Class Indistinguishability for Outsourcing Equality Conjunction Search

Weipeng Lin[1], Ke Wang[1], Zhilin Zhang[1], Ada Waichee Fu[2], Raymond Chi-Wing Wong[3], and Cheng Long[4]

[1] Simon Fraser University, Vancouver, Canada,
{weipengl,ke_wang,zhilinz}@sfu.ca
[2] Chinese University of Hong Kong, Hong Kong, China,
[3] Hong Kong University of Science and Technology, Hong Kong, China
[4] Nanyang Technological University, Singapore

Abstract. Searchable symmetric encryption (SSE) enables a remote cloud server to answer queries directly over encrypted data on a client's behave, therefore, relieves the resource limited client from complicated data management tasks. Two key requirements are a strong security guarantee and a sub-linear search performance. The bucketization approach in the literature addresses these requirements at the expense of downloading many false positives or requiring the client to search relevant bucket ids locally, which limits the applicability of the method. In this paper, we propose a novel approach CLASS to meet these requirements for equality conjunction search while minimizing the client work and communication cost. First, we generalize the standard ciphertext indistinguishability to partitioned data, called class indistinguishability, which provides a level of ciphertext indistinguishability similar to that of bucketization but allows the cloud server to perform search of relevant data and filtering of false positives. We present a construction achieving these goals through a two-phase search algorithm for a query. The first phase finds a candidate set through a sub-linear search. The second phase finds the exact query result using a linear search applied to the candidate set. Both phases are performed by the server and are implemented by plugging in existing search methods. The experiment results on large real-world data sets show that our approach outperforms the state-of-the-art.

Keywords: Searchable encryption; equality conjunction search; sub-linear search.

1 Introduction

The current trend towards cloud-based Database-as-a-Service (DaaS) as an alternative to traditional on-site relational database management systems has largely been driven by the perceived simplicity and cost-effectiveness. On one hand, the sensitive and confidential nature of data requires that outsourced data need to be stored in encrypted form to preserve the privacy. On the other hand, outsourcing encrypted data precludes the client from delegating query processing tasks that depend on plaintext data information to the remote cloud server (or server for short), thus, induces inefficiency. Apparently, sending the whole encrypted data to the client for each query is impractical for most applications that deal with a large amount of data.

A promising solution to the above problem is *searchable symmetric encryption* (SSE) that allows the server to answer search queries directly over encrypted data on a client's behave while protecting the confidentiality of plaintext data and queries, for example, in the sense of *ciphertext indistinguishability* [9][7]. Like most works [7, 5, 9], we focus on concealing plaintext data and queries but allow the disclosure of "access pattern", which refers to the set of (encrypted) records retrieved by each query as a result of granting the server the search capacity. For hiding access patterns, please refer to private information retrieval [25] and Oblivious RAM [21] techniques.

1.1 Motivation

A key challenge for SSE is dealing with two conflicting goals: a strong security guarantee, e.g., ciphertext indistinguishability (more details later), and a sub-linear search performance for computing a query. Ciphertext indistinguishability requires that the adversary cannot distinguish two *histories* of interacting with the system having the same trace that includes everything observed by the adversary during the interaction other than encrypted data and queries, such as the database size and the result size. Unfortunately, this level of indistinguishability is difficult to satisfy if we want the server to perform a sub-linear search that entails distinguishing the records that do not need to be searched from those that do.

In some practical scenarios, it suffices to maintain indistinguishability among a small number of individuals. For example, k-*anonymity* [22] ensures that each individual cannot be distinguished from $(k-1)$ other individuals, where k is a security parameter, which protects each individual by the indistinguishability within a group of size k. Another scenario is that not all individuals care about indistinguishability and indistinguishability is needed only for those who care. For example, suppose that $Alice$ and Bob care about indistinguishability between them but Cat and Dog do not, it suffices to partition the domain into three classes $g_0 = \{Cat\}, g_1 = \{Dog\}, g_2 = \{Alice, Bob\}$ and enforce ciphertext indistinguishability *within* each class. On the other hand, such class indistinguishability would allow us to prune irrelevant classes for a query, which is impossible for the standard ciphertext indistinguishability.

Enforcing class indistinguishability is not new. The *bucketization* [11][12][13] can be considered as a construction of class indistinguishability. In this approach, records in the database are partitioned into buckets (i.e., classes) according to a specified partitioning of attribute domains and the records in a bucket are retrieved using the bucket id whereas plaintext data are encrypted using traditional techniques and stored on the server. To answer a query, the client first maps the query to relevant bucket ids using a local index and submits the bucket ids to the server. The server returns encrypted data according to the received bucket ids. The client recovers the query result after decrypting returned data and filtering false positives. Sub-linear search performance is supported by retrieving only the data in relevant buckets for a query.

However, bucketization suffers from two main drawbacks. One is that the client needs to search locally for relevant bucket ids for a query, referred to as query translation processing in [12, 11, 13]. This requires additional overheads on the resource limited client for storing and maintaining the translation information (i.e., the information for all buckets) for dynamic data, but the powerful server only needs to retrieve encrypted

data through bucket ids computed by the client. Another drawback is that false positives are communicated to the client because they can only be filtered by the client. These are indicated by the boxes named "Search" and "Filtering" on the client side in Figure 1(A). A finer bucket granularity will increase client's search work due to the increased number of buckets (especially for multi-dimensional data) whereas a coarser bucket granularity will increase the communication cost increased false positives. Since client resources and network bandwidth are limited, this approach's application will be limited.

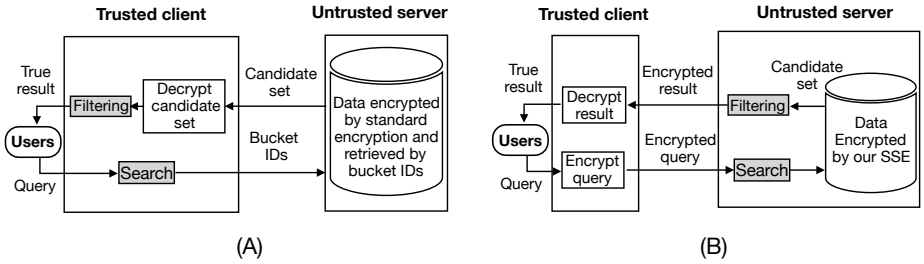

Fig. 1: (A) Bucketization [11][12][13]: The client searches for relevant bucket ids, the server returns all records in the buckets, and the client filters false positives; (B) Proposed scheme: The client encrypts the query predicate, the server searches for a candidate set and filters false positives, and the client decrypts the query result

1.2 Contributions

A preferred solution is pushing the "Search" and "Filtering" tasks to the server as in Figure 1(B) where search for the relevant buckets and filtering of false positives are done by the server; the client only needs to encrypt the query and decrypt the query result. This approach calls for a new encryption scheme that would enable the server to perform search and filtering tasks. In this work, we present a novel scheme called CLASS to meet these requirements. We consider a relational database $\mathcal{D} = \{P_1, \cdots, P_{|\mathcal{D}|}\}$ containing $|\mathcal{D}|$ records with d attributes $\{A_1, \cdots, A_d\}$, where each A_t has a discrete domain $dom(A_t)$. A numeric domain can be discretized into a small number of intervals. We consider *equality conjunction queries* containing one or more equalities $A_t = v$ with $v \in dom(A_t)$, and $Att(Q)$ denotes the set of attributes on which a query Q has an equality. Each record has a unique record ID and $RID(\mathcal{D}, Q)$ denotes IDs for the records in $\mathcal{D}$ satisfying a query Q. Note that the database may contain other attributes that do not occur in any query. Our contributions are as follows.

- (Section 3) We formalize a relaxed notion of ciphertext indistinguishability, called *class indistinguishability*, that achieves a level of indistinguishability similar to that of bucketization.
- (Section 4) We propose a novel SSE scheme, called CLASS. CLASS is the first SSE scheme for equality conjunction queries that meets class indistinguishability and supports sub-linear search while pushing search and filtering tasks to the server

as in Figure 1(B). CLASS can be implemented by plugging in existing search methods without designing specialized methods.
- (Section 5) We formally prove class indistinguishability of CLASS.
- (Section 6) We present an empirical study to evaluate the practical efficiency of CLASS on large and real life databases. Our results show that CLASS outperforms the state-of-the-art.

2 Related Work

This work is at the intersection of cryptography (for formal security notion) and database (for high performance query computing).

Cryptography. Most works on SSE consider single keyword queries and a linear search [4]. [2][10] pioneered the construction of conjunctive keyword search with a linear search. A few recent works consider sub-linear search for conjunctive keyword search, for example, [5][15]. These schemes relax the notion of ciphertext indistinguishability by capturing certain disclosures (using a leakage function) caused by a sub-linear search process. One problem with these approaches is that it is a daunting task to capture the full extent of such low-level disclosures that are specific to the design of the index structure and the sub-linear search algorithm. In fact, the real-world consequences of such low-level disclosures are poorly understood, which was highlighted as an important open question [26][7]. Conjunctive keyword search is also studied based on Hidden Vector Encryption (HVE) [16], which suffers from prohibitive computation and communication costs.

Database. The research in database traditionally focused on scalability for large databases by adopting ad hoc security definitions. Examples are order preserving encryption [3] and distance preserving encryption [18], which makes indexing easy but discloses order and proximity information of plaintext. CryptDB [19] enables the DBMS server to execute SQL queries on encrypted data, using deterministic encryption for equality checks, group by, and equality-joins, and order preserving encryption for order checks. It is well known that deterministic encryption does not provide sufficient protection in practice. Asymmetric scalar-product preserving encryption (ASPE) [24] is suitable for designing a sub-linear search algorithm for equality conjunction search, but can not provide sufficient security [17]. Bucketization [11][12][13] provides a trade-off solution to security and sub-linear search performance. As discussed out in Section 1.1, this approach requires either significant client work or a high communication cost.

3 Proposed Security

In this section, we propose the notion of class indistinguishability. We consider the "honest-but-curious" adversary (i.e., the server) who follows all protocols honestly but may passively attempt to learn the plaintext information.

3.1 Classes

We assume that for an attribute A_t ($1 \leq t \leq d$), the domain of A_t is partitioned into l_t disjoint *value classes*, $\{g_0^t, \cdots, g_{l_t-1}^t\}$. Typically, the class partitioning for each at-

tribute is specified by the data owner and is public. In the following definition, we assume that the class partitioning for each attribute is given; we will discuss the specification of class partitioning after the definition. Given the class partitioning for every attribute, we can define the classes of records, queries, databases, and histories naturally as follows.

Definition 1. *Let* $\{g_0^t, \cdots, g_{l_{t-1}}^t\}$ *be the class partitioning for* A_t, $1 \leq t \leq d$.

- *A* record class *is a set of records* $\prod_{t=1}^{d} g_j^t$, *where* g_j^t *is a value class for* A_t. *In other words, a record class consists of all records* P_i *whose* $P_i[t]$*s are in the same value class for every attribute* A_t.
- *A* database class *consists of all databases* $\mathcal{D}$ *such that for any database* $\mathcal{D}'$ *in the class, there is a bijection* η *from* $\mathcal{D}$ *to* $\mathcal{D}'$ *such that for each record* P_i *in* $\mathcal{D}$, P_i *and* $\eta(P_i)$ *are in the same record class.*
- *A* query class *consists of all queries* Q *such that for any query* Q' *in the class,* $Att(Q) = Att(Q')$ *and for each* $A_t \in Att(Q)$, $Q[t]$ *and* $Q'[t]$ *are in the same value class.*
- *A* history class *consists of all histories* $H = (\mathcal{D}, \mathcal{Q} = \{Q_1, \cdots, Q_m\})$ *such that for any history* $H' = (\mathcal{D}', \mathcal{Q}' = \{Q'_1, \cdots, Q'_m\})$ *in the class,* $\mathcal{D}$ *and* $\mathcal{D}'$ *are in the same database class, and for* $1 \leq j \leq m$, Q_j *and* Q'_j *are in the same query class.*□

Intuitively, a database class consists of all databases obtained by replacing each record with a record from the same record class; a query class consists of all queries obtained by replacing each specified value with a value from the same value class; a history class consists of all histories obtained by replacing the database with a database from the same database class and replacing each query with a query from the same query class. One extreme case is a singleton record class that contains a single record, which corresponds to a singleton value class on every attribute. Another extreme case is that there is a single class that contains all domain values for every attribute. In this case, all records (databases, queries, and histories) belong to the same class. Our class indistinguishability (Definition 3) ensures ciphertext indistinguishability for the members from the same class. Therefore, the size of value classes becomes a security parameter because a larger size leads to more members in a class that are indistinguishable from one another. The data owner can specify class partitioning through the size of value classes for each attribute. In this case, any grouping of value classes of the specified size suffices. In other cases, the data owner may want to group certain domain values into the same class, which can be done by enumerating the domains values in each value class.

In the rest of discussion, whenever it is clear from the context, we use the term "class" for any of record class, database class, query class, and history class.

3.2 Class Indistinguishability

Simply put, class indistinguishability is ciphertext indistinguishability of any two histories from the the same class. Formally, we can define this notion by a probabilistic game (or experiment) between an adversary and a challenger. We first borrow the definition of SSE from [7] as follows.

Definition 2. *(Symmetric Searchable Encryption (SSE) [7]) A SSE scheme is a collection of four polynomial-time algorithms* $(KeyGen, Enc, Enc_q, Search)$ *such that,*

- $K \leftarrow KeyGen(1^k)$: *a probabilistic algorithm run by the client to initialize the secret key* K *for the scheme. The input is a security parameter* k.
- $(\mathbf{I}, \mathbf{c}) \leftarrow Enc(K, \mathcal{D})$: *a probabilistic algorithm run by the client to encrypt the database* $\mathcal{D} = \{P_1, \cdots, P_{|\mathcal{D}|}\}$. *The output is the ciphertexts* $\mathbf{c}$ *of records and the encrypted structure* $\mathbf{I}$ *for query testing.*
- $\mathbf{t} \leftarrow Enc_q(K, Q)$: *an algorithm run by the client to encrypt a query* Q. *The output is called the trapdoor for query testing.*
- $X \leftarrow Search(\mathbf{I}, \mathbf{t})$: *a deterministic algorithm run by the server to search for the records in* $\mathcal{D}$ *that satisfy* Q *with the input* $\mathbf{I}$ *and* $\mathbf{t}$. *The output is a set of record IDs.*

A SSE scheme is correct if for all $k \in \mathbb{N}$, *for all* K *output by* $KeyGen(1^k)$, *for all* $\mathcal{D} \subseteq \prod_{t=1}^{d} dom(A_t)$, *for all* $\mathbf{I}$ *output by* $Enc(K, \mathcal{D})$, *for all* $\mathbf{t}$ *output by* $Enc_q(K, Q)$, *the output of* $Search(\mathbf{I}, \mathbf{t})$ *is the set of IDs for the records in* $\mathcal{D}$ *satisfying* Q. □

One common technique for defining ciphertext indistinguishability is the probabilistic game [7]. We adopt this technique for defining class indistinguishability. Consider a history $H = (\mathcal{D}, \mathcal{Q} = \{Q_1, \cdots, Q_m\})$, which specifies a database and a sequence of queries. The *access pattern* induced by H is the tuple $\alpha(H) = (RID(\mathcal{D}, Q_1), \cdots, RID(\mathcal{D}, Q_m))$. The *search pattern* induced by H is the symmetric binary matrix $\sigma(H)$ such that for $1 \leq i, j \leq m$, the element in the i-th row and j-th column is 1 if $Q_i = Q_j$, and 0, otherwise. The *trace* induced by H is $\tau(H) = (|\mathcal{D}|, \alpha(H), \sigma(H))$. Two histories H_0 and H_1 have the same trace if there is a renaming ρ of RIDs such that $|\mathcal{D}_0| = |\mathcal{D}_1|$, $\alpha(H_0) = \rho(\alpha(H_1)), \sigma(H_0) = \sigma(H_1)$. For $b \in \{0, 1\}$, let $|\mathcal{D}_b| = n$, $\mathcal{D}_b = \{P_{b,1}, \cdots, P_{b,n}\}$ and let $\mathcal{Q}_b = \{Q_{b,1}, \cdots, Q_{b,m}\}$.

Definition 3. (Class indistinguishability) *Assume that the class partitioning* $\{g_0, \cdots, g_{l-1}\}$ *is given for every attribute* A_t. *Let SSE* $= (KeyGen, Enc, Enc_q, Search)$ *and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. Consider the following probabilistic experiment:*

$\boldsymbol{Ind}_{SSE,\mathcal{A}}(k)$

1. $K \leftarrow KeyGen(1^k)$
2. $(st_{\mathcal{A}}, H_0, H_1) \leftarrow \mathcal{A}_1(1^k)$
3. $b \xleftarrow{\$} \{0, 1\}$
4. *parse* H_b *as* $(\mathcal{D}_b, \mathcal{Q}_b)$
5. *for* $1 \leq i \leq n$
6. $\quad \mathbf{s}_{b,i} \leftarrow Enc(K, P_{b,i})$
7. *let* $\mathbf{I}_b = (\mathbf{s}_{b,1}, \cdots, \mathbf{s}_{b,n})$
8. *for* $1 \leq j \leq m$
9. $\quad \mathbf{t}_{b,j} \leftarrow Enc_q(K, Q_{b,j})$
10. *let* $\mathbf{t}_b = (\mathbf{t}_{b,1}, \cdots, \mathbf{t}_{b,m})$
11. $b' \leftarrow \mathcal{A}_2(st_{\mathcal{A}}, \mathbf{I}_b, \mathbf{t}_b)$
12. *if* $b' = b$, *output 1*
13. *otherwise output 0*

subject to two restrictions: (i) H_0 *and* H_1 *have the same trace, (ii)* H_0 *and* H_1 *are from the same class.* $st_{\mathcal{A}}$ *is a string that captures* $\mathcal{A}_1$*'s state after choosing the plaintext. We say that SSE ensures* **class indistinguishability** *if for all polynomial-size adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$|\Pr[\boldsymbol{Ind}_{SSE,\mathcal{A}}(k) = 1] - \frac{1}{2}| \leq negl(k) \tag{1}$$

where the probability is taken over the choice of b and the coins of KeyGen, Enc, Enc_q. We say that SSE ensures **strict class indistinguishability** *if*

$$\Pr[\textit{Ind}_{SSE,\mathcal{A}}(k) = 1] = \frac{1}{2}.\square \tag{2}$$

In the above game, the adversary chooses two histories H_0 and H_1 (line 2), and the challenger makes a choice $b \in \{0, 1\}$ uniformly at random (line 3) and encrypts the data and queries in H_b and return the result $\mathbf{I}_b$ and $\mathbf{t}_b$ to the adversary (lines 4-10). The adversary then guesses the value of b based on $\mathbf{I}_b$ and $\mathbf{t}_b$ (lines 11-13). $\Pr[\mathbf{Ind}_{\text{SSE},\mathcal{A}}(k) = 1]$ is the probability of the correct guess. Eqn (1) states that this probability is negligibly different from $\frac{1}{2}$, and Eqn (2) states that the adversary's guess is a random guess. Note the difference from ciphertext indistinguishability in [7]: the additional condition (ii) restricts H_0 and H_1 to be from the same history class, therefore, the indistinguishability is required only for the members from the same history class.

Remark 1. Class indistinguishability ensures that any two histories from the same class cannot be distinguished by the server given their ciphertexts and the search result (captured by traces). The standard ciphertext indistinguishability is the extreme case of a single value class containing all domain values of A_t for every attribute A_t, which produces the maximum number of histories in the class, thus, the maximum level of indistinguishability. This extreme class partitioning would lead to ineffective pruning in computing queries because the single class contains all records is relevant to every query. Class indistinguishability offers a trade-off between the level of indistinguishability and the effectiveness of sub-linear search through the specification of a more general class partitioning for each attribute, because classes containing no query result will not be searched.

The above definition considers a non-adaptive adversary in that all queries are chosen by the adversary before receiving any encrypted data or queries. In Section 5, we will show that our approach achieves class indistinguishability for an adaptive adversary as well.

4 Construction

In this section, we construct CLASS to meet two important goals: achieve class indistinguishability, and support a sub-linear search for equality conjunction queries through pushing the tasks of searching for relevant data and filtering false positives to the server as in Figure 1(B).

4.1 Overview

At the high level, CLASS consists of two SSEs: $SSE_s = (KeyGen_s, Enc_s, Enc_{q,s}, Search_s)$ for $s \in \{1, 2\}$. The client encrypts each record P_i in $\mathcal{D}$ as $Enc(P_i) = (Enc_1(P_i), Enc_2(P_i))$ and uploads $Enc(\mathcal{D})$, i.e., the collection of $Enc(P_i)$, to the server. $Enc_1(\mathcal{D})$ and $Enc_2(\mathcal{D})$ denote the projection of $Enc(\mathcal{D})$ onto $Enc_1(P_i)$ and $Enc_2(P_i)$, respectively. At the query time, the client encrypts a query Q_j into $Enc_q(Q_j) = (Enc_{q,1}(Q_j), Enc_{q,2}(Q_j))$ and submits $Enc_q(Q_j)$ to the server.

Algorithm 1 *Search*$(Enc_{q,1}(Q_j), Enc_{q,2}(Q_j))$

Require: The server has *Enc*$(\mathcal{D})$

Candidate Phase:

Cand $\leftarrow$ *Search*$_1$(*Enc*$(\mathcal{D})$, *Enc*$_{q,1}(Q_j)$)

Filtering Phase:

Results $\leftarrow$ *Search*$_2$(*Cand*, *Enc*$_{q,2}(Q_j)$)

The search for the query answer proceeds in two phases given in Algorithm 1. In the ***Candidate Phase***, a sub-linear time *Search*$_1$ is applied to *Enc*$(\mathcal{D})$ to compute a candidate set *Cand* that contains all records from the classes relevant to the query. This phase prunes all classes irrelevant to the query. In the ***Filtering Phase***, a linear time *Search*$_2$ is applied to *Cand* to filter false positives. The precision based (i.e., false positive free) *Search*$_2$ is expensive but is applied to the small candidate set *Cand*. These two phases correspond to the Search and Filtering in Figure 1(B), respectively. With a small *Cand*, any existing SSE with a linear search such as [10][9] can serve as *SSE*$_2$. Therefore, our discussion below focuses on the construction of *SSE*$_1$.

4.2 Construction of *SSE*$_1$

Consider the class partitioning $\{g_0, \cdots, g_{l-1}\}$ for an attribute A_t, $1 \leq t \leq d$, where the domain values in each class g_y are arranged in any order. The intuition of our encryption scheme is modeling the equivalence of the domain values in the same class g_y by encoding each domain value into an angle and by exploiting the periodicity of circular functions sin and cos over such angles. Let v be the domain value v at the x-th position in the class g_y. We encode v by the angle computed by

$$\alpha(v) = y\frac{\pi}{l} + (x-1)\pi \tag{3}$$

where $1 \leq x \leq |g_y|$ and $0 \leq y \leq l-1$. In other words, the class label y determines the initial angle $y\frac{\pi}{l}$ for the class and each next value in the class adds an additional angle π. To compute $\alpha(v)$, we need to choose an assignment of class labels to classes and the order of values in a class, but any such assignment and order will do. The next lemma follows because any two values from the same class have the same first term $y\frac{\pi}{l}$.

Lemma 1. *For any two values* (v, v') *in the domain of* A_t, $(\alpha(v) - \alpha(v'))$ *is a multiple of* π *if and only if* v *and* v' *are from the same class of* A_t.

Below, we construct each component of *SSE*$_1$. *KeyGen*$_1(1^{k_1})$ outputs the secret key $K_1 = M$, where M is a $(2d \times 2d)$ invertible matrix (i.e., $M^{-1}M$ is equal to the $(2d \times 2d)$ identity matrix) randomly chosen. The key size k_1 is implicitly specified by the data dimensionality d. If necessary, dummy attributes can be added to increase d. For simplicity, we omit K_1 in the following discussion.

Algorithm 2 $Enc_1(P_i)$

Require: The client has the secret key (M)

1. for $1 \leq t \leq d$
   (a) $\epsilon_{t,i} \xleftarrow{\$} [-U, -L] \cup [L, U], (0 < L \leq U)$
   (b)
$$I_i[t]_1 = \epsilon_{t,i} \sin(\alpha_t(P_i[t])) \ , \ I_i[t]_2 = \epsilon_{t,i} \cos(\alpha_t(P_i[t])) \tag{4}$$
2. $I_i = (I_i[1]_1, I_i[1]_2, \cdots, I_i[d]_1, I_i[d]_2)$
3.
$$Enc_1(P_i) = \frac{M^{-1} I_i}{|M^{-1} I_i|} \tag{5}$$

Algorithm 3 $Enc_{q,1}(Q_j)$

Require: The client has the secret key (M), $Att(Q_j) \neq \varnothing$

1. for $1 \leq t \leq d$
   (a) if A_t is in $Att(Q_j)$
      (i) $\mu_{t,j} \xleftarrow{\$} [-U, -L] \cup [L, U], (0 < L \leq U)$
      (ii)
$$T_j[t]_1 = \mu_{t,j} \cos(\pi - \alpha_t(Q_j[t])) \ , \ T_j[t]_2 = \mu_{t,j} \sin(\pi - \alpha_t(Q_j[t])) \tag{6}$$
   (b) if A_t is not in $Att(Q_j)$
$$T_j[t]_1 = T_j[t]_2 = 0$$
2. $T_j = (T_j[1]_1, T_j[1]_2, \cdots, T_j[d]_1, T_j[d]_2)$
3.
$$Enc_{q,1}(Q_j) = \frac{M^T T_j}{|M^T T_j|} \tag{7}$$

Data encryption. The detail of $Enc_1(P_i)$ is presented in Algorithm 2. Step 1 encodes each entry $P_i[t]$ into a pair $(I_i[t]_1, I_i[t]_2)$, where $\alpha(P_i[t])$ is the angle in Eqn (3) and $\epsilon_{t,i}$ is a noise randomly sampled from $[-U, -L] \cup [L, U]$ for t and i, $0 < L \leq U$. This (t, i)-specific noise is chosen independently for each record and each attribute. Step 2 assembles such pairs into a randomized $2d$-dimensional vector I_i. Step 3 blends all dimensions together using the private matrix M and produces $Enc_1(P_i)$ as a point on the $2d$-dimensional unit sphere centered at the origin. Note that the location of the point is randomized by the random noise $\epsilon_{t,i}$.

Query encryption. Algorithm 3 gives the details for $Enc_{q,1}(Q_j)$. Step 1 encodes each specified $Q_j[t]$ into a pair $(T_j[t]_1, T_j[t]_2)$ using the angle $(\pi - \alpha_t(Q_j[t]))$, and encodes each unspecified $Q_j[t]$ into $(0, 0)$. Step 2 creates a randomized $2d$-dimensional vector T_j and Step 3 blends all dimensions together and produces $Enc_{q,1}(Q_j)$ as a randomized point on the $2d$-dimensional unit sphere centered at the origin.

Search function. $Search_1$ computes the *candidate set* of the query Q_j, denoted by $Cand(Q_j)$, as the set of $Enc_2(P_i)$ such that $(Enc_1(P_i), Enc_2(P_i))$ is in $Enc(\mathcal{D})$ and

$P_i[t]$ is in the same class as $Q_j[t]$ for every $A_t \in Att(Q_j)$. *Cand*(Q_j) contains the query result and possibly false positives. The next lemma gives the computation of *Cand*(Q_j). By "P_i is in *Cand*(Q_j)", we mean "*Enc*$_2(P_i)$ is in *Cand*(Q_j)".

Lemma 2. *If P_i is in Cand(Q_j), Enc$_{q,1}(Q_j)^T$Enc$_1(P_i) = 0$. If P_i is not in Cand(Q_j), Enc$_{q,1}(Q_j)^T$Enc$_1(P_i) = 0$ holds with an exceedingly small probability.*

Proof. From Eqns (5) and (7), we have

$$\textit{Enc}_{q,1}(Q_j)^T \textit{Enc}_1(P_i) = \frac{T_j^T I_i}{|M^T T_j||M^{-1} I_i|} \tag{8}$$

where the superscript T denotes a transpose operation. *Enc*$_{q,1}(Q_j)^T$*Enc*$_1(P_i) = 0$ holds if and only if $T_j^T I_i = \sum_{t=1}^{d}(I_i[t]_1 T_j[t]_1 + I_i[t]_2 T_j[t]_2) = 0$. Since $T_j[t]_1 = T_j[t]_2 = 0$ for all A_t which are not in $Att(Q_j)$, from Eqns (4) and (6), we have

$$T_j^T I_i = \sum_{A_t \in Att(Q_j)} \epsilon_{t,i}\mu_{t,j} \sin(\Delta_t) \tag{9}$$

where $\Delta_t = (\pi + \alpha_t(P_i[t]) - \alpha_t(Q_j[t]))$. If P_i is in *Cand*(Q_j), $P_i[t]$ and $Q_j[t]$ are in the same class for every $A_t \in Att(Q_j)$, so Δ_t is a multiple of π (Lemma 1) and $\sin(\Delta_t) = 0$. In this case, *Enc*$_{q,1}(Q_j)^T$*Enc*$_1(P_i) = 0$ holds. If P_i is not in *Cand*(Q_j), $P_i[t]$ and $Q_j[t]$ are not in the same class for some $A_t \in Att(Q_j)$, and Δ_t is not a multiple of π (Lemma 1), so $\sin(\Delta_t) \neq 0$. In this case, the chance that $T_j^T I_i = 0$ holds in a small probability because noises ϵ's and μ's are randomly chosen. □

From Lemma 2, the server can compute *Cand*(Q_j) by computing the *hyperplane query* defined by *Enc*$_{q,1}(Q_j)^T V = 0$ for a $2d$-dimensional point V. Therefore, computing the candidate set is transformed into a hyperplane query in the ciphertext space, which enables any existing sub-linear methods for hyperplane queries to be deployed by the server, such as R-Tree [20], M-Tree [6] and halfspace queries [23]. As these methods are well studied, we do not further discuss their details.

Remark 2. It is interesting to compare our approach with the bucketization approach [11][12][13]. Our candidate set is similar to the result retrieved using the bucket ids of the query in bucketization. The difference is that bucketization requires the client to perform local search of bucket ids for a query, whereas the client in our approach only needs to encrypt the query. Bucketization requires the client to filter false positives, whereas our approach filters false positives by the server (through *SSE*$_2$). Finally, bucket ids in bucketization are static, thus, directly tell what records are in the same bucket, whereas our encryption functions are probabilistic thanks to fresh random noises for each encryption.

4.3 Constructing Class Partitioning

While we expect that the class partitioning $\mathcal{X}_t = \{g_0, \cdots, g_{l-1}\}$ for an attribute A_t is specified by the data owner, the class partitioning can also be constructed to minimize

a cost metric for a given class size $|g_y|$, $1 \leq y \leq l-1$, which is useful if the data owner has no preference except that each class must have a minimum size. Below, we give a construction of $\mathcal{X}_t = \{g_0, \cdots, g_{l-1}\}$ to minimize the number of false positives in the candidate set, thus, the search cost of the linear time $Search_2$.

The cost metric is minimized with respect to a chosen query workload. For simplicity, we consider only queries with a single equality. For each attribute A_t, the query workload is denoted by $\{Q_1, \cdots, Q_{|A_t|}\}$ where Q_j, $1 \leq j \leq |A_t|$, denotes the query with the single equality $A_t = v_j$. We assume that the frequency for Q_j, $1 \leq j \leq |A_t|$, denoted by f_j, is known. Let O_j, $1 \leq j \leq |A_t|$, be the number of records in the database $\mathcal{D}$ having $A_t = v_j$. Consider a value class $g_y = \{v_1, \cdots, v_\kappa\}$ for A_t. For a query Q_j, all records having a value $v_k \in g_y - v_j$ are false positives, so the cost of false positives is $Cost(g_y, Q_j) = \Sigma_{v_k \in g_y - v_j} O_k f_j$ (recall that each false positive is returned f_j times). The cost of false positives related to g_y for all queries is $Cost(g_y) = \sum_{v_j \in g_y} Cost(g_y, Q_j)$, and the cost of all false positives is $Cost(\mathcal{X}_t) = \Sigma_{y=0}^{l-1} Cost(g_y)$.

Definition 4 (Optimal κ-sized class partitioning). *Given a class size $\kappa > 1$ such that $|A_t|$ is divisible by κ and $l = \frac{|A_t|}{\kappa}$, $(O_1, \cdots, O_{|A_t|})$ and $(f_1, \cdots, f_{|A_t|})$ specified above, find a class partitioning for the attribute A_t, $\mathcal{X}_t = \{g_0, \cdots, g_{l-1}\}$, such that $Cost(\mathcal{X}_t)$ is minimized and all g_y have the size κ.*

This problem can be solved as an instance of the following *r-way equipartition problem* for which a branch-and-cut algorithm exists [14]: divide the vertices of a weighted graph $G = (V, E)$ into r equally sized sets, so as to minimize the total weight of edges that have both endpoints in the same set. To solve the optimal class partitioning problem, we can define the graph $G = (V, E)$ as follows: $V = \{1, \cdots, |A_t|\}$ and $E = \{(i, j) \mid 1 \leq i < j \leq |A_t|\}$, where for each edge $(i, j) \in E$, the weight $w_{(i,j)} = O_i f_j + O_j f_i$. Let $r = l = \frac{|A_t|}{\kappa}$. Intuitively, $w_{(i,j)}$ is the total number of false positives for queries Q_i and Q_j if i and j are grouped into the same class. It can be shown that $\mathcal{X}_t = \{g_0, \cdots, g_{l-1}\}$ is an optimal κ-sized class partitioning if and only if $\mathcal{X}_t$ is an optimal solution to the r-way equipartition problem for $G = (V, E)$.

5 Security Analysis

We formally prove that $SSE = (SSE_1, SSE_2)$ presented in Section 4 achieves *class indistinguishability* (Definition 3). In other words, the adversary can not win the probabilistic game defined in Definition 3 with significantly greater probability than an adversary who must guess randomly. Intuitively, this is achieved by the same probability of the records (queries) from the same class given the observed ciphertext of a record (query) produced by SSE_1 (as shown in Lemma 3) so that the adversary can not distinguish two histories in the probabilistic game which are restricted to the same history class.

Lemma 3. *(i) For any $2d$-dimensional vector V, $\Pr[Enc_1(P_i) = V] = \Pr[Enc_1(P'_i) = V]$ holds for any records P_i and P'_i from the same record class. (ii) For any $2d$-dimensional vector V, $\Pr[Enc_{q,1}(Q_j) = V] = \Pr[Enc_{q,1}(Q'_j) = V]$ holds for any queries Q_j and Q'_j from the same query class.*

Proof. We give a brief proof for (i) only; the proof of (ii) is similar. Since P_i and P'_i are from the same class, in Eqn (4), each $\alpha(P_i[t])$ and $\alpha(P'_i[t])$, $1 \leq t \leq d$, differ by a multiple of π according to Lemma 1. This means $\sin(\alpha_t(P'_i[t])) = \theta \sin(\alpha_t(P_i[t]))$ and $\cos(\alpha_t(P'_i[t])) = \theta \cos(\alpha_t(P_i[t]))$ where θ is either + or - sign. The random noises ϵ from the symmetric distribution would cancel the effect of θ, that is, for any (v_1, v_2), $\Pr[(I_i[t]_1, I_i[t]_2) = (v_1, v_2)]$ equals $\Pr[(I'_i[t]_1, I'_i[t]_2) = (v_1, v_2)]$. Therefore, $\Pr[\textit{Enc}_1(P_i) = V] = \Pr[\textit{Enc}_1(P'_i) = V]$.

In the following, we first show that *SSE*$_1$ ensures strict class indistinguishability and then show that *SSE* composed by *SSE*$_1$ and *SSE*$_2$ achieves class indistinguishability.

Theorem 1. SSE$_1$ *constructed in Section 4.2 meets strict class indistinguishability, i.e.,* $\Pr[\textbf{\textit{Ind}}_{\text{SSE}_1,\mathcal{A}}(k_1) = 1] = \frac{1}{2}$.

Proof. Consider two histories H_0 and H_1 chosen by the adversary in Definition 3. H_0 and H_1 are from the same class and have the same trace. The challenger randomly chooses $b \in \{0, 1\}$ to encrypt $(\mathcal{D}_b, \mathcal{Q}_b)$ with *Enc*$_1$ and *Enc*$_{q,1}$ and sends the results to the adversary. From Lemma 3, H_0 and H_1 are equally likely to be the underlying history based on the observed ciphertexts. This remains true even if the adversary is allowed to compute the candidate set *Cand*$(Q_{b,j})$, $1 \leq j \leq m$, because *Cand*$(Q_{0,j})$ and *Cand*$(Q_{1,j})$ have the same size. Finally, any index structure $\mathbf{I}$ constructed using *Enc*$_1(P_{b,i})$, $1 \leq i \leq n$, discloses no more information than *Enc*$_1(P_{b,i})$ does. So the adversary gains no advantage in guessing the value of b from accessing $\mathbf{I}_b$ and $\mathbf{t}_b$ computing the queries.□

Theorem 2. *Let SSE*$_1$ *be constructed in Section 4.2 and let SSE*$_2$ *be any scheme meeting ciphertext indistinguishability (say [7]). Then SSE* $=$ (*SSE*$_1$, *SSE*$_2$) *meets class indistinguishability, that is,* $|\Pr[\textbf{\textit{Ind}}_{\text{SSE},\mathcal{A}}(k_1, k_2) = 1] - \frac{1}{2}| \leq \textit{negl}(k_2)$ *where* k_2 *is the security parameter of SSE*$_2$.

Proof. Consider the two histories $H_0 = (\mathcal{D}_0 = \{P_{0,1}, \cdots, P_{0,n}\}, \mathcal{Q}_0 = \{Q_{0,1}, \cdots, Q_{0,m}\})$ and $H_1 = (\mathcal{D}_1 = \{P_{1,1}, \cdots, P_{1,n}\}, \mathcal{Q}_1 = \{Q_{1,1}, \cdots, Q_{1,m}\})$, chosen by the adversary for *SSE*$_1$. Unlike *SSE*$_1$ alone, the adversary also has access to *Enc*$_2(P_{b,i})$ and *Enc*$_{q,2}(Q_{b,j})$, as well as *Cand*$(Q_{b,j})$, $1 \leq j \leq m$, computed by SSE_1. The ciphertext indistinguishability assumption of *SSE*$_2$ implies that the advantage in guessing the value of b from accessing *Enc*$_2(P_{b,i})$ and *Enc*$_{q,2}(Q_{b,j})$ is negligibly different from the probability $\frac{1}{2}$. This remains so even in the access to *Cand*$(Q_{b,j})$, $1 \leq j \leq m$, because *Cand*$(Q_{0,j})$ and *Cand*$(Q_{1,j})$ have the same size. Finally, this advantage is unaffected by running the game of *SSE*$_1$ because the adversary gains no advantage in the game of *SSE*$_1$ according to Theorem 1. □

So far, we considered a non-adaptive adversary in Definition 3 where the adversary chooses all queries in the query sequences $\mathcal{Q}_0$ and $\mathcal{Q}_1$ before receiving the encryption of any record or query. An adaptive adversary can choose adaptively the next query pair $(Q_{0,j}, Q_{1,j})$ in the query sequences *after* receiving the encrypted records and encrypted queries for the previous queries $\{Q_{b,1}, \cdots, Q_{b,j1}\}$. The strict class indistinguishability in Theorem 1 allows us to extend Theorems 1 and 2 to an adaptive adversary: the strict class indistinguishability implies that receiving the ciphertexts of previous queries does not give the adversary any advantage of guessing the value of b.

6 Evaluation

In this section, we evaluated CLASS presented in Section 4.

Data sets. We used the **US Census data set** [1] which was collected from 2006 to 2011 with $d = 3$ categorical attributes: Race (237), PlaceOfBirth (531) and City (1134), with the domain size indicated in the bracket. $\mathcal{D}_{1M}$, $\mathcal{D}_{10M}$, $\mathcal{D}_{50M}$ and $\mathcal{D}_{100M}$, denote four samples containing the first 1, 10, 50, and 100 million records, respectively.

Queries. We generated a query pool $QW = Q^1 \cup \cdots \cup Q^d$ using $\mathcal{D}_{1M}$. For each integer $q \in [1, d]$, Q^q contains 100 q-equality queries generated as follows. Let Q^q_* contain all q-equality queries that have a non-empty result in $\mathcal{D}_{1M}$. Let Sel_Q denote the *selectivity* of a query Q, defined as the percentage of records in the data that satisfy the query. We picked 100 queries Q from Q^q_*. The probability of picking a query Q is modeled by the beta distribution $Beta(\alpha, \beta)$ of the selectivity sel_Q [8]. In general, with a fixed β a smaller α leads to a higher probability for a query with a smaller selectivity. We set $\alpha = 0.5$ and $\beta = 3$, which assigns a higher probability to a query having a smaller selectivity, modeling the typical scenario that more queries retrieve more specific information.

Competing methods. For CLASS, we implemented the sub-linear method $Search_1$ for hyperplane queries by M-Tree [6] and the linear method $Search_2$ by Secure Index [9]. Since [9] deals with only single-keyword search, we convert equality conjunction queries to single-keyword search by treating each conjunction up to the maximum number of equalities in a query as a new keyword. We used the method in Section 4.3 to construct the class partitioning for each attribute for a given class size κ with the single equality queries Q^1_* as the input. *By default*, we set the class size as $\kappa = 6$, and the bounds for the noise interval as $L = 1000$ and $U = 1100$.

We consider two baselines. We provide brief outlines of the baselines as follows to keep the paper self-contained. Please refer to the references for more details. The first baseline is OXT, the state-of-the-art sub-linear search for conjunctive keywords queries [5]. OXT uses a disk-resident data structure *TSet* to locate the documents containing the least frequent keyword in the query, called s-term, and uses a RAM-resident data structure *XSet* to filter the result using the remaining keywords in the query, called x-terms. The second baseline, denoted by SI, is Secure Index [9] applied to the full database following the same strategy of converting equality conjunction queries to single-keyword search as described above for $Search_2$ in CLASS. We wrote all codes in C++ and leveraged OpenSSL library to implement cryptographic primitives. We simulated both the server and the client by a Linux machine with a single Intel Core i7 CPU with 2.3 GHz and 16 GB RAM.

6.1 Setup Cost

At system initialization, there is a one-time *setup cost*. Here, we focus on the storage overhead. For $\mathcal{D}_{1M}$, the storage overhead of SI, CLASS and OXT are 16MB, 139MB and 1.2GB, respectively. The storage overhead for other sample sizes scales up linearly. These structures were stored on the server side, thus, there is no client side storage overhead. Since these structures were generated by the client, they also represent the upload communication cost at setup. OXT uses most storage and SI uses least storage.

6.2 Query Cost

For each query, We focus on query computing time (averaged over the queries in QW). Note that we omit the comparison on communication cost because our method has the minimum communication cost by filtering false positives on the server side. Figure 2 reports query time in log scale vs four different data cardinality. For $\mathcal{D}_{100M}$, we could not get OXT's query time due to long database encryption time. In fact, OXT hides the entries on an inverted list by storing them in random locations on disk, which results in a large number of random I/O accesses during index construction and query process. As expected, the query time of SI grows linearly with data cardinality. However, SI took about 1000 seconds on $\mathcal{D}_{100M}$ which is too slow for large databases. It is clear that CLASS outperforms SI and OXT.

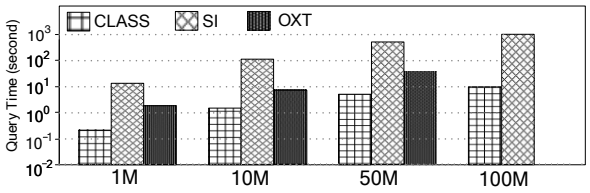

Fig. 2: Query time vs data cardinality

The efficiency of CLASS relies on the sub-linear Candidate Phase to reduce the search space of the linear Filtering Phase to a small candidate set. We measure this effectiveness by two metrics:

$$\textit{candidate\_size} = \frac{|\textit{Cand}|}{|\mathcal{D}|}, \quad \textit{search\_size} = \frac{|\textit{Test}|}{|\mathcal{D}|}$$

where *Cand* denotes the candidate set computed by Candidate Phase and *Test* denotes the set of records that are searched in Candidate Phase to compute *Cand*. *search_size* measures the reduction of search space in Candidate Phase whereas *candidate_size* measures the reduction of search space in Filtering Phase. In all data sets tested, *candidate_size* is no more than 0.1% and *search_size* is no more than 4%. For example, the average total query time of CLASS on $\mathcal{D}_{50M}$ is less than 6 seconds. In the following, we study the effect of other factors to the efficiency of CLASS based on $\mathcal{D}_{50M}$.

Effect of class size. The class size κ of a class partitioning plays a role in balancing the level of indistinguishability and the sub-linear search performance. We studies the effect of the class size $\kappa \in \{2, 6, 10, 14, 18\}$ (x-axis) on query time. As shown in Figure 3, a larger κ leads to larger *search_size* and *candidate_size* due to more data tested in Candidate Phase and more false positives in the candidate set *Cand*. Despite this trend, even for $\kappa = 18$, *Cand* is 0.2% of the full data set. This significantly reduces the time of Filtering Phase that is applied to the candidate set, as shown in Figure 3(B). In all cases, the total average query time of the two phases is no more than 8 seconds. This study clearly shows that the sub-linear Candidate Phase is highly effective in pruning the search space.

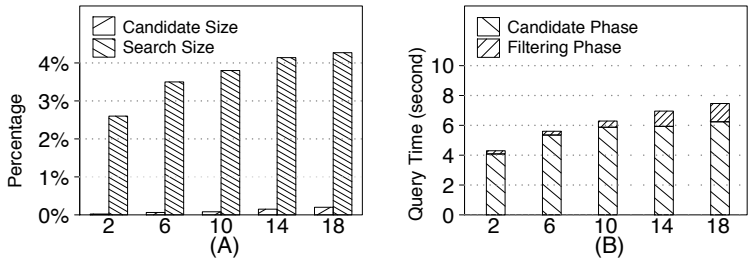

Fig. 3: Query time of CLASS vs value class size κ (50M Records)

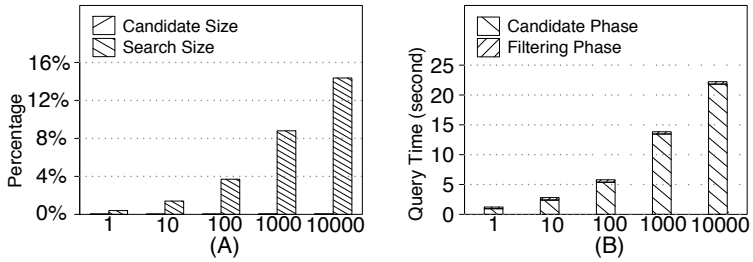

Fig. 4: Query time of CLASS vs nosize interval $(U - L)$ (50M Records, $L = 1000$)

Effect of random noises. Figure 4 examines the impact of the interval $[-U, -L] \cup [L, U]$ for drawing random noises ϵ, μ in Enc_1 and $Enc_{q,1}$. We fixed the lower limit $L = 1000$ and varied the size $(U - L)$ (x-axis). A larger $(U - L)$ leads to more random noises injected, thus less effective indexed search in Candidate Phase as shown by the larger *search_size*. However, even with the maximum $(U - L) = 10000$, *candidate_size* remains very small, which suggests that restricting Filtering Phase to the candidate set is highly effective. In general, Filtering Phase employs crypto primitives for producing the exact query result, therefore, it is more important to reduce the search space in this phase. Our two phase search exactly achieves this goal.

7 Conclusion

A key challenge of outsourcing data management is providing a provable security guarantee (e.g., ciphertext indistinguishability) while supporting a sub-linear search performance for dealing with large databases. The existing bucketization approach partially addresses this requirement at the cost of client performing search or increased communication cost of transmitting false positives. We proposed a novel SSE scheme, called CLASS, that provides a similar level of security to that of bucketization and pushes the work of search and false positive filtering tasks to the server. CLASS is a "framework" of sub-linear search through a two-phase search in which the search algorithms in both phases can be instantiated by existing methods.

Acknowledgments. This work was partially supported by a Discovery Grant from Canada's NSERC.

References

1. Ipums us census data set, https://www.ipums.org
2. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: ICIC (2005)
3. Boldyreva, A., Chenette, N., Lee, Y., O'neill, A.: Order-preserving symmetric encryption. In: EUROCRYPT (2009)
4. Bösch, C.T., Hartel, P.H., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Computing Surveys (2014)
5. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: CRYPTO (2013)
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB (1997)
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In: CCS (2006)
8. Falls, L.W.: The beta distribution: a statistical model for world cloud cover. Journal of Geophysical Research (1974)
9. Goh, E.J.: Secure indexes. IACR (2003)
10. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: ACNS (2004)
11. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: SIGMOD (2002)
12. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. VLDB (2012)
13. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: VLDB (2004)
14. Ji, X., Mitchell, J.E.: Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. Discrete Optimization (2007)
15. Kamara, S., Moataz, T.: Boolean searchable symmetric encryption with worst-case sublinear complexity. In: Eurocrypt (2017)
16. Li, M., Yu, S., Cao, N., Lou, W.: Authorized private keyword search over encrypted data in cloud computing. In: ICDCS (2011)
17. Lin, W., Wang, K., Zhang, Z., Chen, H.: Revisiting security risks of asymmetric scalar product preserving encryption and its variants. In: ICDCS (2017)
18. Oliveira, S.R., Zaiane, O.R.: Privacy preserving clustering by data transformation. In: SBBD (2003)
19. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: SOSP (2011)
20. R-Tree: https://en.wikipedia.org/wiki/R-tree
21. Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path oram: an extremely simple oblivious ram protocol. In: CCS (2013)
22. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (2002)
23. Wang, P., Ravishankar, C.V.: Secure and efficient range queries on outsourced databases using $\hat{R}$-tree. In: ICDE (2013)
24. Wong, W.K., Cheung, D.W.L., Kao, B., Mamoulis, N.: Secure knn computation on encrypted databases. In: SIGMOD (2009)
25. Yi, X., Kaosar, M.G., Paulet, R., Bertino, E.: Single-database private information retrieval from fully homomorphic encryption. TKDE (2013)
26. Zhang, Y., Katz, J., Papamanthou, C.: All your queries are belong to us: The power of file-injection attacks on searchable encryption. Usenix (2016)