EARRING: Efficient Authentication of Outsourced Record Matching

Boxiang Dong
Department of Computer Science
Montclair State University
Montclair, NJ 07043
dongb@montclair.edu

Hui (Wendy) Wang
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ 07030
Hui.Wang@stevens.edu

Abstract

Cloud computing enables the outsourcing of big data analytics, where a third-party server is responsible for data management and processing. In this paper, we consider the outsourcing model in which a third-party server provides record matching as a service. In particular, given a target record, the service provider returns all records from the outsourced dataset that match the target according to specific distance metrics. Identifying matching records in databases plays an important role in information integration and entity resolution. A major security concern of this outsourcing paradigm is whether the service provider returns the correct record matching results. To solve the problem, we design EARRING*, an Efficient Authentication of outsouRced Record matchING framework.* EARRING *requires the service provider to construct the verification object (VO) of the record matching results. From the VO, the client is able to catch any incorrect result with cheap computational cost. Experiment results on real-world datasets demonstrate the efficiency of* EARRING.

***Keywords*—**authentication, integrity, security, record matching

1 Introduction

With the rapid growth of the capability to generate and collect data, recent years have witnessed the explosion in data volumes. To distill fruitful knowledge from the massive data, sophisticated data analysis models are being developed progressively in many fields (e.g., healthcare and market decision). However, in practice, many companies and organizations, especially those small- and medium-sized ones, have limited benefits from big data analytics due to the inadequacy of computational resources. Fortunately, outsourcing provides a cost-effective alternative. A computationally powerful service provider can execute the complex and expensive data analytics tasks for the data owners.

In this paper, we consider *record matching* as the core computation task, which is an important operation in information integration, data cleaning, and information retrieval. In the outsourcing setting, the data owner outsources a dataset D to a third-party service provider (*server*). Hereafter, the legitimate users (clients) send the matching requests of specific target records to the server. The server is responsible to return all records in the outsourced dataset D that match the target records, where whether two records match depends on the similarity metrics.

Although outsourcing provides a cost-effective solution, there exist several security challenges. One major challenge is the *correctness* of the matching results that are returned by the server. There are numerous incentives for the server to cheat on the matching results. For instance, the server may try to save the computational cost by searching a portion of the outsourced dataset and returning the matching records in the portion. Or the server may use an approximate (and cheaper) distance metrics [9] instead of the exact ones for similarity measurement. Therefore, it is essential for the client to verify the correctness of the results returned by the server before exploiting them for benefits.

A naive method of correctness verification is that the client executes record matching locally and compares the local results with the results returned by the server. Obviously the naive method is not acceptable due to its prohibitively high cost. Considering the limited computational power of the client, our goal is to design a lightweight authentication method that enables the client to verify the correctness of the outsourced record matching results with cheap verification cost. The correctness verification is twofold: (1) *soundness* - all returned records indeed match the target records; and (2) *completeness* - all matching records are returned by the server.

There are quite a few existing work for authentication of various computational tasks (e.g. SQL queries [13], function queries [20], and nearest neighbors search [18]). Among all the existing work, [5] is the one that is most relevant to ours. By their approach, the server builds the *verification object* (VO) that includes all records, including both matching and mismatching ones, to prove both soundness and completeness of the matching results. However,

the verification method in [5] can be prohibitively expensive, especially when the number of mismatching records is large, since the client has to calculate the similarity between the target record and every single mismatching record.

In this paper, we design *EARRING*, an Efficient Authentication of outsouRced Record matchING. The key idea is that the server constructs a small set of *representative objects* of the mismatching records, and only includes the representative objects in VO instead of the mismatching records. The client can verify the result correctness by inspecting the representative objects only. To construct the representative objects, we first apply a similarity-preserving mapping function to transform the records into the Euclidean space, in the way that the similar records are embedded as Euclidean points that are close to each other. Then the Euclidean points of the mismatching records are partitioned into a small number of groups named *distant bounding hyper-rectangles* (DBHs). The $DBHs$ act as the representative objects of the mismatching records and thus are included in the VO. The client can verify the correctness by re-computing the Euclidean distance between the embedded Euclidean point of the target record and DBHs. Note that the computation of Euclidean distance is much cheaper than the measurement of record similarity.

In particular, we make the following contributions. First, we prove that it is NP-complete to construct the minimum number of DBHs for the mismatching records. We design an efficient heuristic algorithm to build a small number of DBHs to represent the mismatching records. Second, we design a lightweight approach to construct VO for result authentication. Compared with [5], our approach replaces a large amount of string edit distance calculations with a small number of cheap Euclidean distance computations. Therefore, the verification cost at the client side is significantly reduced. Third, we provide detailed security analysis to show that *EARRING* can catch any incorrect result returned by the server. Last but not least, we launch an extensive set of experiments on real-world datasets. Experiment results demonstrate that *EARRING* saves up to 91% verification time at the client side compared with *ARM* [5].

2 Preliminaries

2.1 Record Matching

In this paper, we consider the *categorical* datasets where the records consist of a certain number of strings as the attribute values. For the sake of simplicity, we assume that each record only contains a single string value. In the rest of the paper, we use strings and records interchangeably. The key of record matching is to evaluate the similarity of attribute values. There exist quite a few evaluation metrics for strings, e.g., Hamming distance, Jaccard distance based on q-grams, and Levenshtein (edit) distance [3]. Among these options, we concentrate on the edit distance due to its popularity. Informally, the edit distance between two strings s_1 and s_2, denoted as $DST(s_1, s_2)$, is the minimum number of character operations to transform s_1 to s_2. We say two records s_1 and s_2 match (or similar) if $DST(s, s_q) \leq \theta$, where θ is a user-specified threshold. Otherwise we say s_1 and s_2 mismatch (or dissimilar). Given a dataset D and a target string s_q, the *record matching* problem is to search for all strings $M \subseteq D$ that match s_q.

2.2 Mapping Records to Euclidean Space

Given two records s_1 and s_2, normally the complexity of computing edit distance is $O(|s_1||s_2|)$, where $|s_1|$ and $|s_2|$ are the lengths of s_1 and s_2. One way to reduce the complexity of similarity measurement is to map the records into a multi-dimensional Euclidean space, such that the similar strings are mapped to close Euclidean points. The main reason of the embedding is that the computation of Euclidean distance is much cheaper than that of string edit distance. In this paper, we use $dst()$ and $DST()$ to denote the Euclidean distance and edit distance. A few string embedding techniques (e.g., [9, 8]) exist in the literature. In this paper, we use the *SparseMap* method [8] for string embedding, since it preserves the contractiveness property. An embedding method is contractive [7] if for any pair of strings (s_i, s_j) and their embedded Euclidean points (p_i, p_j), $dst(p_i, p_j) \leq DST(s_i, s_j)$, where $dst()$ and $DST()$ are the distance function in the Euclidean space and string space respectively. We will show how to leverage this property to reduce the verification cost in Section 4. Note that the embedding methods may introduce false positives, i.e. the embedding points of dissimilar strings may be close in the Euclidean space.

2.3 MB-tree

In [5], an authentication tree structure named MB-tree is designed. MB-tree integrates Merkle hash tree [19] and B^{ed}-tree [24]. To construct MB-tree, first, all string values in D are sorted based on a function ϕ that maps each string to an integer value. Then the sorted strings are split into disjoint ranges; each range corresponds to a leaf node in the MB-tree. After range partitioning, the leaf nodes of the MB-tree are constructed. Each leaf node contains the entry of the format (s, p), where s is the string value, and p is the pointer to the disk position that stores s. Each internal node consists of up to f children nodes that correspond to adjacent f ranges, where f is the fanout of the tree. For each internal node N, its range is the union of the ranges of all of its children. The range is stored in the format of $[N_b, N_e]$; all the strings encoded by N fall into the range $[N_b, N_e]$. After the tree structure is constructed, for each node N, a hash value $h_N = h(h(N_b)||h(N_e)||h^{1\to f})$ is computed, where h is a collision-free hash function and $h^{1\to f} = h(h_{C_1}||\ldots||h_{C_f})$, with $C_1, \ldots, C_f$ being the children of N. If N is a leaf node, then $C_1, \ldots, C_f$ are the strings $s_1, \ldots, s_f$ that are covered by N. Finally, the root hash value h_{root} is computed as the digest of the whole

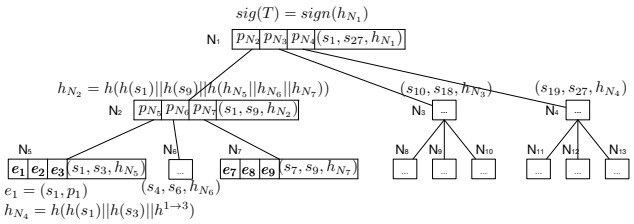


Figure 1: An example of MB-tree

tree structure. The signature of the hash value of the root is generated at the end for the verification purpose. Figure 1 shows an example of MB-tree.

We use $DST_{min}(s_q, N)$ to denote the minimal edit distance between a string s_q and any string s that is covered by a MB-tree node N. Based on this, next, we define *non-candidate nodes* (NC-nodes) and C-strings.

Definition 2.1. *Given a dataset D and its MB-tree T, for any string s_q, we say a MB-tree node N in T is a* non-candidate *node (NC-node) of s_q if $DST_{min}(s_q, N) > \theta$, where θ is a user-specified similarity threshold for record matching. Otherwise, we say N is a* candidate *node. For any string $s \in D$ that does not match s_q but is not covered by any NC-node in T, we call it a C-string.*

A nice property of NC-nodes is that it is guaranteed that none of the strings covered by either these NC-nodes or their descendents in the MB-tree match the given string s_q. Thus the matching search algorithm can stop at the NC-nodes. We will utilize this property to reduce VO size (more details in Section 4).

3 Problem Definition

3.1 System Framework

We consider the outsourcing model that involves three parties: (1) a data owner who possesses a dataset D that contains n records; (2) the user (client) who requests for record matching services on D; and (3) a third-party service provider (server) that provides storage and record matching as services. The client can be the data owner or an authorized party. To facilitate authenticated record matching, the data owner generates some auxiliary information of D, and sends D together with the auxiliary information to the server. The client sends the target record s_q and the similarity threshold θ to the server, asking for all records in D that matches s_q in terms of θ. The server returns the matching result M^S, as well as the verification object VO of M^S, to the client. The client checks the correctness of M^S via VO. Given the fact that the client may not possess D, we require that the availability of D is not necessary for the authentication.

3.2 Verification Goal

In this paper, we assume that the server may return incorrect record matching results to the client. Given a target string s_q and the similarity threshold θ, let M^S be the result returned by the server. We consider the following cheating behaviors that are executed by the server.

- **Data integrity violation.** The server may manipulate the data after receiving D and return any matching record that does not exist in D. In other words, there exists $s \in M^S$ s.t. $s \notin D$.
- **Result soundness violation.** The server returns at least one record that does not match s_q, i.e., there exists $s \in M^S$ s.t. $DST(s, s_q) > \theta$.
- **Result completeness violation.** The server may miss at least one matching record of s_q, i.e., there exists $s \in D$ such that $DST(s, s_q) \leq \theta$, but $s \notin M^S$.

Our authentication goal is to catch these three types of cheating behaviors on the matching results and assuring the client of the correctness of M^S with 100% certainty.

4 Authentication Approach

4.1 Approach Overview

We design an efficient authentication approach named *EARRING* to check the correctness of the outsourced record matching computations. *EARRING* consists of three phases.

- **Preparation phase.** The data owner constructs the MB-tree T from the dataset D and generates the embedding function f that transforms the strings to Euclidean points. The data owner sends D, T and f to the server. To reduce the network bandwidth consumption and the store overhead at the client side, the data owner sends the signature $sig(T)$ of the MB-tree T to the legitimate clients, as well as the embedding function f.
- **VO construction phase.** Upon receiving a target record s_q and the record matching request from the client, the server executes the computation to obtain the result M^S. In order to verify the correctness of M^S, the server also constructs the VO of M^S from the tree T and the embedded Euclidean points of D. After that, the server sends both M^S and VO to the client.
- **VO verification phase.** The client inspects the correctness of M^S by checking the VO against the auxiliary information (i.e., $sig(T)$ and f) obtained from the data owner.

It is worth noting that the preparation phase is only a one-time setup. Once the signature $sig(T)$ and the embedding function f are produced, they can be readily utilized for all the matching requests.

4.2 Authentication Preparation

Before outsourcing the dataset D to the server, the data owner constructs the MB-tree T on D [5]. He signs the root hash value of T and generates the signature $sig(T) = Encrypt_{sk}(h_{root})$ with his secret key sk. In addition, the data owner maps D to the Euclidean space E via a similarity-preserving embedding function f. In this paper, we use SparseMap [8] as the embedding function due to its contractive property (Section 2). The complexity of the embedding is $O(cdn^2)$, where c is a constant value between

0 and 1, d is the number of dimensions of the Euclidean space, and n is the number of strings of D. We admit that the complexity of string embedding is comparable to the complexity of record matching over D. However, the embedding procedure is performed only once. Its cost will be amortized over all record matching requests.

After embedding, the data owner sends D, T and the embedding function f to the server. The server constructs the embedded space of D by using the embedding function f. Meanwhile, the data owner sends the signature $sig(T)$ and the function f to the client for result authentication.

4.3 VO Construction

Given the record matching request (s_q, θ) from the client, the server discovers the set of matching strings in D whose edit distance to s_q is no larger than θ. We call those strings that do not match s_q the *false hits*. According to the property of MB-tree (Section 2.3), for any NC-node N of the MB-tree (i.e., $DST_{min}(s_q, N) > \theta$), all the strings covered by N must be false hits. We call a NC-node N in the MB-tree a (*maximal false hit*) node (MF-node) if the parent of N is a candidate node. Each MF-node covers at least f false hits, where f is the fan-out of the MB-tree. Thus, intuitively, to reduce the verification cost and the network communication overhead, the server only needs to include the MF-nodes in the VO. However, there may still exist a large fraction of false hits that are not covered by any MF-node. In [5], it is required that the client calculates the exact edit distance between s_q and any such false hit. However, it can lead to high verification overhead when there a large number of false hits that are not covered by any MF-node.

Our intuition to further reduce the verification cost of those false hits that are not covered by any MF-node is to: (1) substitute the expensive edit distance calculations with cheap Euclidean distance computation; and (2) use a small number of objects that represent those false hits that are not covered by any MF-node, and include these objects in the VO instead. Before we discuss the details of the VO construction, we define four types of the strings. We say two Euclidean points P and P_q are *distant* if $dst(P, P_q) > \theta$, where θ is the similarity threshold for record matching.

- M**-strings:** the strings that match the target s_q;
- NC**-strings:** the false hits that are covered by a MF-node;
- E**-strings:** the false hits that are not covered by any MF-node, but their Euclidean points are distant from the embedded point P_q of s_q, and
- FP**-strings:** the false hits that are neither NC-strings nor E-strings.

Obviously, any string in D must belong to exactly one of the four types.

In order to verify both soundness and completeness, all four types of strings must be represented in VO for the verification. Intuitively, NC-strings can be represented by MF-nodes. Our next task is to build the *representative objects* for E-strings. We design a novel concept as *distant bounding hyper-rectangles* (DBHs) as such representative objects. Before we define DBH, first, we define the minimum bounding hyper-rectangle (MBH).

Definition 4.1. *Given a set of points $\mathcal{P} = \{P_1, \ldots, P_t\}$ in a d-dimensional Euclidean space, a hyper-rectangle $R(\langle l_1, u_1\rangle, \ldots, \langle l_d, u_d\rangle)$ is the* minimum bounding hyper-rectangle *(MBH) of $\mathcal{P}$ if $l_i = min_{k=1}^{t}(P_k[i])$ and $u_i = max_{k=1}^{t}(P_k[i])$, for $1 \le i \le d$, where $P_k[i]$ is the i-dimensional value of P_k. For any point P and any hyper-rectangle R, the minimum Euclidean distance between P and R is $dst_{min}(P, R) = \sqrt{\sum_{1 \le i \le d} m[i]^2}$, where $m[i] = max\{l_i - p[i], 0, p[i] - u_i\}$.*

Intuitively, if the node P is inside R, the minimum distance between P and R is 0. Otherwise, we pick the length of the shortest path that starts from P to reach R. We have:

Lemma 4.1. *Given a hyper-rectangle R and a point $P \notin R$, for any point $P' \in R$, the Euclidean distance $dst(P', P) \ge dst_{min}(P, R)$.*

The proof of Lemma 4.1 is trivial. We omit the details due to the space limit.

Now we are ready to define (DBHs).

Definition 4.2. *Given a hyper-rectangle R and a point $P \notin R$, R is a* distant bounding hyper-rectangle *(DBH) of P if $dst_{min}(P, R) > \theta$, where θ is the given threshold.*

Next, we have the following theorem:

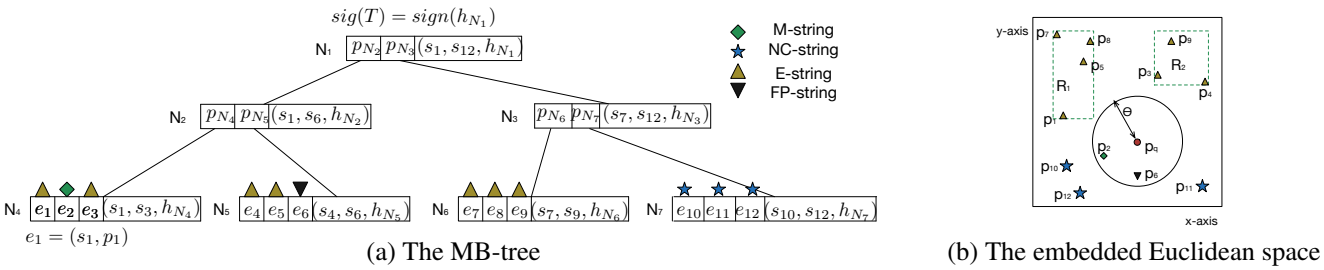
Theorem 4.1. *Given a target record s_q, let P_q be its embedded point in Euclidean space. Then for any record s, s must be a false hit of s_q if there exists a DBH R of P_q such that $P \in R$, where P is the embedded point of s.*

Based on Theorem 4.1, the server can build a number of DBHs from the embedded Euclidean points to represent all E-strings. In order to minimize the verification cost at the client side, our aim is to minimize the number of DBHs. Next, we define the $MDBH$ problem formally below.

MDBH problem: Given a set of E-strings $\{s_1, \ldots, s_t\}$, let $\mathcal{P} = \{P_1, \ldots, P_t\}$ be their embedded points in the Euclidean space. The $MDBH$ problem is to find a minimum number of DBHs $\mathcal{R} = \{R_1, \ldots, R_k\}$ such that: (1) $\forall R_i, R_j \in \mathcal{R}$, R_i and R_j do not overlap; and (2) $\forall P_i \in \mathcal{P}$, there exists a DBH $R \in \mathcal{R}$ such that $P_i \in R$.

Next, we discuss the intractability of the $MDBH$ problem and present our heuristic solution. We first present the simple case for the 2-D space (i.e. $d = 2$). Then we discuss the scenario when $d > 2$. For both settings, consider the same input that includes a query point P_q and a set of Euclidean points $\mathcal{P} = \{P_1, \ldots, P_t\}$ which are the embedded points of a target string s_q and its E-strings respectively.

When $d = 2$. We construct a graph $G = (V, E)$ such that for each point $P_i \in \mathcal{P}$, it corresponds to a vertex $v_i \in V$.



(a) The MB-tree (b) The embedded Euclidean space

Figure 2: An example of VO construction by EARRING

For any two vertices v_i and v_j that correspond to two points P_i and P_j, there is an edge $(v_i, v_j) \in E$ if $dst_{min}(P_q, R) > \theta$, where R is the MBH of P_i and P_j. We have:

Theorem 4.2. *Given the graph $G = (V, E)$ constructed as above, for any clique C in G, let R be the MBH constructed from the points corresponding to the vertice in C. Then R must be a DBH.*

Due to the space limit, the proof of Theorem 4.2 is omitted. Based on Theorem 4.2, next, we show the intractability of the *MDBH* problem for the 2D space.

Theorem 4.3. *Given a constant k, it is NP-complete to determine if there exists a solution to the* MDBH *problem whose size is at most k.*

Proof. Due to the space limit, we only show the proof sketch of Theorem 4.3. The proof is by a polynomial-time reduction from the well-known *clique partition problem*, which is to find the smallest number of cliques in a graph such that every vertex in the graph belongs to exactly one clique. ∎

Given the NP-completeness of the *MDBH* problem, we design a heuristic solution to solve the problem efficiently. Our heuristic algorithm is based on the concept of *maximal cliques*. Formally, a clique is *maximal* if it cannot be extended by one more adjacent vertex. The maximal cliques can be constructed in polynomial time [6]. It is shown that every maximal clique is part of some optimal clique-partition [6]. Based on this, finding a minimal number of cliques is equivalent to the problem of finding a number of maximal cliques. Based on this, we construct the maximal cliques of G iteratively by adapting the *RLF* algorithm in [12], until all the vertices belong to at least one clique.

When $d > 2$. Unfortunately, Theorem 4.2 can not be extended to the case of $d > 2$. There may exist MBHs of the pairs (v_i, v_j), (v_i, v_k), and (v_j, v_k) that are DBHs. However, the MBH of the triple (v_i, v_j, v_k) is not a DBH, as it includes a point w such that w is not inside $R(v_i, v_j)$, $R(v_i, v_k)$, and $R(v_j, v_k)$, but $dst(P_q, w) < \theta$.

To construct the DBHs for the case $d > 2$, we slightly modify the clique-based construction algorithm for the case $d = 2$. In particular, when we extend a clique C by adding an adjacent vertex v, we check if the MBH of the extended clique $C' = C \cup \{v\}$ is a DBH. If not, we delete the edge (u, v) from G for all $u \in C$. This step ensures that if we merge any vertex into a clique C, the MBH of the newly generated clique is still a DBH.

For both cases $d = 2$ and $d > 2$, the complexity of constructing DBHs from E-strings is $O(n_{ES}^3)$, where n_{ES} is the number of E-strings.

Now we are ready to describe the VO construction procedure. Given a dataset D and a target string s_q, the server constructs VO from the M-strings, FP-strings, MF-nodes, and DBHs. Formally,

Definition 4.3. *Given a target string s_q, let T be the MB-tree, and $\mathcal{R}$ be the set of DBHs constructed from E-strings. The VO of matching records of s_q consists of:*
(i) string s, for each M-string or FP-string;
(ii) a pair $([N_b, N_e], h^{1\to f})$ for each MF-node N, where $[N_b, N_e]$ is the range of the strings covered by N, and $h^{1\to f} = h(h_{C_1}||\ldots||h_{C_f})$, with $C_1, \ldots, C_f$ being the children of N;
(iii) the DBHs in $\mathcal{R}$; and
(iv) a pair (s, p_R) for each E-string, where p_R is the pointer to the DBH in $\mathcal{R}$ that covers the Euclidean point of s.
Furthermore, in VO, a pair of square bracket is added around the strings that share the same parent in T.

Intuitively, M-strings and FP-strings are included as original strings in VO, while NC-strings are represented by MF-nodes, and E-strings are represented by DBHs.

Example 4.1. *Consider the MB-tree in Figure 2 (a). The target record is s_q. s_2 is the only M-string. The NC-strings are s_{10}, s_{11}, and s_{12}. Consider the embedded Euclidean space shown in Figure 2 (b). Apparently s_6 is a FP-string as $dst(p_q, p_6) < \theta$. So the E-strings are $s_1, s_3, s_4, s_5, s_7, s_8$, and s_9. The DBHs of these E-strings are shown in the rectangles in Figure 2 (b). The VO of target record s_q is*

$VO = \{(((((s_1, p_{R_1}), s_2, (s_3, p_{R_2})), ((s_4, p_{R_2}), (s_5, p_{R_1}), s_6)),$
$(((s_7, p_{R_1}), (s_8, p_{R_1}), (s_9, p_{R_2})), ([s_{10}, s_{12}], h^{10\to 12}))), \{R_1,$
$R_2\}\},$ *where* $h^{10\to 12} = h(h(s_{10})||h(s_{11})||h(s_{12}))$.

4.4 VO Verification

After receiving M^S and VO from the server, the client uses VO to verify if M^S meets the data integrity, soundness, and completeness requirements. The verification of VO consists of four steps.

Step 1: Re-construction of MB-tree. First, the client sorts the strings and string ranges (in the format of $[N_b, N_e]$) in VO according to the string ordering scheme. String s is put ahead of the range $[N_b, N_e]$ if $s < N_b$. It returns a total order of strings and string ranges. If there exists any two ranges $[N_b, N_e]$ and $[N'_b, N'_e]$ that overlap, the client concludes that the VO is not correct. If there exists a string $s \in M^S$ and a range $[N_b, N_e] \in VO$ such that $s \in [N_b, N_e]$, the client concludes that M^S is not sound, as s indeed is a false hit (i.e., it is included in a MF-node). Second, the client maps each string $s \in M^S$ to an entry in a leaf node in T, and each pair $([N_b, N_e], h_N) \in VO$ to an internal node in T. The client re-constructs the parent-children relationships between these nodes by following the matching brackets () in VO.

Step 2: Re-computation of root hash. After the MB-tree T is re-constructed, the client computes the root hash value of T. For each string value s, the client calculates $h(s)$, where $h()$ is the same hash function used for the construction of the MB-tree. For each internal node that corresponds to a pair $([N_b, N_e], h^{1\to f})$ in VO, the client computes the hash h_N of N as $h_N = h(h(N_b)||h(N_e)||h^{1\to f})$. Finally, the client re-computes the hash value of the root node, namely h'_{root}. The client recovers the original MB-tree's root hash value $h_{root} = Decrypt_{pk}(sig(T))$ by decrypting $sig(T)$ received from the data owner using the data owner's public key pk. The client then compares h'_{root} with h_{root}. If $h'_{root} \neq h_{root}$, the client concludes that the server's result fails the verification.

Step 3: Re-computation of necessary edit distance. First, for each string $s \in M^S$, the client re-computes the edit distance $DST(s_q, s)$, and verifies whether $DST(s_q, s) \leq \theta$. If all strings $s \in M^S$ pass the verification, then the client concludes that M^S is *sound*. Second, for each FP-string $s \in VO$ (i.e., those strings appear in VO but not M^S), the client verifies whether $DST(s_q, s) > \theta$. If it is not (i.e., s matches s_q indeed), the client concludes that the server fails the completeness verification. Third, for each range $[N_b, N_e] \in VO$, the client verifies whether $DST_{min}(s_q, N) > \theta$, where N is the corresponding MF-node associated with the range $[N_b, N_e]$. If it is not (i.e., node N is indeed a candidate node), the client concludes that the server fails the completeness verification.

Step 4: Re-computation of necessary Euclidean distance. Step 3 only checks the if the NC-strings and FP-strings in VO are indeed false hits. In this step, the client checks the E-strings. First, for each pair $(s, p_R) \in VO$, the client checks if $P_s \in R$, where P_s is the embedded point of s, and R is the DBH that p_R points to. If all pairs pass the verification, the client ensures that the DBHs in VO cover the embedded points of all the E-strings. Second, for each DBH $R \in VO$, the client checks if $dst_{min}(P_q, R) > \theta$. If not, the client concludes that the results are not complete.

Note that we do not require to re-compute the edit distance between any E-string and the target string. Instead we only require the computation of the Euclidean distance between a set of DBHs and the embedded points of the target record. Since the Euclidean distance computation is much faster than that of the edit distance, our approach saves much verification cost compared with [5]. More complexity analysis can be found in Section 4.5.

Example 4.2. *Following the running example in Example 4.1, after calculating the root hash h'_{root} from VO and comparing it with the one decrypted from the signature sig received from the data owner, the client performs the following computations: (1) for $M^S = \{s_2\}$, compute $DST(s_q, s_2)$; (2) for FP-string s_6, compute $DST(s_q, s_6)$; (3) for NC-strings s_{10}, s_{11}, and s_{12}, compute $DST_{min}(s_q, N_7)$; and (4) for E-strings $s_1, s_3, s_4, s_5, s_7, s_8$ and s_9, compute $dst_{min}(P_q, R_1)$ and $dst_{min}(P_q, R_2)$. Compared with [5] which computes 10 edit distances, our approach only computes 3 edit distances, and 2 Euclidean distances. Note that the Euclidean distances computation is much cheaper than the edit distance computation.*

4.5 Complexity Analysis

In this section, we compare our approach *EARRING* with the *ARM* approach [5] in terms of the time and space of the authentication preparation, VO construction, and verification phases. The comparison results are summarized in Table 1. Regarding the VO construction overhead at the server side, the DBH construction introduces $O(n^3_{ES})$ complexity. This makes the higher VO construction complexity of *EARRING* than that of *ARM*. However, such overhead can be afforded by the server given its computational power.

Regarding the VO size, the VO size of *ARM* is calculated as the sum of two parts: (1) the total size of the M-strings, FP-strings and E-strings (in string format), and (2) the size of MF- nodes. Note that $\sigma_M = 2\sigma_S + |h|$, where $|h|$ is the size of a hash value. In our experiments, it turned out that $\sigma_M/\sigma_S \approx 10$. The VO size of the our approach is calculated as the sum of three parts: (1) the total size of the M-strings, FP-strings and E-strings (in string format), (2) the size of MF-nodes, and (3) the size of DBHs. Our experimental results show that σ_D is small, and the VO size of *EARRING* is comparable to that of *ARM*.

Regarding the complexity of verification time, note that usually $n_{DBH} << n_{ES}$ as a single DBH can cover the Euclidean points of a large number of E-strings, where n_{DBH} and n_{ES} are the number of DBHs and E-strings respectively. Also note that C_{Ed} (i.e., complexity of an edit distance computation) is much more expensive than C_{El} (i.e, the complexity of Euclidean distance calculation). Our experiments show that the time to compute one single edit distance can be 20 times of computing one Euclidean distance.

Phase	Measurement	*EARRING* (our approach)	*ARM* [5]
Authentication preparation	Time	$O(cdn^2)$	$O(n)$
	Space	$O(n)$	$O(n)$
VO construction	Time	$O(n + n_{ES}^3)$	$O(n)$
	VO Size	$(n_M + n_{ES} + n_{FP})\sigma_S + n_{MF}\sigma_M + n_{DBH}\sigma_D$	$(n_M + n_{ES} + n_{FP})\sigma_S + n_{MF}\sigma_M$
Verification	Time	$O((n_M + n_{MF} + n_{FP})C_{Ed} + n_{DBH}C_{El})$	$O((n_M + n_{MF} + n_{ES} + n_{FP})C_{Ed})$

Table 1: Complexity comparison between *ARM* and *EARRING*
(n: # of strings in D; c: a constant in [0, 1]; d: # of dimensions of Euclidean space; σ_S: the average length of the string; σ_M: Avg. size of a MB-tree node; σ_D: Avg. size of a DBH; n_M: # of M-strings; n_{FP}: # of FP-strings; n_{ES}: # of E-strings; n_{DBH}: # of DBHs; n_{MF}: # of MF-nodes; C_{Ed}: the complexity of an edit distance computation; C_{El}: the complexity of Euclidean distance calculation.)

Therefore, compared with *ARM*, *EARRING* significantly reduces the verification overhead at the client side. We admit that it increases the overhead of authentication preparation at the data owner side and the VO construction at the server side. We argue that, as the authentication preparation phase is a one-time operation, the cost of constructing the embedding function can be amortized by a large number of queries from the client. And as the server is equipped with strong computing resources, it is of the highest priority to reduce the verification complexity at the client side. Therefore, *EARRING* is remarkably advantageous over *ARM* [5].

5 Security Analysis

In this section, we discuss how EARRING can catch any data integrity, result soundness and completeness violation based on VO.

Data integrity. The server may include tampered values in M^S. The tampered values can be easily caught by the VO verification procedure, as the hash values of the tampered strings are not the same as the original strings. This leads to that the root hash value of MB-tree re-constructed by the client different from the root value of original dataset. The client can catch the tampered values by Step 2 of the verification procedure.

Result soundness. To violate soundness, the server returns $M^S = M \cup FS$, where M is the set of matching strings in D, and FS is a subset of false hits. We consider two possible ways that the server constructs VO: (Case 1.) the server constructs the VO of the correct result M, and returns $\{M^S, VO\}$ to the client; and (Case 2.) the server constructs the VO of M^S, and returns $\{M^S, VO\}$ to the client. Note that the strings in FS can be NC-strings, FP-strings, and E-strings. Next, we discuss how to catch these three types of strings for both cases.

For Case 1, for each NC-string $s \in FS$, s must fall into a MF-tree node in VO. Thus, there must exist an MF-tree node whose associated string range overlaps with s. The client can catch s by Step 1 of the verification procedure. For any FP-string $s \in FS$, s can be caught by re-computing the edit distance $DST(s, s_q)$ (i.e., Step 3 of verification). For any E-string $s \in FS$, let P_s be the embedded point of s. Since the VO is constructed from the correct M and F, there must exist a DBH in VO that includes P_s. Therefore, s can be caught by verifying whether there exist any DBH that includes the embedded points of s (Step 4 of verification).

For Case 2, the NC-strings and FP-strings in FS can be caught in the same way as in Case 1. The E-strings cannot be caught by Step 4 now, as: (1) the DBHs constructed from a subset of E-strings are still DBHs, and (2) no string in FS is included in a DBH in the VO. However, these E-strings are treated as FP-strings (i.e., not included in any DBH), and thus can be caught by Step 3.

Result completeness. To violate the completeness requirements, the server returns returns $M^S = M - SS$, where $SS \subseteq M$. Let V and V' be the VO constructed from M and M^S respectively. We again consider the two cases as for the discussion of result soundness violation.

For Case 1, where the VO is constructed from the correct result M, any string $s \in SS$ is a FP-string, as s is not included in M. Then by calculating $DST(s, s_q)$, (i.e., Step 3 of the verification procedure), the client discovers that s is indeed similar to s_q and thus catch the incomplete results.

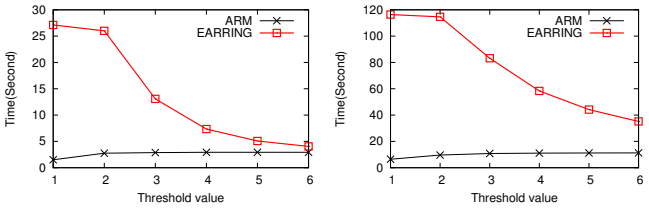
For Case 2, where the VO is constructed from M^S, any string $s \in SS$ is either a FP-string or a E-string. If s is treated as a FP-string, it can be caught by recomputing of edit distance (Step 3 of verification). If s is a E-string, then its Euclidean point P_s must be included in a DBH R. Including P_s into any MBH will make it a non-DBH. The client can easily catch it by re-computing the euclidean distance (Step 4 of verification).

6 Experiments

6.1 Experiment Setup

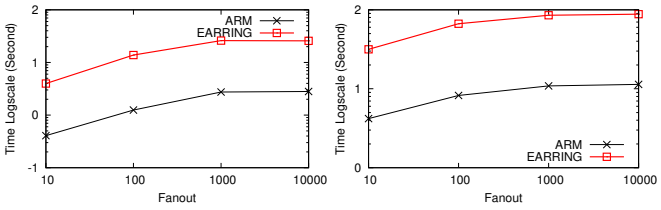
Datasets and queries. We use two real-world datasets : (1) the *Actors* dataset from IMDB[1] that contains 260K last names; and (2) the *Authors* dataset from DBLP[2] including 1,000,000 researcher names. As both datasets are very large, it is too time-consuming to execute approximate matching for all records in these two datasets. Thus, we picked 10 target strings for approximate matching. For the following results, we report the average performance of matching of the 10 target strings.

[1] http://www.imdb.com/interfaces
[2] http://dblp.uni-trier.de/xml/



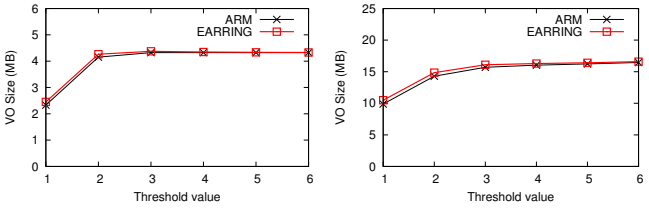
(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 3: VO construction time w.r.t. distance threshold θ



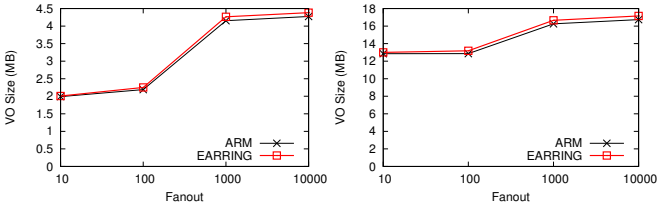
(a) The *Actors* dataset ($\theta = 2$) (b) The *Authors* dataset ($\theta = 3$)

Figure 4: VO construction time w.r.t. MB-tree fanout f



(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 5: VO size w.r.t. distance threshold θ



(a) The *Actors* dataset ($\theta = 2$) (b) The *Authors* dataset ($\theta = 3$)

Figure 6: VO size w.r.t. MB-tree fanout f

Experimental environment. We implement our *EARRING* approach in C++. The hash function we use is the *SHA256* function from the OpenSSL library. We execute the experiments on a machine with 2.4 GHz CPU and 48 GB RAM, running Linux 3.2.

Baseline. As the baseline method, we implement the *ARM* approach [5] that authenticates the record matching result purely based on the MB-tree.

Parameter setup. In the experiments, we use various string edit distance threshold θ and fanout f of MB-tree nodes. We are interested in observing the performance of our approach under different parameter values.

6.2 VO Construction Time

First, we measure the impact of distance threshold θ on the VO construction time. In Figure 3, we show the VO construction time with regard to different θ values on both datasets. First, we observe a dramatic decrease in VO construction time of *EARRING* when θ increases. The reason lies in the sharp reduction in n_{ES}, i.e., the number of E-strings. For example, on the *Actors* dataset, when θ changes from 2 to 3, n_{ES} decreases from $113,673$ to $49,886$. Since the complexity of VO construction is cubic to n_{ES}, the total VO construction time decreases intensively when n_{ES} decreases. Second, we note that *EARRING* consumes more time on constructing VOs than *ARM*, especially when θ is a small value. This is because besides traversing the MB-tree like *ARM*, *EARRING* constructs DBHs for the E-strings. The number of E-strings n_{DS} is large when θ is small. As a result, the DBH construction complexity is higher when the threshold is small.

We also measured the impact of the MB-tree structure on the VO construction time. From Figure 4, we observe that, in all the experiments, the VO construction time increases with the fanout value f. Even though the total number of nodes in the MB-tree decreases with larger fanout values, the decrease in the number of MFs n_{MF} is more intense. For example, on the *Authors* dataset, when $f = 100$, $n_{MF} = 5,420$; while when $f = 1,000$, $n_{MF} = 57$. The reason behind the significant reduction in n_{MF} is that when f is large, $DST_{min}(s_q, N)$ becomes a loose lower-bound for any MB-tree node N. This leads to a significant portion of the MB-tree nodes to become candidates. Besides, *EARRING* demands more VO construction time than *ARM* regardless of the fanout of the MB-tree. The underlying reason resides in the DBH construction cost of *EARRING*.

6.3 VO Size

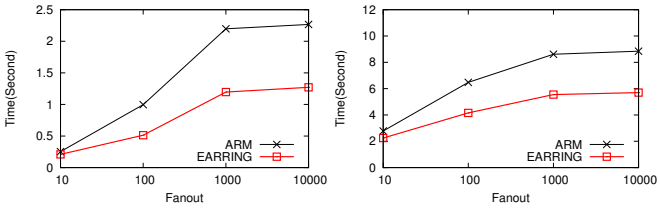
We measure the impact of the distance threshold θ on the VO size. The results are shown in Figure 5. We notice that the VO size of *EARRING* and *ARM* is very close. The reason is two manifold. On the one hand, the size of each DBH is very small. On the other hand, the number of DBHs n_{DBH} is also very small. In the experiments, $n_{DS}/n_{DBH} \approx 26$. In other words, each DBH covers the Euclidean points of 26 DBH-strings. Due to these two reasons, even though *EARRING* introduces DBHs additionally to the VO, the increment in VO size is small. Another observation is that the VO size increases with the growth of θ value, especially when θ increases from 1 to 2. Apparently, larger θ values lead to more M-strings (i.e., larger n_M), fewer MF-nodes (i.e., smaller n_{MF}), and fewer FP-strings and E-strings. When we increase θ, the intensive growth of n_M leads to the increase of VO size. For example, on the *Actors* dataset, when $\theta = 1$, $n_M = 27,655$. While when $\theta = 3$, $n_M = 49,373$.

Furthermore, we measure the impact of the MB-tree fanout f on the VO size. Following the same reason as before, we notice in Figure 6 that *EARRING* produces VO of similar size as *ARM*. For both approaches, the VO size increases with the growth of f.

6.4 VO Verification Time

In this section, we measure the VO verification time of *EARRING* at the client side. We compare the VO verification time with the performance of *ARM*, aiming to show that *EARRING* reduces the client's verification effort.

In Figure 7, we report the VO verification time with regard to various distance threshold values. First, the VO verification time of *EARRING* increases when θ increases. Recall that the complexity of VO verification is $O((n_M + n_{MF} + n_{FP})C_{Ed} + n_{DBH}C_{El})$. When θ increases, n_{MF} decreases, while n_M and n_{FP} increase. As n_M and n_{FP} increase faster than the decrease of n_{MF}, it leads to more edit distance calculations. Therefore, the total verification time increases. Second, the verification time of *EARRING* is always smaller than that of *ARM*, and when θ is smaller, *EARRING* wins more over *ARM*. We conduct deeper study to explore the reason. We find that rather than θ, the key factor to verification time is the fraction of E-strings (denoted as p_{ES}), where $p_{ES} = \frac{n_{ES}}{n}$. In Figure 8, we compare the VO verification time of *EARRING* and *ARM* with regard to various p_{ES} values. In particular, let t_1 and t_2 be the verification time of of *EARRING* and *ARM* respectively. We report the ratio $r = \frac{t_1}{t_2}$ under different p_{ES} values. Intuitively, smaller r values show the effectiveness of *EARRING* in saving the verification time at the client side. The main observation is that r is smaller for larger p_{ES} value. This is because with *EARRING*, the client replaces the edit distance calculation for E-strings by efficiently computing the Euclidean distance of a small amount of DBHs. The larger p_{ES} is, the larger fraction of edit distance calculations that the client can avoid. In particular, *EARRING* can save up to 91% verification cost at the client side than *ARM* ($r = 9\%$). This demonstrates the strength of *EARRING* in reducing the VO verification complexity.



(a) The *Actors* dataset ($\theta = 2$) (b) The *Authors* dataset ($\theta = 3$)

Figure 9: VO verification time w.r.t. MB-tree fanout f

We also measure the VO verification time with various MB-tree fanout f. On both datasets, the VO verification of *EARRING* is always more efficient than *ARM*. Moreover, the advantage of *EARRING* is more evident when f grows. This is straightforward as larger f results in the growth of the number of E-strings.

In sum, the experiment results demonstrate that, compared with *ARM* [5], *EARRING* dramatically saves the verification cost at the client side, especially for small θ value. Meanwhile, the VO size is comparable with *ARM*.

7 Related Work

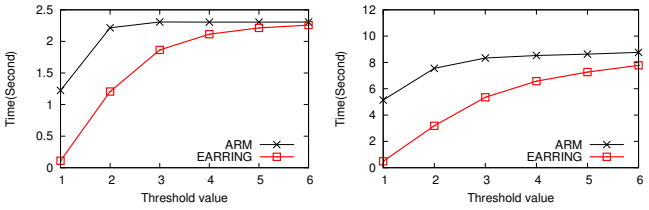
There are a lot of previous research that focus on the security issues of the outsourced computing. In this section, we discuss the related work in four directions.

When outsourcing the record matching task that involves private information to a third party, it is necessary to prevent the sensitive data from leakage. Karapiperis et. al [10] design a *secure multi-party computation* protocol based on homomorphic operations to calculate the distance between a pair of records over the ciphertext values. To make the solution practical for real-world applications, a blocking technique is applied to find candidate matching pairs in advance. Dong et al. [4] propose two encoding techniques to transform the sensitive data values in a similarity-preserving way. The record matching on the encoded data yields high accuracy. Zhang et al. [23] consider the potential collusions between parties that execute the record matching. By exploiting cryptographic operations such as pseudorandom permutations, a collusion-resistant protocol is proposed to protect the data privacy.

The correctness of outsourced SQL queries have been the research concentration for decades. Narasimha et al. [16] integrate digital signature aggregation and chaining to ensure the soundness and completeness of range selection queries and projections. Mykletun et al. [15] focus on the selection-projection queries. An authentication approach based on signature aggregation is proposed. Cheng et al. [2] propose a new authenticated data structure named verification R-tree (VR-tree) to check the correctness of queries on multidimensional databases. Bajaj et al. [1] propose *CorrectDB* by incorporating trusted hardware and cryptographic techniques. It supports the authentication of arbitrary SQL queries over the outsourced database.

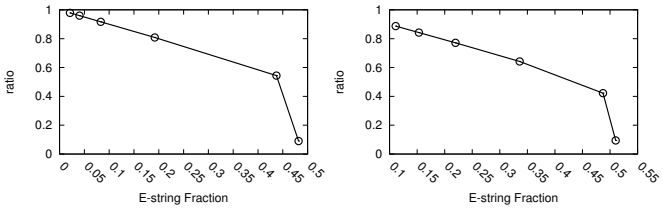
Regarding the authentication of nearest neighbor search, Yang et al. [21] designs a MR-tree index that integrates an R-tree with the Merkle hash tree for authentication of multi-dimensional range queries. Yiu et al. [22] focus on the moving KNN queries that continuously reports the k nearest neighbors of a moving query point. They designed the Voronoi MR-tree as the authenticated data structure for VO construction and authentication. [5] is the closest to our work. The authors propose a new authenticated data structure named MB-tree. To prove the correctness of the result, VO is constructed by traversing the tree. We improve the VO construction procedure, so that the verification cost at the client side is significantly reduced.

A large body of works authenticate various types of outsourced computations. Li et al. [14] integrate Merkle tree with a multi-dimensional indexing method to verify the recommendations made by the cloud broker. Papadopoulos et al. [17] design a new authenticated data structure based on bilinear mapping to verify outsourced pattern matching. Lee et al. [11] propose a reversible image authentication ap-



(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 7: VO verification time w.r.t. distance threshold θ



(a) The *Actors* dataset ($f = 1,000$) (b) The *Authors* dataset ($f = 1,000$)

Figure 8: VO verification time comparison

proach based on watermarking. The original image can be recovered after the hidden watermark is removed. However, these techniques cannot be directly applied to authenticate record matching due to the intrinsic difference in the computations.

8 Conclusion

In this paper, we design a lightweight authentication framework named *EARRING* to check the result correctness of the outsourced record matching. We require the server to construct verification object (VO) from the MB-tree and the embedding space to prove the result correctness. In the future, we plan to study how to authenticate the record matching on dynamic dataset. We also plan to design the authentication methods that support other types of similarity metrics such as semantic similarity and Hamming distance.

References

[1] S. Bajaj and R. Sion. Correctdb: Sql engine with practical query authentication. *VLDB Endowment*, 2013.

[2] W. Cheng and K.-L. Tan. Query assurance verification for outsourced multi-dimensional databases. *Journal of Computer Security*, 2009.

[3] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *KDD Workshop on Data Cleaning and Object Consolidation*, 2003.

[4] B. Dong, R. Liu, and W. H. Wang. Prada: Privacy-preserving data-deduplication-as-a-service. In *International Conference on Information and Knowledge Management*, 2014.

[5] B. Dong and W. Wang. Arm: Authenticated approximate record matching for outsourced databases. In *International Conference on Information Reuse and Integration*, 2016.

[6] S. Eidenbenz and C. Stamm. Maximum clique and minimum clique partition in visibility graphs. In *Theoretical Computer Science*. 2000.

[7] G. Hjaltason and H. Samet. Contractive embedding methods for similarity searching in metric spaces. Technical report, Technical Report TR-4102, Computer Science Department.

[8] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *Transactions on Pattern Analysis and Machine Intelligence*, 2003.

[9] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications*, 2003.

[10] D. Karapiperis and V. S. Verykios. An lsh-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 2015.

[11] S.-K. Lee, Y.-H. Suh, and Y.-S. Ho. Reversiblee image authentication based on watermarking. In *IEEE International Conference on Multimedia and Expo*, 2006.

[12] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 1979.

[13] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM Transactions on Information and System Security*, 2010.

[14] J. Li, A. Squicciarini, D. Lin, S. Sundareswaran, and C. Jia. Mmbcloud-tree: Authenticated index for verifiable cloud service selection. *IEEE Transactions on Dependable and Secure Computing*, 2015.

[15] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *ACM Transactions on Storage*, 2006.

[16] M. Narasimha and G. Tsudik. Dsac: integrity for outsourced databases with signature aggregation and chaining. In *International Conference on Information and Knowledge Management*, 2005.

[17] D. Papadopoulos, C. Papamanthou, R. Tamassia, and N. Triandopoulos. Practical authenticated pattern matching with optimal proof size. *VLDB Endowment*, 2015.

[18] S. Papadopoulos, L. Wang, Y. Yang, D. Papadias, and P. Karras. Authenticated multistep nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[19] R.C.Merkle. Protocols for public key cryptosystems. In *Symposium on Security and Privacy*, 1980.

[20] G. Yang, Y. Cai, and Z. Hu. Authentication of function queries. In *International Conference on Data Engineering*, 2016.

[21] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial outsourcing for location-based services. In *International Conference on Data Engineering*, 2008.

[22] M. L. Yiu, E. Lo, and D. Yung. Authentication of moving knn queries. In *International Conference on Data Engineering*, 2011.

[23] Y. Zhang, W.-K. Wong, S.-M. Yiu, N. Mamoulis, and D. W. Cheung. Lightweight privacy-preserving peer-to-peer data integration. In *VLDB Endowment*, 2013.

[24] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *International Conference on Management of Data*, 2010.