Differentially Private Access Patterns for Searchable Symmetric Encryption

Guoxing Chen†, Ten-Hwang Lai†, Michael K. Reiter‡, Yinqian Zhang†
†Department of Computer Science and Engineering, The Ohio State University, USA
‡Department of Computer Science, University of North Carolina at Chapel Hill, USA
{chenguo, lai, yinqian}@cse.ohio-state.edu, reiter@cs.unc.edu

***Abstract*—Searchable encryption enables searches to be performed on encrypted documents stored on an untrusted server without exposing the documents or the search terms to the server. Nevertheless, the server typically learns which encrypted documents match the query—the so-called *access pattern*—since the server must return those documents. Recent studies have demonstrated that access patterns can be used to infer the search terms in some scenarios. In this paper, we propose a framework to protect systems using searchable symmetric encryption from access-pattern leakage. Our technique is based on d-privacy, a generalized version of differential privacy that provides provable security guarantees against adversaries with arbitrary background knowledge.**

I. Introduction

Encryption is often used to protect data stored in incompletely trusted servers (e.g., public clouds). Searchable symmetric encryption (SSE) further enables that data to be searched by its owner and only the matched documents returned, without disclosing the search terms or the content of the matched documents to the server. A long line of research has investigated SSE with improved efficiency, stronger security and more flexible functionality [1]–[8]. Typically, with an SSE scheme, a client first produces an encrypted version of the database, along with encrypted metadata, to be outsourced to a cloud server. Later, the client can interact with the server to perform a search on the encrypted database and retrieve the encrypted results, which will be decrypted locally.

When a search query is made by the client, however, the cloud server is typically able to observe which files are accessed in the encrypted database and returned to the client. This type of leakage is called access-pattern leakage. To be used in practice, most existing SSE schemes allow this type of access-pattern leakage. However, recent studies (e.g., [9], [10]) have demonstrated that with some *a priori* knowledge of the outsourced documents, the adversary is able to recover the content of the queries with high accuracy.

One solution to access-pattern leakage is oblivious RAM (ORAM) [11], [12]. An ORAM algorithm allows a client to hide its access pattern from the remote server by continuously shuffling and re-encrypting data as they are accessed. To obliviously access one of n documents in the storage, at least $O(\log n)$ documents need to be accessed [11]. This overhead makes it impractical to use ORAM to hide the access pattern for SSE. As pointed out by Naveed [13], the communication overhead of SSE schemes using ORAM could be larger than that of simply sending back all data stored in the remote server.

In this paper, we propose to protect SSE schemes against access-pattern leakage with a form of access-pattern obfuscation (APO). We borrow a statistical privacy definition, *differential privacy* [14], to define the desired security guarantee of such an obfuscation mechanism. Specifically, we adopt a generalized version of differential privacy, d-privacy [15], and aim to obfuscate the access patterns of SSE schemes so that access patterns follow the definition of d-privacy. In particular, d-privacy implies that the adversary cannot distinguish between queries using distinct search terms that induce access patterns that are within specified distance (in terms of a distance metric d) of one another.

Rather than constructing a new SSE scheme, our framework simply builds over any given SSE scheme. That is, given a database, our framework applies obfuscation to the database and then provides the obfuscated database to the SSE scheme. When a search query is processed, obfuscated results are obtained, and then de-obfuscated by our framework.

In sum, the contributions of this paper are as follows:

- We define d-privacy for access patterns of general SSE schemes.
- We propose a d-private access-pattern obfuscation mechanism that is compatible with existing SSE schemes.
- We implement a prototype of the proposed obfuscation mechanism.

The paper is organized as follows: Sec. II introduces preliminaries of SSE and the threat model. Sec. III presents the overview of our proposed access-pattern obfuscation framework. Sec. IV presents our privacy definition for access-pattern obfuscation, and a d-private mechanism to obfuscate the access patterns of SSE schemes. Sec. V discusses how the parameters of the obfuscation framework can be selected optimally. Sec. VI presents the implementation of a prototype. Sec. VII details the evaluation and Sec. VIII summarizes the related work. Finally, Sec. IX concludes the paper.

II. Background

A. Searchable Symmetric Encryption

SSE was introduced by Song et al. [1] with a scheme whose search time is linear in the size of the database. Curtmola et al. [2] were the first to design an index-based SSE scheme that

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

achieves sublinear search time. This index-based construction contributed to various subsequent efficient SSE schemes [3]–[8]. In this paper, we focus on index-based SSE schemes.

Let $\mathbf{D} = (D_1, D_2 \cdots, D_n)$ denote a collection of n documents. Let $\mathbf{\Delta} = \{w_1, w_2, \cdots, w_{\|\mathbf{\Delta}\|}\}$ be a set of all possible keywords. Each document D_i is associated with a keyword list $W_i \subseteq \mathbf{\Delta}$ which consists of all the keywords the document D_i should be indexed with. For most existing SSE schemes, the keywords are extracted from the content of the document, typically using keyword extraction algorithms provided by the SSE implementations.

A symmetric key encryption scheme is a tuple of three polynomial-time algorithms $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes as input a security parameter κ and outputs a secret key K; Enc is a probabilistic algorithm that takes as input a key K and a message π, and outputs a ciphertext c; Dec is a deterministic algorithm that takes as input a key K and a ciphertext c, and outputs π if c is produced from π under K.

A basic single-server index-based SSE scheme consists of algorithms for building and searching a secure index, and a symmetric key encryption scheme for encrypting and decrypting the documents. Specifically, an index-building algorithm takes as input a secret key K and a collection $\mathbf{D}$ of documents along with their keyword lists $\mathbf{W}$, and outputs a secure index $\mathbf{I}$. A document collection $\mathbf{D}$ is encrypted as a collection $\mathbf{c}$ of ciphertexts using SKE.Enc. Both $\mathbf{I}$ and $\mathbf{c}$ are stored in the remote server. The search algorithm takes as input a query τ and a secure index $\mathbf{I}$, and outputs a set of indices that point to the ciphertexts to be returned to the client, where they are decrypted locally using SKE.Dec. These algorithms are called during an SSE scheme's different phases, referred as the setup phase and the search phase, shown in Fig. 1. A more formal definition of SSE is provided as follows.

Definition 1. *(Searchable Symmetric Encryption (SSE)). An SSE scheme is a tuple* (KeyGen, BuildIndex, Token, Search, SKE) *of four polynomial-time algorithms and a symmetric key encryption scheme such that,*

- $(K_I, K_D) \leftarrow \mathsf{KeyGen}(1^\kappa)$ *is a probabilistic key generation algorithm for the client to setup the SSE scheme. It takes a security parameter κ as input, and outputs a secret key K_I for the secure index, and a secret key $K_D \leftarrow \mathsf{SKE.Gen}(1^\kappa)$ for the document collection.*
- $\mathbf{I} \leftarrow \mathsf{BuildIndex}(K_I, (\mathbf{D}, \mathbf{W}))$ *is a probabilistic algorithm for the client to build a secure index. It takes as input a secret key K_I, a document collection $\mathbf{D}$ along with keyword lists $\mathbf{W}$, and outputs a secure index $\mathbf{I}$.*
- $\tau \leftarrow \mathsf{Token}(K_I, w)$ *is a (possibly probabilistic) algorithm for the client to generate search tokens. It takes a secret key K_I and a keyword w, and outputs a search token τ.*
- $\mathbf{R} \leftarrow \mathsf{Search}(\mathbf{I}, \tau)$ *is a deterministic algorithm for the server. It takes as input a secure index $\mathbf{I}$ and a search token τ, and outputs a set $\mathbf{R}$ of document identifications.*
- $c \leftarrow \mathsf{SKE.Enc}(K_D, D)$ *is a probabilistic algorithm for the client to encrypt the document collection. It takes a secret key K_D and a document D, and outputs a ciphertext c. A collection $\mathbf{c} = \{c_1, c_2, \ldots, c_n\}$ of ciphertexts is produced by encrypting the documents in $\mathbf{D}$ one by one.*
- $D \leftarrow \mathsf{SKE.Dec}(K_D, c)$ *is a deterministic algorithm for the client to decrypt a ciphertext of a document. It takes as input a secret key K_D and a ciphertext c, and outputs a document D.*

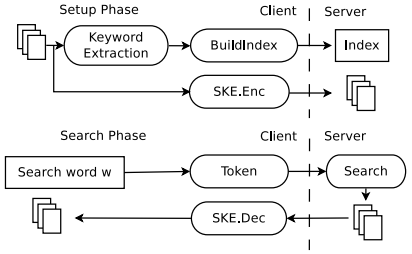


Fig. 1: Searchable Symmetric Encryption schemes

An SSE scheme is *correct* if the returned set of indices contains all indices of the documents that satisfy the query.

The main difference between our SSE definition and others from the literature [2] is that ours explicitly describes the keyword lists in BuildIndex while others implicitly assume the keyword lists are all words extracted from the documents.

While more recent SSE schemes might have additional algorithms to support improved utility, such as updates [3] and boolean queries [5], we focus on the above basic setting in this paper for its generality.

B. Query Recovery Attacks

Almost all existing efficient SSE schemes, except costly ORAM based schemes [12], leak access patterns, namely, which documents are returned in response to a query. However, it has been demonstrated that SSE schemes with access-pattern leakage are vulnerable to *query recovery attacks*.

The goal of a query recovery attack is to recover the content of a query (i.e., the plaintext keyword) issued by the client to the server. This attack, which was first proposed by Islam et al. [9], is usually dubbed the *IKK attack*. In the IKK attack, it is assumed that the adversary has some *a priori* knowledge of the whole document collection. In particular, the IKK attack assumes that the adversary has the knowledge of a $r \times r$ matrix M that depicts the probability of keyword co-occurrence, where r is the number of keywords that are known to the adversary. The value in the $(i, j)^{th}$ cell of matrix M represents the probability that both the i^{th} and j^{th} keywords appear in any random document $D \in \mathbf{D}$. Note if the adversary can exactly compute matrix M for the entire document collection, the effectiveness of the attack can be greatly improved.

After observing the access patterns revealed by l client-issued queries, the adversary carries out the attack as follows: An $l \times l$ co-occurrence matrix $\hat{M}$ is computed from the observed access patterns. This should be a sub-matrix of M. The best match of $\hat{M}$ to M can be generated by optimization methods such as simulated annealing. From the match, a guess for each query can be made. The authors conducted evaluations on the Enron email dataset [16]. The queryable

keywords were the most common words in the document collection, after removing a set of stopwords (e.g., "the", "is"). Queries were picked from these keywords uniformly at random. A near perfect (100%) recovery rate was reported with 500 queryable keywords and 150 unique queries.

Cash et al. [10] improved the IKK attack with the additional assumption that the adversary also knows the number of documents in the document collection that matches each keyword. Instead of using a co-occurrence probability matrix, the adversary uses a co-occurrence count matrix. Each $(i,j)^{th}$ cell contains the number of documents containing both i^{th} and j^{th} keywords. The evaluations on the Enron email dataset demonstrated a recovery rate of near 100% when the number of queryable keywords varied from 500 to 7500.

C. Erasure Coding

Erasure coding is widely used in storage systems to tolerate failures by adding redundancy to stored data. An erasure code transforms a "message," such as a document, into a longer message in a way that the original message can be reconstructed from the longer message even if parts of the longer message have been lost. More specifically, for a message of k symbols, a longer message of m symbols can be generated in such a way that the original message can be recovered from any k symbols of the longer message. Such an erasure code is called a k out of m erasure code.

In this paper, erasure coding works as follows: a document is equally partitioned into k shards.Then, $m-k$ parity shards are produced such that the original document can be reconstructed from any k of these m shards. Note that all these m shards are of the same size, which is $\frac{1}{k}$ of the original's size.

D. Attack Model

The scenario considered in this paper is a user searching documents, through SSE client software, in an encrypted database that is stored on a remote server. The adversary in the scenario is the remote server who is capable of observing the accessed indices and documents (i.e., the access patterns) and is curious about the content of the user's queries.

To be more specific, in this paper, we consider an *honest-but-curious adversary, who has complete knowledge of the document collection*. An honest-but-curious server follows the pre-defined algorithms run between the server and the client. It simply passively monitors the storage access patterns and infers the content of the corresponding queries.

We emphasize that our threat model is different from the model for private information retrieval (PIR). PIR allows the content of each uploaded document to be known by the adversary but aims to prevent the adversary from learning the content of the queries [17], [18]. In contrast, in our threat model, each document is encrypted before it is uploaded to the server. The adversary is unable to figure out which document a ciphertext is created from. Moreover, the queries of SSE schemes are tokens generated from the keywords, but most PIR schemes do not involve keywords.

III. Framework for Access-Pattern Obfuscation

In this section, we present a framework to obfuscate access patterns in a principled way. The design goal of our framework is to provide an access-pattern obfuscation mechanism that is compatible with existing SSE schemes.

To obfuscate the access patterns, we need to add both *false positives* and *false negatives* to the search results. By false positives, we mean the server returns some documents that do not match the query (also called fake documents, or dummy documents in prior work [9], [10], [19]). By false negatives, we mean the server does not return some documents that match the query. While false positives contribute to the communication overhead, false negatives would violate the *correctness* of the SSE scheme.

To handle the *correctness* issue, we introduce redundancy to the document collection using erasure codes. Each document is encoded into multiple shards, and the collection of all these shards, instead of the original document collection, is encrypted and outsourced to the remote server. As long as enough shards of a matched document are returned, this document can be successfully decoded and presented to the client. We use *recall rate* to represent the percentage of matching documents that are successfully decoded for a query. We argue that allowing a probability that some matching documents may not be returned is essential to achieve a provable privacy guarantee such as differential privacy, and a fairly high recall rate, say 99.99%, can be useful in practice already.

Our framework has two phases, in accordance with the setup phase and the search phase of an SSE scheme. We will detail these phases in Sec. III-A and Sec. III-B, respectively.

A. Setup Phase

In the SSE setup phase, as described in Sec. II, the SSE client first extracts keywords from the documents, and then builds the secure index using the BuildIndex algorithm provided by the SSE scheme. To work with any SSE implementations, our design of the access-pattern obfuscation framework does not make any changes to the BuildIndex algorithm itself, but rather obfuscates the (D, W) pairs before using them as the input of BuildIndex algorithm.

Suppose a k out of m erasure code is used. Fig. 2 illustrates the access-pattern obfuscation during the setup phase.

1. For each document D, we first use the keyword extraction algorithm of the SSE scheme to extract a keyword list W;
2. Erasure coding is applied to create m shards for D. Each shard is appended with the same keyword list W of D.
3. For newly created shards and their keyword lists, the access-pattern obfuscation mechanism is applied (detailed in Sec. IV) so that some shards of the matching documents will not be returned in response to a query (i.e., false negatives) while some shards of the non-matching documents will be returned (i.e., false positives).
4. The rest of the setup process of the original SSE scheme is performed. The encrypted shards are uploaded to the remote server in a random order to prevent the adversary from

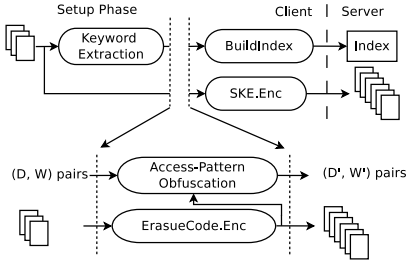


Fig. 2: Access-pattern obfuscation in the setup phase.

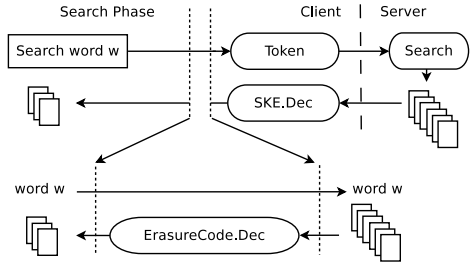


Fig. 3: Access-pattern obfuscation in the search phase.

Algorithm 1: Setup_APO

Input: A document collection and keyword lists ($\mathbf{D}$,$\mathbf{W}$); Secret keys (K_I, K_D); Parameter m, k
Output: a secure index $\mathbf{I}$ and a ciphertext collection $\mathbf{c}$;
initialize $(\mathbf{D}', \mathbf{W}') = (\{\}, \{\})$;
for $(D, W) \in (\mathbf{D}, \mathbf{W})$ **do**
  $D_1, \ldots, D_m \leftarrow ErasureCode.Enc(m, k, D)$
  for $i \in [1, \ldots, m]$ **do**
    $D', W' \leftarrow \mathsf{Func\_APO}(D_i, W)$
    append D' to $\mathbf{D}'$, W' to $\mathbf{W}'$
$\mathbf{I} \leftarrow \mathsf{BuildIndex}(K_I, (\mathbf{D}', \mathbf{W}'))$
$\mathbf{c} \leftarrow \{c | c \leftarrow \mathsf{SKE.Enc}(K_D, D'), D' \in \mathbf{D}'\}$
return ($\mathbf{I}$, $\mathbf{c}$)

Algorithm 2: Search_APO

Input: A search keyword w; Secret keys (K_I, K_D); Parameter m, k
Output: Resulting documents $\hat{\mathbf{D}}$;
initialize $\hat{\mathbf{D}} = \{\}$;
$\tau \leftarrow \mathsf{Token}(K_I, w)$
$\mathbf{R} \leftarrow \mathsf{Search}(\mathbf{I}, \tau)$
$Shard_i \leftarrow \mathsf{Dec}(K_D, c_i), i \in \mathbf{R}$
Group $Shard_i$ of the same original documents together
for *each shard group* $\mathbf{G}$ **do**
  if $|\mathbf{G}| \geq k$ **then**
    $D \leftarrow ErasureCode.Dec(m, k, \mathbf{G})$
    append D to $\hat{\mathbf{D}}$
return $\hat{\mathbf{D}}$

trivially figuring out that consecutively uploaded ciphertexts are shards of the same document.

Due to the use of erasure codes, the number of outsourced units (originally documents, now shards) increases by m times. From the view of the client, she has a collection of n_c documents. From the view of the server, he stores $n = m \times n_c$ shards. We will use the notation n_c and n in the following sections indiscriminately.

Algorithm 1 shows the algorithm used for access-pattern obfuscation in the setup phase. The access-pattern obfuscation function Func_APO will be detailed in Sec. IV.

B. Search Phase

In the search phase, the search related algorithms of the original SSE scheme will be performed first. The returned shards are decrypted. And the matching documents with enough shards are reconstructed. Fig. 3 shows this procedure.

1. Given a search keyword w, the default Token algorithm is applied and the resulting search token is sent to the server.
2. The received shards are decrypted.
3. Shards from the same original documents are grouped together. Documents with at least k shards are reconstructed and presented to the client.

Algorithm 2 shows the algorithm used for access-pattern obfuscation in the search phase.

IV. d-Private Access-Pattern Obfuscation

A. Formal Definition

Access Patterns. Consider a collection of n_c documents, each of which is encoded into m shards using k out of m erasure coding. The resulting $n = m \times n_c$ shards are shuffled, encrypted, and then stored in the server. We use a n-bit vector x to represent the access pattern of the shards when processing a query τ, with $x_i = 1$ indicating the i-th shard is accessed and returned, and $x_i = 0$, not accessed. For example, when $n_c = 2, k = 2, m = 3$, the server stores 6 encrypted shards. Suppose the first, third and fourth shards are from the first original document and the rest are from the other. An access pattern $x = 101100$ means the first, the third and the fourth shards are retrieved. Therefore, the first original document will be successfully decoded. We will use this case as a running example in the following discussion. Let $\mathcal{X} \subseteq \{0, 1\}^n$ denote the set of all possible access-pattern vectors.

As mentioned in Sec. III, to obfuscate access patterns, we need to intentionally induce false positives and false negatives. Interpreted from the view of an access-pattern vector, we need to flip some bits of x from 0 to 1 (returning non-matching shards, i.e., false positives) and flip some bits from 1 to 0 (not returning matching shards, i.e., false negatives), to obtain an obfuscated access pattern y. In our running example, suppose the access-pattern vector $x = 101100$ is obfuscated into $y = 001110$. When the query τ is processed, the server returns two matching shards (the third and fourth) and a non-matching shard (the fifth). Since $k = 2$, two shards are enough to decode the first original document. We will discuss how to choose m and k to achieve high recall in Sec. V. Here, we focus on how to obfuscate access patterns with privacy guarantees.

The goal of our privacy definition is to guarantee that for access patterns that are *similar*, the obfuscated access patterns generated from them are indistinguishable. For example, when $x = 111000$, $x' = 110100$, and an obfuscated access pattern $y = 110001$ is observed with high probability, the adversary cannot tell whether x or x' was the original access pattern. We use *metrics* to measure the similarity of access patterns.

Metrics. A metric on a set $\mathcal{X}$ is defined as a function d: $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, such that $d(x, y) = 0$ iff $x = y$; $d(x, y) = d(y, x)$; and $d(x, y) + d(y, z) \geq d(x, z)$ for all $x, y, z \in \mathcal{X}$.

For example, Hamming distance d_h is a metric for n-bit binary vectors, such that $d_h(x, y) = |\{i \mid x_i \neq y_i, i = 1, \ldots, n\}|$.

Mechanisms. Let $\mathcal{Y} \subseteq \{0,1\}^n$ denote the set of all possible obfuscated access-pattern vectors. An access-pattern obfuscation mechanism probabilistically converts an access-pattern vector to another access-pattern vector. More precisely, an access-pattern obfuscation mechanism $\mathcal{K}$ from $\mathcal{X}$ to $\mathcal{Y}$ is a probabilistic function: $\mathcal{K} : \mathcal{X} \to \mathcal{Y}$

d-Privacy. In this paper, we leverage a generalization of differential privacy, called d-privacy [15]. The notation d in d-privacy stands for a metric. We will customize d-privacy to define the privacy guarantee of our access-pattern obfuscation mechanism by using the Hamming distance d_h.

Definition 2. *An access-pattern obfuscation mechanism $\mathcal{K} : \mathcal{X} \to \mathcal{Y}$ gives ϵd_h-privacy, iff $\forall x, x' \in \mathcal{X}$ and $\forall S \subseteq \mathcal{Y}$*

$$Pr[\mathcal{K}(x) \in S] \leq e^{\epsilon d_h(x,x')} Pr[\mathcal{K}(x') \in S]$$

Here ϵ is a privacy parameter which is a non-negative real number. Generally, the smaller ϵ, the stronger privacy guarantee and usually larger overhead. The intuition of d-privacy is that similar access patterns (e.g., x, x' with small Hamming distance) generate similar obfuscated access patterns. So from an obfuscated access pattern, it is difficult to infer its original access pattern. Note that the privacy guarantee is determined by both the privacy parameter and the distances between the access patterns to be protected. So for more diverse access patterns, smaller ϵ is needed, resulting in larger overhead.

B. Achieving ϵd_h-private APO

Consider flipping each bit of the access pattern independently. Specifically, let p denote the probability flipping bit $x_i = 1$ to $y_i = 1$, i.e., $p = \Pr[y_i = 1|x_i = 1]$; let q denote the probability flipping bit $x_i = 0$ to $y_i = 1$, i.e., $q = \Pr[y_i = 1|x_i = 0]$.

Define an obfuscation mechanism $\mathcal{K}_f$ such that, given an access pattern $x \in \mathcal{X}$, it outputs any $y \in \mathcal{Y}$ with probability

$$\Pr[\mathcal{K}_f(x) = y] = \Pr[y|x] = \prod_{i=1}^{n} \Pr[y_i|x_i]$$

where

$$\begin{array}{ll} \Pr[y_i = 1|x_i = 1] = p & \Pr[y_i = 1|x_i = 0] = q \\ \Pr[y_i = 0|x_i = 1] = 1 - p & \Pr[y_i = 0|x_i = 0] = 1 - q \end{array}$$

We enforce two constraints on p and q to make the mechanism practical:

- $\Pr[y_i = 1|x_i = 0] < \Pr[y_i = 1|x_i = 1]$: the intuition is that a non-matching shard should have a lower probability to be retrieved than a matching shard;
- $\Pr[y_i = 1|x_i = 0] < \Pr[y_i = 0|x_i = 1]$: the intuition is that a non-matching shard should have a lower probability to be flipped than a matching shard.

So we have $q < p$ and $q < 1 - p$.

Now, we prove that $\mathcal{K}_f$ achieves ϵd_h-privacy.

Theorem 1. *The access-pattern obfuscation mechanism $\mathcal{K}_f$ achieves ϵd_h-privacy, where $\epsilon = \ln \frac{p}{q}$.*

Algorithm 3: Func_APO

Input: A document shard and its keyword list (D_s, W); Privacy parameters p, q;
Output: A document shard with its obfuscated keyword list (D_s, W');
initialize $W' = \{\}$;
for $w \in \mathbf{\Delta}$ **do**
  if *($w \in W$ **and** $rand() \leq p$) **or** ($w \notin W$ **and** $rand() \leq q$)* **then**
    add w to W';
return (D_s, W')

Proof. It is equivalent to show that $\forall x, x' \in \mathcal{X}$ and $\forall y \in \mathcal{Y}$

$$\Pr[\mathcal{K}_f(x) = y] \leq e^{\epsilon d_h(x,x')} \Pr[\mathcal{K}_f(x') = y] \tag{1}$$

We have

$$\begin{aligned} \frac{\Pr[\mathcal{K}_f(x) = y]}{\Pr[\mathcal{K}_f(x') = y]} &= \frac{\prod_{i=1,\ldots,n} \Pr[y_i|x_i]}{\prod_{i=1,\ldots,n} \Pr[y_i|x'_i]} = \prod_{x_i \neq x'_i} \frac{\Pr[y_i|x_i]}{\Pr[y_i|x'_i]} \\ &\leq \prod_{x_i \neq x'_i} \max\left\{\frac{p}{q}, \frac{q}{p}, \frac{1-p}{1-q}, \frac{1-q}{1-p}\right\} \\ &= \left(\frac{p}{q}\right)^{d_h(x,x')} = e^{\ln \frac{p}{q} d_h(x,x')} \end{aligned}$$

Hence, we have inequality (1). □

Algorithm 3 shows the algorithm of access-pattern obfuscation for a single document shard.

V. Parameter Optimization

In this section, we discuss how to select parameters m, k, p, q. We model it as a optimization problem. Here we consider the average fraction v of keywords (with regards to all queryable keywords in the SSE scheme) contained in each document as a constant. The average fraction of documents that will be returned for each query is also v if the query is chosen uniformly at random, since each document will be returned with probability v for containing that query keyword.

The measurements to be considered are:

- *Privacy budget*: $T_1 = \epsilon m = m \ln \frac{p}{q}$. Theorem 1 shows that $\epsilon = \ln \frac{p}{q}$ for p, q such that $p > q$ and $p + q < 1$. And due to the use of erasure codes, the Hamming distances between the access patterns is proportional to m. This should be reflected in the privacy budget as a multiplier.
- *Document storage overhead*: $T_2 = \frac{m}{k}$. Each document is encoded into m shards, each has $\frac{1}{k}$ of the original's size, resulting in $\frac{m}{k}$ times overhead for document storage.
- *Index storage overhead*: $T_3 = (p + (\frac{1}{v} - 1)q)m$. The number of keyword lists is m times of the original. For each keyword list, each keyword in it has a probability of p to appear in its obfuscated list, and any other keyword will be added with a probability of q. So any keyword is contained in a obfuscated keyword list with expectation $vp + (1 - v)q$. Dividing by v and multiplying by m, we get the storage overhead for the index, T_3.
- *Communication overhead*: $T_4 = (p + (\frac{1}{v} - 1)q)\frac{m}{k}$. The number of shards retrieved is proportional to the total number of keywords in the index. Each shard is $\frac{1}{k}$ of the original document's size. So the communication overhead is T_4 times of the original.

- *Recall*: $T_5 = \sum_{i=k}^{m} C_m^i p^i (1-p)^{m-i}$. This is the probability, for any matching document, to have at least k shards returned. The probabilities of recoverability are independent for different documents. Hence, the expected recall, i.e., the percentage of successfully decoded matching documents is T_5.
- *Precision*: $T_6 = \frac{vT_5}{vT_5 + (1-v)\sum_{i=k}^{m} C_m^i q^i (1-q)^{m-i}}$. Similarly, we can get the probability that a non-matching document is successfully decoded, $\sum_{i=k}^{m} C_m^i q^i (1-q)^{m-i}$. Weighted by the percentage of matching documents and non-matching documents, we have the precision T_6.

We can see that these six measurements are all nonlinear functions in m, k, p, q which is undesirable for optimization problems. Note that for erasure coding, the values of m, k are always small integers (≤ 256 for the open-source erasure-coding implementation used in our implementation). So it is feasible to enumerate the values of m, k, such that $0 < k < m < 256$, and optimize p, q. This makes measurements T_2, T_3, T_4 linear functions in p, q.

For the privacy budget T_1, we propose to set a constant threshold ϵ_0 that indicates the maximal allowable leakage. So we have a constraint $T_1 \leq \epsilon_0$, which can be further translated into a linear constraint for p, q:

$$p \leq e^{\frac{\epsilon_0}{m}} q$$

For the recall T_5, we propose to set a lower bound, or minimal required recall τ_0, which should be as close to 1 as possible. So we have a constraint $T_5 \geq \tau_0$. However, given m, k, T_5 is still a nonlinear function of p, denoted as $T_5(p)$. We found that $T_5(p)$ is monotonically increasing with p when $p \geq \frac{k}{m}$, by proving its counterpart $1 - T_5 = \sum_{i=0}^{k-1} C_m^i p^i (1-p)^{m-i}$ is monotonically decreasing with p, which can be easily proved by showing the first derivative of each term is negative when $p \geq \frac{k}{m}$. Hence, we can use binary search to find p^* such that $T_5(p^*) \approx \tau_0$, so when $p \geq p^*$, $T_5(p) \geq \tau_0$ is always satisfied. Note that $p \geq \frac{k}{m}$ implies $mp \geq k$, where mp is the expected number of returned shards of a matching document, and k is the minimal number of shards for successful decoding. This constraint is reasonable for a high recall. Hence, with a minimal required recall τ_0, we derive two linear constraints for p:

$$p \geq \frac{k}{m} \qquad p \geq p^*$$

For precision T_6, we exclude it from our optimization since the false positive documents could be easily filtered out by the client. We will give the evaluation results on precision in Section VII-B for completeness.

Now we can formulate the optimization problem for selecting m, k, p, q: Minimize

$$\omega_1 \frac{m}{k} + \omega_2 \left(p + \left(\frac{1}{v} - 1\right) q\right) m + \omega_3 \left(p + \left(\frac{1}{v} - 1\right) q\right) \frac{m}{k}$$

subject to

$$\begin{array}{llll} p \leq e^{\frac{\epsilon_0}{m}} q & p > q & p + q < 1 & 0 < k < m < 256 \\ p \geq p^* & p \geq \frac{k}{m} & 0 < p, q < 1 & \end{array}$$

where $\omega_i, i = 1, 2, 3$, s.t., $\omega_1 + \omega_2 + \omega_3 = 1$ are weights for each specific measurement.

The goal is to minimize the weighted sum of different types of overhead, given privacy and recall requirements. We found that for given m, k, the minimum can be reached by setting $p = p^*$ and $q = p^* e^{-\frac{\epsilon_0}{m}}$. So we can enumerate values of m, k, calculate p, q and corresponding weighted overhead sum, and output the values of m, k, p, q that achieve the minimum weighted overhead sum.

Interestingly, in the first optimization attempts, we found that the optimal p was very close to 1. The reason might be that given a recall requirement, larger p allowed smaller m and $\frac{m}{k}$, resulting in smaller overhead. However, such p makes the obfuscated access pattern very close to the original access pattern, leading to quite limited effectiveness of our obfuscation mechanism. Hence, we additionally added a upper bound to p, i.e., $p \leq \hat{p}$. In our evaluation, we set $\hat{p} = 0.9$. Another observation was that the weighted sum of different types of overhead was quite consistent with most $\omega_i, i = 1, 2, 3$ settings. So we fixed their values such that $\omega_1 = 0.3, \omega_2 = 0.1, \omega_3 = 0.6$ in our evaluation.

VI. Implementation

We implemented an access-pattern obfuscation mechanism as a Java package `APOSSE`[1] to support an open source SSE library, Clusion [20], which provides Java implementations of various state-of-the-art SSE schemes. We chose the implementation *RR2Lev* of the basic SSE scheme in [21] as our underlying SSE scheme. For erasure codes, we use an open source Reed-Solomon Code implementation [22] in Java.

Clusion currently provides only local versions of SSE implementations. But it does not affect our implementation, since our proposed obfuscation is done locally in the client side by design. Clusion provides a keyword extraction method to generate a *Multimap* (a data structure from Google Guava that maps keys to values) that maps each keyword to the names of documents that contains that keyword. It also provides BuildIndex, Token and Search algorithms. Note that its Search algorithm outputs the names of the matching documents.

Specifically, we implemented a Java class `APOModules` to achieve access-pattern obfuscation. It has 3 components:

- `erasureCodeEncoding`, an encoding function that encodes each document in the collection using Reed-Solomon Code. Each shard is named by its original name followed by the shard number. For example, a document named "foo" will be encoded into shards with names "foo.0", "foo.1", ..., "foo.7" respectively, when $m = 8$.
- `obfuscateKeywordLists`, an access-pattern obfuscation function that takes in a keyword-document $multimap_1$ and generates an obfuscated keyword-document $multimap_2$. For each $(kw, doc) \in multimap_1$, it adds each $(kw, doc.i), i = 0, \ldots, m-1$, to $multimap_2$ with probability p, and for $(kw, doc) \notin multimap_1$, it adds each $(kw, doc.i)$ to $multimap_2$ with probability q.

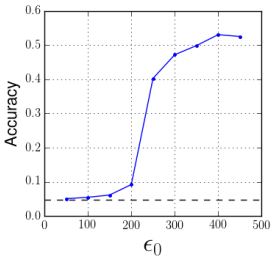
[1] https://github.com/donnod/APOSSE

- `erasureCodeDecoding`, a search result decoding function that collects shards of the query results and decodes documents with enough shards.

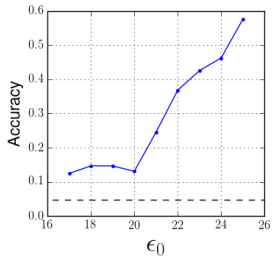
The resulting SSE scheme with access-pattern obfuscation works as follows: when the client provides a document collection, the default keyword extraction algorithm is called to generate a keyword-document $multimap_1$ and `obfuscateKeywordLists` is called to get an obfuscated keyword-document $multimap_2$. $multimap_2$ is provided to the default BuildIndex algorithm to generate a secure index. `erasureCodeEncoding` is called to produce shards of the document collection. When the client issues a query, a search token is generated using the default token generation algorithm and passed into the Search algorithm. A list of shard names is generated, which is input to `erasureCodeDecoding` to recover documents to be presented to the client.

VII. Evaluation

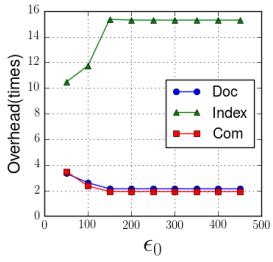
In this section, we empirically evaluate the effectiveness and efficiency of our proposed access-pattern obfuscation method.



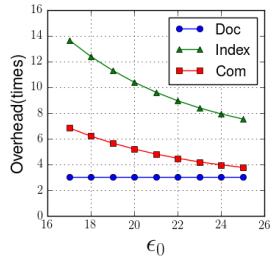
(a) Baseline IKK attack accuracy, with recall $\geq 99.99\%$



(b) Improved IKK attack accuracy, with recall $\geq 99.99\%$



(c) Storage and communication overhead , with recall $\geq 99.9999\%$ (baseline IKK attacks)



(d) Storage and communication overhead, with recall $\geq$ 99.99% (improved IKK attacks)

Fig. 4: Security and performance evaluation w.r.t. different attacks, under different privacy budget threshold ϵ_0 settings

A. Security Evaluation

We replicated the IKK attack [9] on the Enron email dataset, in which 30109 emails from "sent" folders were considered as the document collection and 500 most common words were used as keywords. We randomly chose 150 queries to be issued by the client. The content of 7 of these queries were known to the adversary (4.67%). The average percentage of keywords contained in each document is $v = 3.61\%$. Under this setting, our replicated IKK attack was able to recover 100% of the 150 queries. Next, we considered cases when our access-pattern obfuscation method was applied.

1) The Baseline IKK Attack: We first replicated the original, unmodified IKK attack on obfuscated dataset and access-patterns. The attacks were effectively mitigated with roughly $1.6\times$ of communication overhead, which is similar to the padding countermeasures implemented in [9], [10]. However, we believe these results were due to the adversary's unawareness of the existence of mitigation methods. To better evaluate our approach, we further assume the adversary knows the existence of our defenses, including the parameters, m, k, p, q. However, we assume the correlation between the shards and their original documents remains unknown. This is reasonable when the sizes of shards are the same and the encrypted shards are outsourced in random order. By simply observing the obfuscated access patterns of a limited number of queries, it is very difficult for the adversary to figure out the linkage between the shards and their original documents. We will relax this requirement in the next section.

In particular, we considered a *baseline IKK attacker* whose strategy is to use the obfuscation parameters and her *a priori* knowledge of the document collection to generate a co-occurrence matrix that could better represent the relationship between keywords and the obfuscated access patterns. Specifically, the adversary simulates the obfuscation process multiple times and computes an average co-occurrence matrix from these simulated obfuscated access patterns. With this adjusted co-occurrence matrix, the adversary continues to perform the rest of the original IKK attack.

The obfuscation parameters were selected to achieve a recall greater than 99.9999% and the minimal weighted overhead sum according to Sec. V, under privacy budget threshold ϵ_0 ranging from 50 to 450. The results of the *baseline IKK attack* against this obfuscation scheme is illustrated in Fig. 4a, where the x-axis shows various values of ϵ_0 and the y-axis shows the accuracy of query recovery. We can see that with $\epsilon_0 \leq 200$, the *baseline IKK attack* can be effectively mitigated.

2) The Improved IKK Attack: Now we consider the cases when the adversary can successfully figure out which shards belong to the same documents. This is possible when the shards of different original documents are different in size, or when the obfuscated SSE is a dynamic SSE scheme (which leaks such information when uploading a single document—thus all its shards—at the same time). The improved IKK adversary is much stronger as she knows every detail of the obfuscation (except for the random values generated to flip bits of access-pattern vectors), which enables her to infer the original access patterns and calculate a more accurate co-occurrence matrix.

Specifically, the adversary firstly groups together shards from each original document and infers the original access patterns of the document as follows: As each document is encoded into m shards with the same keyword lists, the original access pattern of the document is an m-bit vector with all 0s, denoted as x_{zeros}, or all 1s, denoted as x_{ones}. By observing the obfuscated access pattern of the shards, which is also a m-bit vector y (with b bits equal to 1), the adversary applies Bayesian estimation on the original access pattern

$x \in \{x_{zeros}, x_{ones}\}$ by comparing the posterior probabilities $\Pr[x_{zeros}|y]$ and $\Pr[x_{ones}|y]$. Using Bayesian Theorem, we have

$$\Pr[x|y] = \frac{\Pr[x]\Pr[y|x]}{\Pr[y]} = \frac{\Pr[x]\Pr[\mathcal{K}_f(x) = y]}{\Pr[y]}$$

which is equivalent to compare

$$\Pr[x_{zeros}]\Pr[\mathcal{K}_f(x_{zeros}) = y] = \quad (1-v)q^b(1-q)^{m-b}$$
$$\Pr[x_{ones}]\Pr[\mathcal{K}_f(x_{ones}) = y] = \quad vp^b(1-p)^{m-b}$$

Under the same parameter settings used to defeat the *baseline IKK attack*, the *improved IKK attacks* achieved almost 100% attack accuracy. To mitigate this attack, we lowered the recall requirement from 99.9999% to 99.99% and chose a lower privacy budget threshold ranging from 17 to 25. The resulting attack accuracy of this *improved IKK attack* is shown in Fig. 4b. We can see that with $\epsilon_0 \leq 20$, the attack accuracy can be reduced greatly.

B. Performance Evaluation

We evaluate the performance of our mechanism in three aspects. First, we calculate the storage and communication overhead under different parameter settings. Second, we report the precision of the retrieved documents. Third, we measure the latency of the SSE due to access-pattern obfuscation.

1) Storage and Communication Overhead: The document storage, index storage and communication overhead under different parameter settings of both the *baseline IKK attack* and the *improved IKK attacks* (with the corresponding recall) are shown in Fig. 4c and Fig. 4d, respectively. In Fig. 4c, $m = 10, k = 3$ for $\epsilon_0 = 50$; $m = 13, k = 5$ for $\epsilon_0 = 100$; and $m = 17, k = 8$ for $\epsilon_0 \geq 150$. p ranges from 0.886 to 0.899 and q ranges from 10^{-2} to 10^{-12}. To defeat the *baseline IKK adversary*, roughly $2\times$ of original communication workload is enough. In Fig. 4d, $m = 6, k = 2$ for all simulated ϵ_0 values. p equals to 0.887 and q ranges from 0.052 to 0.014. To mitigate the *improved IKK adversary*, roughly $5\times \sim 6\times$ of original communication workload is needed. To defeat these two attacks while meeting the recall requirements, the document storage overhead ranges from $2\times$ to $3\times$, and the index storage overhead ranges from $10\times$ to $15\times$.

2) Precision: In the experiments for defending against the *baseline IKK attack*, the precision exceeded 99.9% for all parameter settings. For *improved IKK attack*, Fig. 5 shows the precision under different privacy budget threshold ϵ_0 settings, with the recall $\geq 99.99\%$. A precision less than 60% is needed to effectively defeat the *improved IKK attack*. But as the client sees the decoded documents in plaintext, filtering false positives on the client side is straightforward.

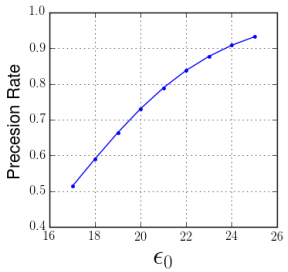


Fig. 5: Precision under different privacy budget threshold ϵ_0 settings, with recall rate $\geq 99.99\%$

TABLE I: Runtime overhead of the obfuscation mechanism

	original	bIKK	iIKK
m	N/A	17	6
k	N/A	8	2
p	N/A	0.89999	0.88703
q	N/A	6.997E-6	0.04416
# of original (kw, doc) pairs	607837	607837	607837
obfuscation in setup phase	N/A	107.97 s	103.92 s
# of resulting (kw, doc) pairs	607837	9301129	7071015
index building time	6.09 s	46.09 s	36.77 s
shards encoding time	N/A	30.87 s	9.72 s
search time per query	3.91 ms	54.51 ms	38.52 ms
decoding time per query	N/A	113.77 ms	104.12 ms

3) Runtime Overhead: We measured the induced access latency of our obfuscation scheme on a server with Intel Xeon 2.3GHz E5-2630 processors and 16GB memory. We run our implementation with Enron email dataset with two parameter settings, one for each attack (bIKK stands for the *baseline IKK attack* and iIKK stands for the *improved IKK attack*), and compared the access time with that of the original SSE schemes. The results are shown in Table I. The search time per query and decoding time per query is measured by taking the average of 150 randomly selected queries. Since our implementation of SSE is a local version, no network latency was measured. We can see that the runtime overhead for BuildIndex increases with the index storage (represented by the resulting number of keyword-document pairs). Since the setup process will be performed only once, this runtime cost can be amortized. For search phase, less than 200 milliseconds is needed to query a large secure index and decode the result.

VIII. Related Work

A. Query Recovery Attacks against SSE

Islam, Kuzu, and Kantarcioglu [9] demonstrated that an honest-but-curious server could guess the user's queries when the content of (almost) the entire database is known. Cash et al. [10] categorized the leakage into four levels, i.e., $L1, L2, L3, L4$, with $L1$ being the least leakage (e.g., leaking only the access patterns). They also improved the IKK attack to make it more efficient when additional information, i.e., *a prior* knowledge of the sizes of search results, is known. Zhang et al. [23] introduced file-injection attacks against dynamic SSE schemes, in which an active adversary injects files of her choice to the dataset and learn the user's queries by observing the access patterns of the injected files. Their attack assumes an active adversary, while our mechanism considers only a passive adversary.

More distant from the threat model we consider in this paper, Kellaris et al. [24] generalized the leakage model of access patterns and communication volumes and proposed generic reconstruction attacks on systems supporting range queries. Rather than focusing on file-based access pattern leakage, Ritzdorf et al. [25] studied block-based access pattern leakage in deduplicated storage systems.

Besides access-pattern leakage based query recovery attack, Liu et al. [26] demonstrated that query content could be recovered when the attacker observes search patterns (whether two issued queries are searching the same keyword), and

has *a priori* knowledge of the victim's search habits. In this paper, we focus on protecting access-pattern leakage. Though our current framework does not hide search patterns, the countermeasure proposed by Liu et al. [26] could be incorporated into our framework easily.

B. Countermeasures

Kuzu et al. [19] proposed to apply differential privacy to obfuscate the total size of search results. Their scheme requires two servers: an untrusted server that holds most of the data and a trusted server that holds the complementary records to ensure the correctness of search results. Laplace-distributed noise is added to the total number of retrieved records. In contrast, we achieve privacy guarantees for access-pattern leakage, a much stronger threat model, and do so without a separate, trusted server. Islam et al. [9] proposed a padding based countermeasure to mitigate the IKK attack. In their countermeasure, keywords are partitioned into sets, each of which contains at least α keywords. The access patterns of the keywords within the same set are padded to be the same. Cash et al. [10] proposed to inject additional file accesses so that the number of the accessed files is some multiple of a constant integer. Our obfuscation mechanism offers more general and principled defense than padding-based countermeasures.

Garg et al. [12] proposed an ORAM-based SSE scheme to eliminate access-pattern leakage, but with a communication cost of $O(\log(n)\log\log(n))$, which becomes very large when the number of documents, n, increases. Naveed et al. [13] also suggested that the communication overhead of ORAM-based schemes could be larger than that of simply sending back all data stored in the remote server. In contrast, our proposed obfuscation mechanism does not incur higher communication overhead with larger n.

Private information retrieval (PIR) schemes protect the access patterns to public databases [27]. Single-server PIR schemes [18] may induce lower communication overhead than ORAM, but the computation overhead is large—the entire document collection must be processed. A differentially private PIR was proposed by Toledo et al. [28], where multiple non-colluding servers were required. In contrast, our scheme does not rely on multiple servers. Moreover, PIR considers a different threat model, which has been discussed in Sec. II-D.

IX. Conclusion

Searchable symmetric encryption schemes trade off security for efficiency by allowing access-pattern leakage. In this paper, we defined ϵd_h-privacy on access patterns, and proposed an obfuscation mechanism to achieve this privacy guarantee. Instead of designing a new SSE scheme, we proposed a framework that could integrate our obfuscation mechanism into existing SSE schemes. We implemented a prototype, and evaluated the effectiveness and performance of our design.

Acknowledgement

We are grateful to the anonymous reviewers for their constructive comments. This work was supported in part by NSF grants 1330599, 1718084 and 1566444.

References

[1] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000.
[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *ACM CCS*, 2006.
[3] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *ACM CCS*, 2012.
[4] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International Conference on Financial Cryptography and Data Security*, 2013.
[5] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *CRYPTO*, 2013.
[6] M. Naveed, M. Prabhakaran, and C. Gunter, "Dynamic searchable encryption via blind storage," in *IEEE Symposium on Security and Privacy*, 2014.
[7] B. Raphael, "Sophos - forward secure searchable encryption," in *ACM CCS*, 2016.
[8] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Eurocrypt*, 2017.
[9] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." in *NDSS*, 2012.
[10] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *ACM CCS*, 2015.
[11] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, 1996.
[12] S. Garg, P. Mohassel, and C. Papamanthou, "Tworam: Efficient oblivious ram in two rounds with applications to searchable encryption," in *CRYPTO*, 2016.
[13] M. Naveed, "The fallacy of composition of oblivious ram and searchable encryption," Cryptology ePrint Archive, Report 2015/668, 2015.
[14] C. Dwork, "Differential privacy: A survey of results," in *International Conference on Theory and Applications of Models of Computation*, 2008.
[15] K. Chatzikokolakis, M. Andrs, N. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *Privacy Enhancing Technologies*, 2013.
[16] Enron dataset. https://www.cs.cmu.edu/%7E./enron/.
[17] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, 1998.
[18] R. Ostrovsky and W. Skeith, *A Survey of Single-Database Private Information Retrieval: Techniques and Applications*, 2007.
[19] M. Kuzu, M. Islam, and M. Kantarcioglu, "Efficient privacy-aware search over encrypted databases," in *ACM Conference on Data and Application Security and Privacy*, 2014.
[20] S. Kamara and T. Moataz. Clusion. https://github.com/orochi89/Clusion.
[21] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *NDSS*, 2014.
[22] Javareedsolomon. https://github.com/Backblaze/JavaReedSolomon.
[23] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *USENIX Security Symposium*, 2016.
[24] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," *ACM CCS*, 2016.
[25] H. Ritzdorf, G. Karame, C. Soriente, and S. Čapkun, "On information leakage in deduplicated storage systems," in *ACM on Cloud Computing Security Workshop*, 2016.
[26] C. Liu, L. Zhu, M. Wang, and Y.-A. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Inf. Sci.*, 2014.
[27] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Annual Symposium on Foundations of Computer Science*, 1995.
[28] R. Toledo, G. Danezis, and I. Goldberg, "Lower-cost epsilon-private information retrieval," *CoRR*, 2016, http://arxiv.org/abs/1604.00223.