization the client applies a pseudorandom permutation to the elements of set X.
2. The client sends the set of permuted elements to the server.
3. Later when the client has a set Y, the client applies the pseudorandom permutation to each element of Y, and uses the resulting values as their set for the PSI protocol.

Recall in the definition of OMC the simulator learns the size of both sets X, Y. Assuming an ideal PSI, in the above protocol, only the size of X is leaked to the server. We also note in our setting the size of Y is a global parameter while the size of X depends on the dataset size (which is leaked to the server).

The rest of this subsection is dedicated to formalizing the above construction. First, let us re-state the PSI definition used for VOLE-PSI [RS21, Figure 5].

Definition 6 (Private Set Intersection (PSI))**.** *Let $\mathcal{X}$ denote a set. Let the client hold a set $X \subset \mathcal{X}$ and the server hold a set $Y \subseteq \mathcal{X}$. Consider*

$$(Z, \perp) \leftarrow [\mathsf{PSI}_{\mathsf{client}}(X), \mathsf{PSI}_{\mathsf{server}}(Y)]\,,$$

the PSI protocol between the client and the server. At the end, the client learns Z and the server learns nothing.

Correctness: *Correctness is determined by the following probability statement,*

$$\Pr\left[Z \neq X \cap Y \mid (Z, \perp) \leftarrow [\mathsf{PSI}_{\mathsf{client}}(X), \mathsf{PSI}_{\mathsf{server}}(Y)]\right] \leq \textsf{negl}(\lambda).$$

Security: PSI *is secure if for any PPT adversary $\mathcal{A}$, there exists a simulator* Sim*, such that the distributions* $\mathsf{Real}_{\mathcal{A},q}$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathsf{Sim},q}$*, described in Figure 11, are computationally indistinguishable* [6]*.*

Remark *We consider PSI where both X and Y are constant size sets so we ignore the case when a party provides too large a set.*

Our construction also requires a definition of pseudorandom permutations.

[5] Cristofaro et al. [CT09] construct efficient PSI procotols, under the restriction that the server can do some precomputation or the client is weak. Dong et al. [DCW13] present an efficient PSI protocol based on a variant of Bloom Filters called *garbled Bloom Filters*. Dachman-Soled et al. [DSMRY09] present an efficient PSI protocol utilizing secret sharing and Reed-Soloman codes. Malicious secure implementations of PSI utilizing OPE also depend on zero-knowledge proofs to prevent parties from deviating from the protocol.

[6] [RS21] uses a different security definition for PSI, their definition implies ours.

```
• OMC.Setup($1^\lambda, X$):
    1. Client randomly samples a PRP key $\mathsf{sk} \xleftarrow{\$} \{0,1\}^\lambda$.
    2. Initialize $ES = \emptyset$.
    3. For each $x_i \in X$, $ES = ES \cup F(\mathsf{sk}, x_i)$.
    4. Client sends $ES$ to server.
    5. Client keeps $EC = \mathsf{sk}$.
• OMC.MemCheck($ES, EC, Y$):
    1. For each $y_j \in Y$, client computes permutation $F(\mathsf{sk}, y_j)$.
    2. Client initializes a map $\mathsf{M}$ and inserts $\mathsf{M}[F(\mathsf{sk}, y_j)] = y_j$.
    3. Client sets $EQ = \{F(\mathsf{sk}, y_j)\}$.
    4. Run $(EQ \cap ES, \perp) \leftarrow [\mathsf{PSI}_{\mathsf{client}}(EQ), \mathsf{PSI}_{\mathsf{server}}(ES)]$.
    5. At client
        (a) Initialize $\mathtt{Res} = \emptyset$. For each $c \in EQ \cap ES$ set $\mathtt{Res} = \mathtt{Res} \cup \mathsf{M}[c]$.
        (b) If $|\mathtt{Res}| > \delta$ set $\mathtt{Res}$ to be $\delta$ random elements of $\mathtt{Res}$.
        (c) Output $\mathtt{Res}$.
```

Figure 12: Construction of OMC from PSI and PRP.

Definition 7 (Pseudorandom Permutation (PRP))**.** *Let $F : \{0,1\}^\lambda \times \mathcal{X} \to \mathcal{X}$ be an efficient keyed permutation and $\mathcal{F}_n$ denote the set of all permutations on $\mathcal{X}$. F is a pseudorandom permutation if for any PPT adversary $\mathcal{A}$,*

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{A}^{F_k(\cdot)}(1^\lambda) = 1 \right] - \Pr_{f \xleftarrow{\$} \mathcal{F}_n} \left[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1 \right] \right| \leq \textsf{\textit{negl}}(\lambda)$$

Construction 4 (OMC from PSI and PRP)**.** *Let* PSI *denote a PSI scheme and $F : \{0,1\}^\lambda \times \mathcal{X} \to \mathcal{X}$ be a pseudorandom permutation. Let X and Y be sets, such that $X \subseteq \mathcal{X}$ and $Y \subseteq \mathcal{X}$. Then we build* OMC *as in Figure 12.*

Theorem 4. *Let* PSI *be a secure private set intersection scheme and F be a pseudorandom permutation, then Construction 4 describes a secure (β, δ)-OMC.*

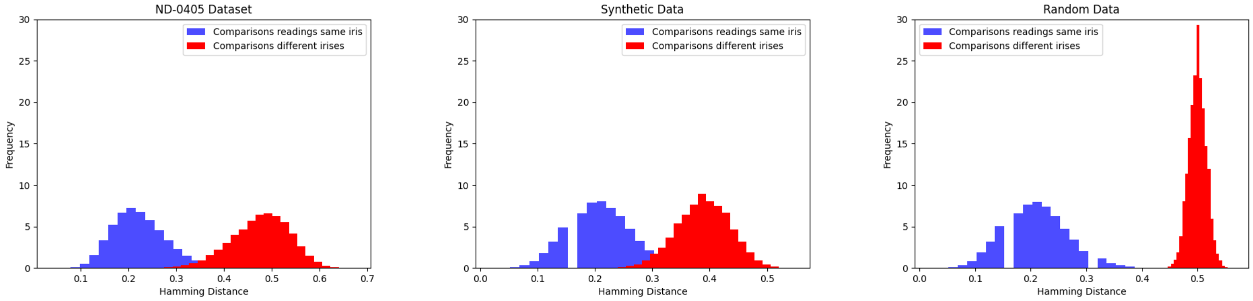
Proof. Correctness is straightforward and follows from correctness of the PSI and PRP schemes. Security follows from the security of the PRP (values seen by the server are indistinguishable from random) and the security of the PSI scheme (server learns nothing about EQ and $EQ \cap ES$). Formally, we build the simulator $\mathsf{Sim}_{\mathsf{OMC}}$ as follows:

1. Receive inputs 1^λ, number of queries q and server's set size γ.
2. For $1 \leq i \leq \gamma$, sample a random value $x_i \xleftarrow{\$} \mathcal{X}$.
3. Define the server set as $ES = \{x_i\}_{i=1}^{\gamma}$.
4. For $1 \leq k \leq q$: run the PSI simulator $\mathsf{ts}^k \leftarrow \mathsf{Sim}_{\mathsf{PSI}}(1^\lambda, \gamma, \beta)$.
5. Output ES and $\mathsf{ts}^1, ..., \mathsf{ts}^q$.

We then use a hybrid argument to show security of OMC. Consider the following games:

1. $\mathsf{Real}_{\mathsf{OMC}}$,
2. Replace the PRP F by a random permutation $f \xleftarrow{\$} \mathcal{F}_n$,
3. $\mathsf{Sim}_{\mathsf{OMC}}$.

Games 1 and 2 are indistinguishable by the security of the PRP scheme, and games 2 and 3 are indistinguishable by the security of the PSI scheme. We note that sampling a random output value is equivalent to sampling a random input value as inputs are guaranteed to be a set. □



(a) Histogram of comparisons for ND0405 dataset.

(b) Histogram of comparisons for synthetic dataset.

(c) Histogram of comparisons for random data.

Figure 13: Histograms of Hamming distance between readings of the same iris (in blue) and different irises (in red). Different irises are stored in the database and queries are drawn from a different reading of an iris in the database. The gaps in the synthetic and random blue histograms are caused by the query generation technique used (see paragraph on random and synthetic queries generation).

4 Datasets

We test and evaluate our implementation on three datasets:

ND-0405 dataset This dataset [PSO$^+$09, BF16] is a superset of the NIST Iris Evaluation Challenge [PBF$^+$08]. It consists of the readings of left and right irises from 356 individuals, each iris having at least 4 distinct readings. We use state-of-the-art feature extractors called ThirdEye [AF19] that is available at `https://github.com/sohaib50k/` to obtain 1024 bits feature vectors from the original iris readings. Since the left irises were used to train the feature extractor, we use the right ones for testing and evaluation. The first reading of each right iris will be the stored value and queries will be drawn from the remaining readings. The Hamming distance distributions for same and different irises can be seen on Figure 13a.

Synthetic dataset Available irises datasets are of limited size, often no more than a few hundreds records (356 individuals for ND-0405). However, real world systems would probably require datasets in the thousands to millions individuals, depending on the application. It is thus desirable to be able to run our systems on larger datasets. Our solution is to generate *synthetic* irises templates, that mimic actual ones. As can be seen in Figure 13b, synthetic data same and different distributions are similar to the ND-0405 ones. The details on synthetic data generation can be found in Appendix A. The high level approach is a generative adversarial network (GAN) [GPAM$^+$20] as in prior approaches on synthetic iris generation [AF20].

Random dataset This dataset is made from randomly generated 1024 bits vectors. As such, the Hamming distance between any two of those vectors should be close to 0.5 with a much smaller variance than other datasets. This is visible in the red histogram from Figure 13c.

Random and synthetic queries generation Contrary to the ND0405 dataset [BF16], the random and synthetic datasets do not include queries. To make comparison more natural, we want to sample queries from a distribution that resembles the one for ND-0405.

To generate queries we use the common observation that like irises comparisons have a distribution close to a binomial across different feature extractors [Dau09, Dau05, SSF19]. From Figure 13a, we extract: the mean $\mu = 0.21$ and the standard deviation, $\sigma = 0.056$. This yields a distribution $B(n, \mu)/n$ binomial distribution for $n = 53$. This is because for $B(n, \mu)$ it is true that $\sigma^2 = \mu(1 - \mu)n$. Thus, by linearity of expectation for $B(n, \mu)/n$ it is true that $\sigma^2 = \mu(1 - \mu)/n$, thus one can compute $n = \lceil \mu(1 - \mu)/\sigma^2 \rceil = \lceil 52.9 \rceil = 53$.

We can then generate queries for the random and synthetic datasets as follows:

1. First we generate a binomial distribution using the mean and standard deviation of the same iris distribution for the ND-0405.

2. For each feature vector in the dataset, we create a corresponding query by sampling an error fraction from the $\texttt{frac} \leftarrow B(53, .21)/53$.

3. We flipping the corresponding number of error bits in the original feature vector, that is, $\texttt{frac} * 1024$.

Using this technique, we obtain the same iris distributions (in blue) for synthetic and random data shown in Figures 13b and 13c. Since there are only 54 possible outcomes for a fraction of error bits, this leads to discontinuities in the histograms presented in Figures 13b and 13c.

5 Implementation

We present a full open-source implementation of our algorithms including the LSH parameter finding, tree building, and oblivious search [CHF22]. This implementation is in Python 3.9 and uses the PathORAM [SDS+18] module [Hac18]. Our experiments use a simple Bloom filter cache on the client as an OMC to focus on the performance of the developed DOMapE. We evaluate a OMC candidate based on PSI at the end of Section 6. Our implementation supports two main conclusions:

1. One can set a $\delta < \beta$ size of the query to DOMapE that supports a high true accept rate. In our implementation we set δ to be 1 more than the observed number of bad matches on 95% of queries across the dataset. See Table 1 for a comparison of true accept rate for the setting when $\delta = \beta$ and when $\delta < \beta$. In all analyzed parameters $\delta/\beta < .06$. We note that δ is higher for real and synthetic data than random data due in large part to the much larger variance of distances between readings of different irises. This greatly increases the number of bad matches.

2. We show that for δ parameters that produced a reasonable true accept rate, one can achieve search with reasonable performance. While setup takes several hours, search completes in at most a couple of minutes on all tested parameters. For these parameters we execute at most 1000 ORAM accesses and the average time for an ORAM access is approaching 1 second indicating a more optimized implementation would ensure that network delay remains the bottleneck.

Our cryptographic implementation is not parallel due to the use of a serial ORAM module (building the trees is done in parallel.) We see two mechanisms for achieving better performance as datasets grow:

1. Use ORAM that supports sending multiple queries in parallel (this is a weaker object than a parallel ORAM [WST12, BCP16, CLT16]).

2. Group LSHs into groups that are placed into a single RAM and analyze the required δ for each group. As $\delta/\beta \approx .06$ one can use standard concentration bounds to argue about δ for each group as long as the data and queries are independent of the LSHs.

Dataset Modifications Our construction does not allow for insertion after the initial building of the tree. However, nothing about our techniques prohibits rebuilding and rebalancing methods that are presented in the original description of an oblivious AVL tree by Wang et al. [WNL+14].

6 Evaluation

Evaluation is split into two parts: 1) parameter analysis and accuracy, and 2) efficiency of the resulting cryptographic construction. Our parameter analysis focuses on the TAR and number of matches. Our efficiency analysis focuses on network roundtrips, storage, and single-threaded computation time.

								TAR		
Dataset	Dataset				Matches			(β, β, β)		(β, β, δ)
size	type	α	β	# FA	Avg bad	Max bad	Avg good	DOMapE	δ	DOMapE
356	random	15	630	7.5	7.2	16	32.1	0.98	13	0.94
356	ND	18	850	14.4	16.8	55	21.6	0.95	37	0.89
356	synthetic	18	850	15.5	12.2	37	22.6	0.96	26	0.92
1000	random	18	850	3.6	3.5	11	21.2	0.96	8	0.96
1000	synthetic	19	1000	3.7	22.7	82	20.2	0.95	47	0.96
2500	random	19	1000	5.7	5.6	13	25	0.94	11	0.89
2500	synthetic	21	1200	35.3	24	70	20.7	0.92	56	0.82
5000	random	20	1200	6.9	6.7	14	21.7	0.92	12	0.87
5000	synthetic	22	1300	50.7	31	112	19.7	0.91	72	1.0

Table 1: TAR/FAR and the number of matches for random, ND0405, and synthetic datasets of different sizes.

6.1 Accuracy - Parameter analysis

We first discuss we set parameters. Datasets were discussed in Section 4. Each experiment is conducted on each dataset. Recall the relevant parameters:

- The number of concatenated LSHs α in a logical AND (an extended LSH),
- The number of maps and number of LSHs, β,
- The maximum number of matching maps of the DOMapE, δ. Recall one uses the OMC to find LSH matches and then obliviously looks up exactly δ maps, δ is critical for efficiency.

Finding α, β, δ The first part of the experiment consisted in a brute force search across α, β and measuring the TAR and number of bad matches. This search did not involve any cryptographic protections, records were simply stored in a standard map. Selected parameters had TAR of at least 90%. The average number of bad matches was at most 10 for random data and at most 50 for ND and synthetic data. Once α, β were selected we recorded the histogram of bad matches and set δ to be one more than the 95 percentile of this histogram.

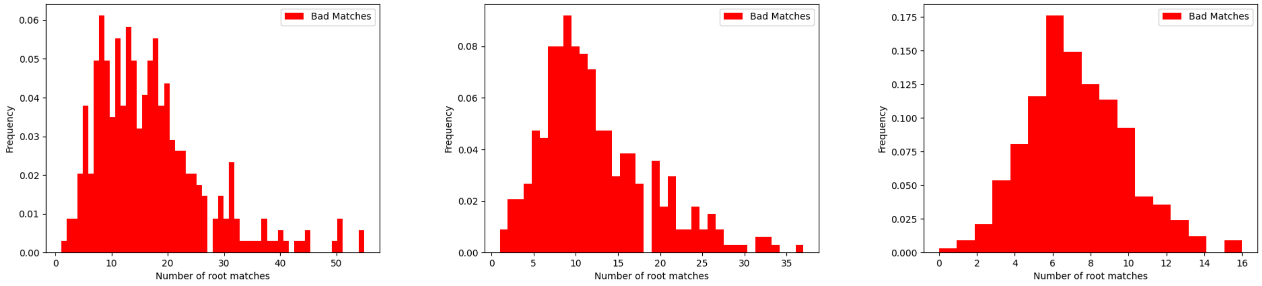
Measuring accuracy We then measured accuracy for both a $(\beta, \nu = \beta, \delta = \beta) -$ DOMapE and a $(\beta, \nu = \beta, \delta) -$ DOMapE. That is, setting δ to restrict the number of LSH values provided to the oblivious map to be δ. For these tests, we only measure the TAR to understand the impact of restricting the number of searched values on accuracy.

Discussion Even with random data, β is substantially higher than predicted in Section 2.1. In proximity search, one wishes to capture most of the tail of the comparisons between different readings of the same iris (the blue distribution in Figure 13). For example, for distance $t = .21n$ and a TAR of .01, Section 2.1 proposed $\beta = 65$ and $\alpha = 13$. However, our experimental results show that even for random data, to sustain high TAR and low FAR, we require $\beta = 630$ and $\alpha = 15$ trees. These parameters increase further on the ND and Synthetic datasets.

Parameters vary widely between random and synthetic data. This is due to a much larger variance in the Hamming distance difference between different templates. This is shown in Figure 13 and is directly observable in larger variance in the number of bad matches in Figure 14. This is in contrast to the histogram of good matches across datasets in Figure 15 which are consistent across datasets.This leads to an increase in the selected δ which will directly impact the efficiency of the cryptographic algorithms. As we increase dataset sizes, δ for synthetic data is about 5 times δ for random data.

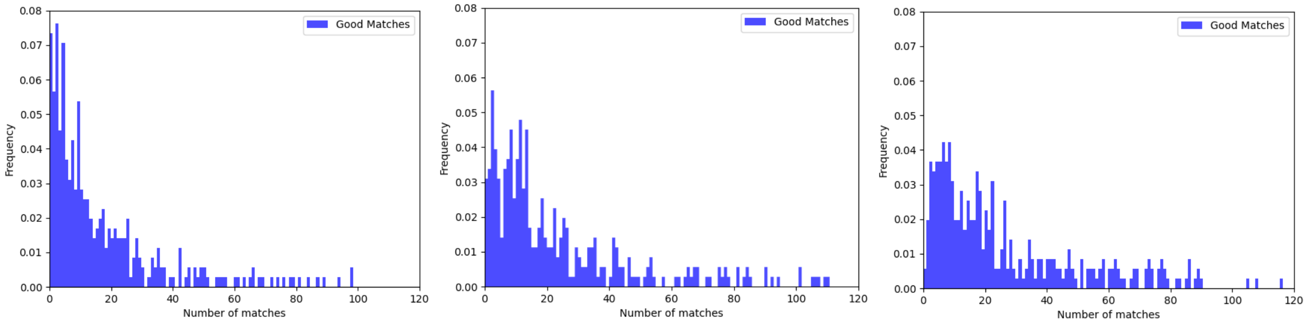
The ND and synthetic data statistics align well, requiring the same parameters of $\alpha = 18$ and $\beta = 850$. This gives some indication that parameters for larger synthetic dataset sizes would yield comparable performance on real irises. More evaluation of real data is necessary before deployment. Across all parameter settings $\delta/\beta < .06$ validating the basic design of using an OMC to reduce the number of clauses as input to the DOMapE.

Restricting to only δ accesses in the DOMapE has an effect on the TAR of the system. Note that the real system does not know which matches are good so even in the case when many matches are bad it is possible for a single good match to be included in the arbitrarily selected δ traversals. The worst degradation of TAR is for synthetic data with 2500 records where TAR drops from .92 to .82.



(a) ND-0405 (b) Synthetic Data (c) Random Data

Figure 14: Number of bad matches across datasets of size 356.



(a) ND-0405 (b) Synthetic Data (c) Random Data

Figure 15: Number of good matches across datasets of size 356.

6.2 Speed - Cryptographic Efficiency

Efficiency was evaluated in terms of storage, timing, and network roundtrips of the implemented algorithms. The machine is a RHEL 7.9 with:

1. An Intel(R) Xeon(R) CPU E5-2630 v3 with 32 cores at 2.40GHz,
2. 64 GB (4x16GB) 2400MHz DDR4 RDIMM ECC RAM, and
3. Two 2TB SATA 7.2K RPM HDD in Raid1.

The results are reported in Table 2. Before reporting on results we note that we did not attempt to model network delay in our experiments. This is because the underlying PathORAM implementation does not support batching ORAM requests into a single roundtrip. Thus, any implementation would have an inflated number of rounds trips, as shown in the sequential round trips column of Table 2.

Storage efficiency Feature vectors are 1024 bit vectors. So for a dataset of size 5000, this represents approximately 640 KB. The unprotected (same structure as DOMapE but without ORAM) index takes approximately 122.3 MB.[7]

As shown in Table 2, for our encrypted storage this amounted to 35.6 GB in storage. This represents a storage increase factor of around 22 between raw data and unprotected index, and ORAM increases storage again approximately 291 times. As we discuss in the Conclusion, one can more efficiently pack ORAM blocks using trees with a branching factor > 2.

[7] Internal node consists of an LSH number, a 22 LSH values, and 2 child identifiers (either the node id or the position of the child in the next ORAM). Leaf nodes consist of a single 32 node identifier.

Dataset	Dataset				#	ORAM	# Roundtrips		Time (s)			Size
size, ℓ	type	α	β	δ	Queries	Reads	sequential	parallel	O.Init	Search	O.Read	(GB)
356	random	15	630	13	356	117	334	18	3.8×10^3	11	.051	1.1
356	ND	18	850	37	356	333	666	18	13.8×10^3	39	.062	1.1
356	synthetic	18	850	26	356	234	468	18	12.3×10^3	27	.061	1.1
1000	random	18	850	8	500	80	160	20	11×10^3	17	.122	2.2
1000	synthetic	19	1000	47	500	470	940	20	13×10^3	113	.135	2.2
2500	random	19	1000	11	500	132	262	24	59×10^3	90	.353	8.9
2500	synthetic	21	1200	56	100	672	1344	24	202×10^3	553	.420	17.8
5000	random	20	1200	12	500	156	312	26	166×10^3	235	.757	35.6
5000	synthetic	22	1300	72	10	936	1872	26	904×10^3	1570	.849	35.6

Table 2: Efficiency results. O.Init is time to initialize all ORAMs. O.Read is average read time (across ORAM layers). Search is time per query and includes tree traversals. Size EDB denotes the size of the ORAM files that are stored on the server (OMC storage is ignored since it is much smaller). Sequential number of roundtrips is $2 * \#$ORAM Reads and Parallel Rounds trips is $\lceil \log_2 \ell \rceil * 2$. All timing numbers are averaged across the number of queries in # Queries.

Time efficiency We time the two main OSE algorithms: BuildIndex and Search. The time to build the encrypted index is largely dominated by the ORAM setup time so we only report the later (column "O.Init"). For small datasets (356 records), ORAM setup takes between 1 and 4 hours, respectively for random and real data. For larger datasets of size 5000, ORAM setup takes almost 2 days for random data and approximately 11 days for synthetic data. While these timings are not prohibitive yet, they will be for very large datasets. We believe that part of this problem is caching of the large data structures to virtual memory and the use of a spinning disk hard drive that perform poorly with ORAM workloads [WST12, Section 2.3]. To check this hypothesis, we ran the synth dataset of size 356 on a M1 Mac mini desktop with 16GB of memory and a 2TB SSD, and observed a 4 fold reduction in ORAM setup time.

Search time (per query) takes only a few dozen seconds for datasets of size 356 and it reaches a few minutes for larger datasets (approximately 4 minutes for 5000 random and 26 minutes for 5000 synthetic). Although these timings for larger datasets are not ideal, they are a vast improvement compared to previous works. In particular, search for ND-0405 (356 records) takes only 39 seconds compared to Cachet et al's search time of approximately 1 hour [CAD+20].[8] Furthermore, if one had to search all β trees, search time would increase by a factor of at least $1/.06 \approx 16$.

Network Round Trips Our primary network measurement is number of roundtrips. We report on two figures, the number of round trips using a purely sequential PathORAM implementation and the number of roundtrips if one is able to fully batch all requests at the same level. For the largest synthetic parameter sizes, if one assumes a fast network with $60ms$ responses and unbounded bandwidth then network delays result in 1.56 seconds in parallel rounds trips, but slower $1s$ responses results in 26 seconds. If one assumes sequential round trips and $60ms$ responses the network delay alone is $112s$. For comparison, we note that our local ORAM read operation took $.849s$ on this dataset, indicating the spinning hard disk was substantially slower than a fast network.

Parallelization The current implementation does not use parallelization because of the non-parallel ORAM module. Many existing ORAM schemes including PathORAM naturally supported batched read/write operations where the client keeps a larger stash. In the case of PathORAM, the client repeatedly reads and writes a "random" path on a tree. One can naturally perform all reads first and then perform all writes, simulating the intermediate storage that would be held by the server. Parallel ORAM is a more complex solution for when the reads are coming from different clients [WST12, BCP16, CLT16]

Evaluation of OMC implementation using private set intersection On the same hardware as the rest of the evaluation we deployed the VolePSI implementation [RS21]. We deployed this with a server set of size 6.5 million items and a client set of 1300 items. This corresponds to the largest set of parameters in Table 2. VolePSI is based on OT extension and requires a setup phase. We benchmarked 32 PSI iterations with the first taking 766ms and the rest taking 2ms of computation. We note that VolePSI requires 7 messages of communication. These results justify the focus on the design of DOMapE.

[8] Their system does have a better accuracy tradeoff with a TAR of .99 with an average of 15 false positives in comparison to our TAR of .89, see [ACD+22, Table 2].

7 Conclusion

This work presents *Private Eyes* the first zero-leakage biometric database. Our system is tested with response times in minutes on databases of thousands of irises. Our construction combines LSHs and oblivious maps. The unique aspect of our design is the recognition and mitigation of the cryptographic inefficiencies caused by the high noise in biometric data. In particular, we use the statistics of biometric data to create our two-stage approach which filters which LSHs to query using a lighter-weight membership checking primitive before the heavy-weight oblivious map.

More work is needed to help zero-leakage biometric database scale to the level of national biometric identity databases which have millions of records. Cachet et al. [ACD+22] proposed two non-interactive iris proximity search schemes based on inner-product encryption. Both of their constructions have more leakage than our system. The first one leaks the distance between all returned points and the query. The other leaks whether returned records are the same distance from the query. For the solution with more leakage, their search took 4 minutes on a dataset of size 356. For the solution with less leakage, search took 4 hours (later reduced to an 1 hour in an ePrint revision [CAD+20]). Our search time on the same dataset is 14 seconds.

Here we present a couple of challenges and opportunities for future work for dealing with that scale:

1. Existing ORAM implementations do not support batching of read/write requests, such a capability is important for practical zero-leakage disjunctive search with a large number of clauses.

2. We relied on binary trees which mean that each tree traversal requires $10-20$ ORAM lookups. Naturally, one can use trees with a higher branching factor (or skiplists as in [BC14]) to reduce the number of ORAM lookups. Ideally, each node would correspond to a single ORAM block which is commonly a multiple of 256 bytes. Our current estimate is that internal nodes account for ≤ 128 bits of storage out of the 256 byte block size. As such, one could make the tree into a 18-ary tree (32 bits for LSH number, 32 bits for left most child, and $22+32$ bits for each additional comparison node). This would reduce the depth of the trees and required number of round trips by $\log_2 18 \approx 4$.

References

[ACD+22] Sohaib Ahmad, Chloe Cachet, Luke Demarest, Benjamin Fuller, and Ariel Hamlin. Proximity searchable encryption for the iris biometric. In *AsiaCCS*, 2022. https://ia.cr/2020/1174.

[AF18] Sohaib Ahmad and Benjamin Fuller. Unconstrained iris segmentation using convolutional neural networks. In *Asian Conference on Computer Vision*, pages 450–466. Springer, 2018.

[AF19] Sohaib Ahmad and Benjamin Fuller. Thirdeye: Triplet-based iris recognition without normalization. In *IEEE International Conference on Biometrics: Theory, Applications and Systems*, 2019.

[AF20] Sohaib Ahmad and Benjamin Fuller. Resist: Reconstruction of irises from templates. In *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–10. IEEE, 2020.

[Ahm20] Sohaib Ahmad. Sohaib ahmad github, 2020. Accessed: 2020-07-23.

[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.

[BC14] Alexandra Boldyreva and Nathan Chenette. Efficient fuzzy search on encrypted data. In *International Workshop on Fast Software Encryption*, pages 613–633. Springer, 2014.

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram and applications. In *Theory of Cryptography Conference*, pages 175–204. Springer, 2016.

[BF16] Kevin W Bowyer and Patrick J Flynn. The ND-IRIS-0405 iris image dataset. *arXiv preprint arXiv:1606.04853*, 2016.

[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.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[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.

[CAD+20] Chloe Cachet, Sohaib Ahmad, Luke Demarest, Serena Riback, Ariel Hamlin, and Benjamin Fuller. Multi random projection inner product encryption, applications to proximity searchable encryption for the iris biometric. Cryptology ePrint Archive, Paper 2020/1174, 2020. `https://eprint.iacr.org/2020/1174`.

[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.

[CGKO11] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19:895–934, 01 2011.

[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.

[CHF22] Chloe Cachet, Julie Ha, and Benjamin Fuller. An implementation of oblivious proximity searchable encryption. `https://github.com/hajulie/searchable_biometric`, 2022.

[CLT16] Binyi Chen, Huijia Lin, and Stefano Tessaro. Oblivious parallel ram: improved efficiency and generic constructions. In *Theory of Cryptography Conference*, pages 205–234. Springer, 2016.

[CT09] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. Cryptology ePrint Archive, Paper 2009/491, 2009. `https://eprint.iacr.org/2009/491`.

[CWD+18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[Dau05] John Daugman. Results from 200 billion iris cross-comparisons. 01 2005.

[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.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800, 2013.

[DGXZ19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2019.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *International Conference on Applied Cryptography and Network Security*, pages 125–142. Springer, 2009.

[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 Lexington United States, 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.

[FNP04] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer, 2004.

[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.

[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.

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*, pages 563–592. Springer, 2016.

[GPAM$^+$20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[GPP23] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. In *IEEE Security and Privacy*, 2023. `https://eprint.iacr.org/2021/879`.

[GRGB$^+$12] Javier Galbally, Arun Ross, Marta Gomez-Barrero, Julian Fierrez, and Javier Ortega-Garcia. From the iriscode to the iris: A new vulnerability of iris recognition systems. *Black Hat Briefings USA*, 1, 2012.

[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.

[Hac18] Gabriel Hackebeil. Path oram python module, 2018.

[HWKL18] Yi Huang, Adams Kong Wai-Kin, and Kwok-Yan Lam. From the perspective of CNN to adversarial iris images. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–10. IEEE, 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.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[JM19] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *International Conference on Learning Representations*, 2019.

[KB15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[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 *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 90–108. IEEE, 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.

[KPT19a] 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.

[KPT19b] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. Cryptology ePrint Archive, Report 2019/441, 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.

[KYV$^+$17] Naman Kohli, Daksha Yadav, Mayank Vatsa, Richa Singh, and Afzel Noore. Synthetic iris presentation attack using idcgan. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*, pages 674–680. IEEE, 2017.

[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.

[LN$^+$96] Michael Lamoureux, Bradford Nickerson, et al. On the equivalence of b-trees and deterministic skip list. 1996.

[MCYJ18] Guangcan Mai, Kai Cao, Pong C Yuen, and Anil K Jain. On the reconstruction of face images from deep face templates. *IEEE transactions on pattern analysis and machine intelligence*, 41(5):1188–1202, 2018.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[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.

[PBF$^+$08] P Jonathon Phillips, Kevin W Bowyer, Patrick J Flynn, Xiaomei Liu, and W Todd Scruggs. The iris challenge evaluation 2005. In *2008 IEEE Second International Conference on Biometrics: Theory, Applications and Systems*, pages 1–8. IEEE, 2008.

[PKV$^+$14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE, 2014.

[PSO$^+$09] P Jonathon Phillips, W Todd Scruggs, Alice J O'Toole, Patrick J Flynn, Kevin W Bowyer, Cathy L Schott, and Matthew Sharpe. FRVT 2006 and ICE 2006 large-scale experimental results. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):831–846, 2009.

[Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[RS21] Peter Rindal and Phillipp Schoppmann. Vole-psi: Fast oprf and circuit-psi from vector-ole. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 901–930, Cham, 2021. Springer International Publishing.

[SDDN19] Sobhan Soleymani, Ali Dabouei, Jeremy Dawson, and Nasser M Nasrabadi. Adversarial examples to fool iris recognition systems. In *2019 International Conference on Biometrics (ICB)*, pages 1–8. IEEE, 2019.

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

[SSF19] Sailesh Simhadri, James Steel, and Benjamin Fuller. Cryptographic authentication from the iris. In *International Conference on Information Security*, pages 465–485. Springer, 2019.

[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.

[VS11] Shreyas Venugopalan and Marios Savvides. How to generate spoofed irises from an iris code template. *IEEE Transactions on Information Forensics and Security*, 6(2):385–395, 2011.

[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.

[WST12] Peter Williams, Radu Sion, and Alin Tomescu. Privatefs: A parallel oblivious file system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 977–988, 2012.

[YCR19] Shivangi Yadav, Cunjian Chen, and Arun Ross. Synthesizing iris images using RaSGAN with application in presentation attack detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

Acknowledgements

The authors thank the anonymous reviewers for their help in improving the manuscript. The authors also thank Mayank Varia for helpful discussions. J.H. is supposed by NSF Award #1950600. C.C. is supported by NSF Award #2141033. L.D. is supported by the Harriott Fellowship. S.A. is supported by a fellowship from Synchrony Inc. and the State of Connecticut. B.F. is supported by NSF Awards # 2141033 and 2232813 and ONR Grant N00014-19-1-2327.

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

Layer	OutputSize	Kernels
Input	1x128	-
G1	64	64
G2	128	128
G3	64	64
G4	128	128
G5	1024	1024
D1	16	16
D2	128	128
D3	1	1

Table 3: Generator architecture

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.

A Synthetic Data Generation

We now describe the neural network used to produce our synthetic templates. Our synthetic templates are built using a generative adversarial network or GAN. A GAN trains two networks in competition, a generator which should produce synthetic irises and a discriminator which classifies irises as real or synthetic. Yadav et al. [YCR19] uses RaSGAN (relativistic average standard GAN) [JM19] to generate synthetic irises for the purpose of studying their effects on presentation attack detection (PAD) algorithms. Irises from the RaSGAN perform well against PAD and follow real iris statistics well. Kohli et al. [KYV$^+$17] use the DCGAN architecture to generate synthetic irises. Synthetic irises can be viewed as irises that must closely resemble bonafide irises as discussed in [YCR19, KYV$^+$17].

Our synthetic data generator is also trained using the ND-0405 dataset [BF16]. We follow the approach of RESIST [AF20] which takes inspiration from synthetic data generation to invert iris templates into realistic looking images.

We denote the network as `SYNTH`. In a GAN formulation, noise is sampled from a multivariate normal distribution ($\mathbb{P}_z$) with a mean of 0 and a variance of 1. The generator converts this noise vector into a synthetic template. Let $\mathbb{P}_y$ denote the distribution of real templates and $\mathbb{P}_{\hat{y}}$ denote the distribution of synthetic templates. We use a recently proposed relativistic average discriminator [JM19] as our discriminator. To build up to the relativistic discriminator we first start with the original GAN loss functions:

$$\begin{aligned} L(D) &= -\,\mathbb{E}_{y\sim\mathbb{P}_y}[\log(D(y))] - \mathbb{E}_{\hat{y}\sim\mathbb{P}_{\hat{y}}}[\log(1 - D(\hat{y}))] \\ L(G) &= -\,\mathbb{E}_{\hat{y}\sim\mathbb{P}_{\hat{y}}}[\log(D(\hat{y}))]. \end{aligned}$$

$L(D)$ is called the discriminator loss and $L(G)$ is called the generator loss, y is an actual template and $\hat{y}$ is a synthetic template generated by the generator.

The $L(D)$ and $L(G)$ losses are minimized using gradient descent. The generator and discriminator play a zero sum game. The generator weights are updated based on how good its synthetic irises are while the discriminator weights are updated on how well it differentiates between real and synthetic templates.

The last layer of the generator is the hyberbolic tangent (tanh) with output ranging from -1 to 1, this is done to generate binary synthetic templates by substituting 0,1 with -1,1. The real templates are also converted to -1,1 for conformity.

Architecture `SYNTH` architecture is a small neural network having only dense (fully connected) layers as shown in Table 3 where G$_x$ are generator layers and D$_x$ are discriminator layers. Each layer is followed by a LeakyReLU [MHN13] activation and a batch normalization [IS15] layer. The last layers of both sub-networks are unique, the generator has a tanh activation while the discriminator has a Sigmoid activation.

Training SYNTH is trained in two stages. First, the generator produces a synthetic template and second, the discriminator outputs how real this synthetic template is. This training is done till convergence of the weights of both networks. Both training stages use the Adam optimizer [KB15]. We randomly flip 2% bits of a real template as noise to aid in network convergence. The networks is trained over 100 epochs. Each epoch having 100 steps. Each step updates weights once by either 1) discriminating a pair of vectors or 2) generating a single synthetic template.

Once trained the SYNTH network can produce an unbounded number of distinct templates. We described how to produce different readings from the same template in Section 4.