ARM: Authenticated Approximate Record Matching for Outsourced Databases

Boxiang Dong Wendy Wang
bdong@stevens.edu Hui.Wang@stevens.edu
Stevens Institute of Technology
Hoboken, NJ, 07030

Abstract

In this paper, we consider the outsourcing model in which a third-party server provides data integration as a service. Identifying approximately duplicate records in databases is an essential step for the information integration processes. Most existing approaches rely on estimating the similarity of potential duplicates. The service provider returns all records from the outsourced dataset that are similar according to specific distance metrics. A major security concern of this outsourcing paradigm is whether the service provider returns sound and complete near-duplicates.

In this paper, we design ARM, an authentication system for the outsourced record matching. The key idea of ARM is that besides the similar record pairs, the server returns the verification object (VO) of these similar pairs to prove their correctness. First, we design an authenticated data structure named MB-tree for VO construction. Second, we design a lightweight authentication method that can catch the service provider’s various cheating behaviors by utilizing VOs. We perform an extensive set of experiment on real-world datasets to demonstrate that ARM can verify the record matching results with cheap cost.

Keywords—authentication, outsourcing, approximate string matching, verification object, MB-tree

1 Introduction

Big data analytics offers the promise of providing valuable insights of knowledge. However, many companies, especially the small- and medium-sized ones, lack the computational resources, in-house knowledge and experience of big data analytics. A practical solution to this dilemma is that the data owner outsources the data to a computational powerful third-party service provider (e.g., the cloud) for cost-effective data analytics solutions.

In this paper, we consider approximate record matching, an important data analytics operation for information integration, as the outsourced computations. Generally speaking, the data owner (client) outsources a record database $D$ to a third-party service provider (server). The server identifies the similar record pairs in $D$ that may refer to the same entity, where the similarity is measured by a specific string distance function and user-defined threshold values.

For all the benefits of outsourcing and cloud computing, though, the outsourcing paradigm deprives the data owner of direct control over her data. This poses numerous security challenges. One of the challenges is that the server may cheat on the record matching results. For example, the server is incentivized to improve its revenue by computing with less resources (e.g., only search a portion of $D$) while charging for more. Therefore, it is important to authenticate whether the service provider has performed the matching faithfully, and returned the correct results to the client. A naive method is to execute the record matching locally, and compare the results with the matching results returned by the server. Apparently this method is prohibitively costly. We aim to design efficient methods that enable the client to authenticate that the server returned sound and complete similar record pairs. By soundness we mean that the returned record pairs are indeed similar. By completeness we mean that all similar records are returned.

Most existing work (e.g. [2, 7, 21]) solve the data privacy issue in the outsourced record matching computations, aiming to prevent the server from possessing the original data while still enabling the server to perform accurate record matching. Previous work (e.g. [13, 19, 26]) on authentication mainly focus on the SQL aggregation queries and similarity search for Euclidean points. To our best knowledge, ours is the first to consider the authentication of approximate record matching on outsourced data.

In this paper, we design ARM, an Authentication mechanism of outsourced Record Matching. The key idea of ARM is that besides returning the similar record pairs, the server returns a verification object (VO) that can prove the soundness and completeness of the returned record matching results. In particular, we make the following contributions. First, we design an authentication tree structure named MB-tree. MB-tree is constructed by integrat-
ing Merkle hash tree [22], a popularly-used authenticated data structure, with $B^{ed}$-tree [28], a compact index for efficient string similarity search based on edit distance. We pick $B^{ed}$-tree because compared with other data structures for string similarity measurement [25, 12], it is not limited to main memory usage and incurs minimal I/O time when there is limited memory in the system. Besides, $B^{ed}$-tree supports incremental updates in the dataset. Second, based on MB-tree, we design a lightweight authentication method named $VS^2$ for the approximate record matching. $VS^2$ constructs the verification object VO from the MB-tree. The VO consists of certain MB-tree entries that can be later utilized by the client to verify the completeness and soundness of results. Third, we formally prove that the VO enables to catch the server’s cheating behaviors such as tampered records, soundness violation, and completeness violation. Fourth, we complement the theoretical investigation with a rich set of experiment study on two real-world datasets. The experiment results show that ARM incurs cheap cost at the client side. It requires no more than 11 seconds to construct the VO. Furthermore, the data owner can save up to 87% computational efforts if she outsources record matching to the server, while only performing verification locally. This shows that ARM is suitable for the outsourcing setting.

The rest of the paper is organized as follows. Section 2 discusses the preliminaries. Section 3 defines the problem. Section 4 and 5 present our verification approach. Section 6 analyzes the security guarantee of our approach. The experiment results are shown in Section 7. Sections 8 discusses the related work. Section 9 concludes the paper.

2 Preliminaries

2.1 Record Similarity Measurement

Record matching is a fundamental problem in many research areas, e.g., information integration, database joins, and more. In general, the evaluation that whether two records match is mainly based on the string similarity of the attribute values. There are a number of string similarity functions, e.g., Hamming distance, n-grams, and edit distance (see [11] for a good tutorial). In this paper, we mainly consider edit distance, one of the most popular string similarity measurement that has been used in a wide spectrum of applications. Informally, the edit distance of two strings $s_1$ and $s_2$, denoted as $DST(s_1, s_2)$, measures the minimum number of insertion, deletion and substitution operations to transform $s_1$ to $s_2$. We say two strings $s_1$ and $s_2$ are similar, denoted as $s_1 \approx s_2$, if $DST(s_1, s_2) \leq \theta$, where $\theta$ is a user-specified similarity threshold. Otherwise, we say $s_1$ and $s_2$ are dissimilar (denoted as $s_1 \neq s_2$). Without loss of generality, we assume that the dataset $D$ only contains a single attribute. Thus, in the following sections, we use record and string interchangeably. Our methods can be easily adapted to the datasets that have multiple attributes.

2.2 Authenticated Data Structure

To enable the client to authenticate the correctness of mining results, the server returns the results along with some supplementary information that permits result verification. Normally the supplementary information takes the format of verification object (VO). In the literature, VO generation is usually performed by an authenticated data structure (e.g., [13, 26, 19]). One of the popular authenticated data structures is Merkle tree [22]. In particular, a Merkle tree is a tree $T$ in which each leaf node $N$ stores the digest of a record $r$: $h_N = h(r)$, where $h()$ is a one-way, collision-resistant hash function (e.g. SHA-1 ). For each non-leaf node $N$ of $T$, it is assigned the value $h_N = h(h_{C_1}||...||h_{C_k})$, where $C_1, \ldots, C_k$ are the children of $N$. The root signature $\text{sig}$ is generated by signing the digest $h_{\text{root}}$ of the root node using the private key of a trusted party (e.g., the data owner). The VO enables the client to re-construct the root signature $\text{sig}$.

2.3 $B^{ed}$-Tree for String Similarity Search

A number of compact data structures (e.g., [1, 4, 12]) are designed to handle edit distance based similarity measurement. In this paper, we consider $B^{ed}$-tree [28] due to its support for external memory and dynamic data updates. $B^{ed}$-tree is a $B^+$-tree based index structure that can handle arbitrary edit distance thresholds. The tree is built upon a string ordering scheme which is a mapping function $\varphi$ to map each string to an integer value. To simplify notation, we say that $s \in [s_i, s_j]$ if $\varphi(s_i) \leq \varphi(s) \leq \varphi(s_j)$. Based on the string ordering, each $B^{ed}$-tree node $N$ is associated with a string range $[N_b, N_e]$. Each leaf node contains $f$ strings $\{s_1, \ldots, s_f\}$, where $s_i \in [N_b, N_e]$, for each $i \in [1, f]$. Each intermediate node contains multiple children nodes, where for each child of $N$, its range $[N_i, N_j] \subseteq [N_b, N_e]$, as $[N_b, N_e]$ is the range of $N$. We say these strings that stored in the sub-tree rooted at $N$ as the strings that are covered by $N$.

We use $DST_{\text{min}}(s_q, N)$ to denote the minimal edit distance between a string $s_q$ and any string $s$ that is covered by a $B^{ed}$-tree node $N$. A nice property of the $B^{ed}$-tree is that, for any string $s_q$ and node $N$, the string ordering $\varphi$ enables to compute $DST_{\text{min}}(s_q, N)$ efficiently by computing $DST(s_q, N_b)$ and $DST(s_q, N_e)$ only, where $N_b, N_e$ refer to the string range values of $N$. Then:

**Definition 2.1** Given a string $s_q$ and a similarity threshold $\theta$, a $B^{ed}$-tree node $N$ is a candidate if $DST_{\text{min}}(s_q, N) \leq \theta$. Otherwise, $N$ is a non-candidate.

$B^{ed}$-tree has an important monotone property.

**Property 2.1 Monotone Property of $B^{ed}$-tree :** Given a node $N_i$ in the $B^{ed}$-tree and any child node $N_j$ of $N_i$,
for any string $s_q$, it must be true that $DST_{min}(s_q, N_i) \leq DST_{min}(s_q, N_j)$. Therefore, for any non-candidate node, all of its children must be non-candidates. This monotone property enables early termination of search on the branches that contain non-candidate nodes.

For any given string $s_q$, the string similarity search algorithm starts from the root of the $B^{ed}_d$-tree, and iteratively visits the candidate nodes, until all candidate nodes are visited. The algorithm does not visit those non-candidate nodes as well as their descendants. An important note is that for each candidate node, some of its covered strings may still be dissimilar to $s_q$. Therefore, given a string $s_q$ and a candidate $B^{ed}_d$-tree node $N$, it is necessary to compute $DST(s_q, s)$, for each $s$ that is covered by $N$.

3 Problem Formulation

In this section, we define our authentication problem.

3.1 System Model

We consider the outsourcing model that involves two parties - a data owner (client) who possesses a dataset $D$ that contains $n$ records, and a third-party service provider (server) that executes the record matching on $D$. The data owner outsources $D$ and the similarity threshold $\theta$ to the server. The server provides storage and record matching services. To facilitate authenticated record matching, the data owner generates some auxiliary information of $D$, and sends $D$ together with the auxiliary information. Upon finishing record matching on $D$, the server returns the matching result $M^S$ to the client.

3.2 Threat Model

We assume that the third-party server is not fully trusted as it could be compromised by the attacker (either inside or outside). The server may alter the received dataset $D$ and return any matching result that does not exist in $D$. It also may tamper with the matching results. For instance, the server may return incomplete results that omit some legitimate similar pairs in the matching results [16]. Note that we do not consider privacy protection for the outsourced database. This issue can be addressed by privacy-preserving record linkage [17] and is beyond the scope of this paper.

3.3 Verification Goal of Result Integrity

To catch the cheating on the matching results, we formally define the integrity of the record matching result from two perspectives: soundness and completeness. Let $M$ be the set of matching pairs in the outsourced database $D$, and $M^S$ be the result returned by the server. We define the precision $pre$ of $M^S$ as $pre = \frac{|M \cap M^S|}{|M|}$ (i.e., the percentage of returned pairs that are similar to each other), and the recall $rec$ of $M^S$ as $rec = \frac{|M \cap M^S|}{|M|}$ (i.e., the percentage of matching pairs that are returned). We say the result $M^S$ is incorrect if $pre < 1$ and incomplete if $rec < 1$. The soundness verification is straightforward. For each record pair $(s_i, s_j) \in M^S$, the data owner checks whether $s_i \approx s_j$. On the other hand, the completeness verification requires to verify for each record pair $(s_i, s_j) \notin M^S$, it is true that $s_i \not\approx s_j$. A naive method for the data owner to verify the result integrity (for both soundness and completeness) is to re-compute the similarity of all record pairs, which is prohibitively costly. In this paper, we design an efficient verification approach for the client to verify both the soundness and completeness of matching results.

3.4 Overview of Our Approach

Given a dataset $D$, the server finds all similar pairs in the format $(s_1, s_j)$. The pairs that have the same first record $s_i$ (called target) are assigned to the same group. For instance, given a set of similar pairs $M = \{(s_1, s_2), (s_1, s_3), (s_2, s_3), (s_2, s_5)\}$, the similar pairs are grouped into $G_1 = \{(s_1, s_2), (s_1, s_3)\}$ and $G_2 = \{(s_2, s_3), (s_2, s_5)\}$. Intuitively, $G_1$ ($G_2$, resp.) contains the records that are similar to its target record $s_i$ ($s_j$, resp.). Then for each group, the server constructs a verification object $VO$. The $VO$ of $G_1$ ($G_2$, resp.) can be used to verify the soundness and completeness of records that are similar to its target record $s_i$ ($s_j$, resp.). We require that by utilizing $VO$, the client only needs to perform a small number of string similarity calculations. This is enabled by using the monotone property of the $B^{ed}_d$-tree (Property 2.1). In particular, for a given target record $s_q$ and its $B^{ed}_d$-tree $T$, the $VO$ only includes the non-candidate nodes $N$ in $T$ but not all children strings that are covered by $N$. Therefore, the client is certain that all children strings of $N$ are dissimilar to $s_q$, and avoids a large number of edit distance calculations.

4 VO Construction for A Single Target Record

In this section, we consider a set of similar record pairs $\{(s_q, s_1), (s_q, s_2), \ldots, (s_q, s_t)\}$ for a single target record $s_q$. We design a verification method of similarity search ($VS^2$) approach. $VS^2$ constructs a proof that proves $S = \{s_1, s_2, \ldots, s_t\}$ include all strings (records) that are similar to $s_q$ (completeness), and all strings (records) in $S$ is indeed similar to $s_q$ (correctness), where $\theta$ is the similarity threshold. $VS^2$ consists of three phases: (1) the preprocessing phase in which the data owner constructs the authenticated data structure $T$ of the dataset $D$. Both $D$ and the root signature $sig$ of $T$ are outsourced to the server; (2) the matching phase in which the server executes the approximate matching on $D$ to find similar strings for $s_q$, and constructs the verification object (VO) of the search results $M^S$. The server returns both $M^S$ and $VO$ to the data owner; and (3) verification phase in which the data owner verifies the integrity of $M^S$ by leveraging $VO$. Next we explain the details of these three phases.
4.1 Pre-Processing

In this one-time phase, the data owner constructs the authenticated data structure of the dataset $D$ before outsourcing $D$ to the server. We design a new authenticated data structure named the Merkle $B^{ed}$ tree (MB-tree). Next, we explain the details of MB-tree.

The MB-tree is constructed on top of the $B^{ed}$ tree by assigning the digests to each $B^{ed}$ node. In particular, every $B^{ed}$-tree node contains a triple $(N_b, N_e, h_N)$, where $N_b, N_e$ correspond to the string range values associated with $N$, and $h_N$ is the digest value computed as $h_N = h(h(N_b)\|h(N_e)|h^{-1}(N))$, where $h^{-1}$ is the inverse of $h$. If $N$ is a leaf node, then $N_1, \ldots, N_f$ are the strings $s_1, \ldots, s_f$ covered by $N$. Besides the triple, each MB-tree node contains multiple entries. In particular, for any leaf node $N$, assume it covers $f$ strings. Then it contains $f$ entries, each of the format $(s, p)$, where $s$ is a string that is covered by $N$, and $p$ is the pointer to the disk block that stores $s$. For any intermediate node, assume that it has $f$ children nodes. Then it contains $f$ entries, each entry consisting of a pointer to one of its children nodes.

The digests of the MB-tree $T$ can be constructed in the bottom-up fashion, starting from the leaf nodes. After all nodes of $T$ are associated with the digest values, the data owner signs the root with her private key. The signature can be created by using a public-key cryptosystem (e.g., RSA).

An example of the MB-tree structure is presented in Figure 1. The data owner sends both $D$ and $T$ to the server. The data owner keeps the root signature of $T$ locally, and sends it to any client who requests for it for authentication purpose.

Following [6][13], we assume that each node of the MB-tree occupies a disk page. For the constructed MB-tree $T$, each entry in the leaf node occupies $|s| + |p|$ space, where $|p|$ is the size of a pointer, and $|s|$ is the maximum length of a string value. The triple $(N_b, N_e, h_N)$ takes the space of $2|s| + h$, where $|h|$ is the size of a hash value. Therefore, a leaf node can have $f_1 = \left\lfloor \frac{P - 2|s| - |h|}{|p|} \right\rfloor$ entries at most, where $P$ is the page size. Given $n$ unique strings in the dataset, there are $\left\lfloor \frac{n}{P} \right\rfloor$ leaf nodes in $T$. Similarly, for the internal nodes, each entry takes the space of $|p|$. Thus each internal node can have at most $f_2 = \left\lfloor \frac{P - 2|s| - |h|}{|p|} \right\rfloor$ entries (i.e., $\left\lfloor \frac{P - 2|s| - |h|}{|p|} \right\rfloor$ children nodes). Therefore, the height $h$ of $T$ is $h \geq \log_{f_2} \left\lceil \frac{n}{P} \right\rceil$.

4.2 VO Construction

For each target string $s_q$, the server constructs a verification object $VO$ to show that the matching result $M^S$ is both sound and complete.

First, we define false hits. Given a target string $s_q$ and a similarity threshold $\theta$, the false hits of $s_q$, denoted as $F$, are all the strings that are dissimilar to $s_q$. In other words, $F = \{s|s \in D, s_q \neq s\}$. Intuitively, to verify that $M^S$ is sound and complete, the $VO$ includes both similar strings $M^S$ and false hits $F$. Apparently including all false hits may lead to a large $VO$, and thus high network communication cost and the verification cost at the data owner side. Therefore, we aim to reduce the $VO$ size of $F$.

Before we explain how to reduce $VO$ size, we first define C-strings and NC-strings. Apparently, each false hit string is covered by a leaf node of the MB-tree $T$. Based on whether a leaf node in MB-tree is a candidate, the false hits $F$ are classified into two types:

- **C-strings**: the strings that are covered by candidate leaf nodes; and
- **NC-strings**: the strings that are covered by non-candidate leaf nodes.

Our key idea to reduce $VO$ size of $F$ is to include representatives of NC-strings instead of individual NC-strings. The representatives of NC-strings take the format of maximal false hit subtrees (MFs). Formally, given a MB-tree $T$, we say a subtree $T^N$ that is rooted at node $N$ is a false hit subtree if $N$ is a non-candidate node. We say the false hit subtree $T^N$ rooted at $N$ is maximal if the parent of $N$ is a candidate node. The MFs can root at leaf nodes. Apparently, all strings covered by the MFs must be NC-strings. And each NC-string must be covered by a MF node. Furthermore, MFs are disjoint (i.e., no two MFs cover the same string). Therefore, instead of including individual NC-strings into the VO, their MFs are included.

As the number of MFs is normally much smaller than the number of NC-strings, this can effectively reduce $VO$ size. Now we are ready to define the $VO$.

**Definition 4.1** Given a dataset $D$, a target string $s_q$, let $M^S$ be the returned similar strings of $s_q$. Let $T$ be the MB-tree of $D$, and $NC$ be the strings that are covered by non-candidate nodes of $T$. Let $M$ be a set of MFs of $NC$. Then the $VO$ of $s_q$ consists of: (i) string $s$, for each $s \in D - NC$; and (ii) a pair $(N, h^{-1})$ for each $MF \in M$ that is rooted at node $N$, where $N$ is represented as $[N_b, N_e]$, with $[N_b, N_e]$ the string range associated with $N$, and $h^{-1} = h(h_C_1 || \ldots || h_C_f)$, 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 $s_1, \ldots, s_f$ covered by $N$. Furthermore, in $VO$, a pair of
brackets is added around the strings and/or the pairs that share the same parent in T.

Intuitively, in VO, the similar strings and C-strings are present in the original string format, while NC-strings are represented by the MFs (i.e., in the format of \( ([N_b, N_c], h_N) \)).

**Example 4.1** Consider the MB-tree \( T \) in Figure 7 and the string \( s_1 \) to construct VO. Assume the similar strings are \( M^S = \{s_3, s_5\} \). Also assume that node \( N_3 \) of \( T \) is the only non-candidate node. Then NC-strings are \( NC = \{s_7, s_8, \ldots, s_{12}\} \), and C-strings are \( \{s_2, s_4, s_6\} \). The set of MFs \( M = \{N_3\} \). Therefore,

\[
VO = \{((s_1, s_2, s_3), (s_4, s_5, s_6)), ([s_7, s_{12}], h^{-12})\},
\]

where \( h^{-12} = h(h_{N_3}||h_{N_4}) \),

\[
h_{N_4} = h(h(s_7)||h(s_9)||h(h(s_7)||h(s_9)||h(s_9)),\]

and \( h_{N_7} = h(h(s_{10})||h(s_{12})||h(h(s_{10})||h(s_{11})||h(s_{12}))).\)

For each MF, we do not require that \( h(N_b) \) and \( h(N_c) \) appear in VO. However, the server must include both \( N_b \) and \( N_c \) in VO. This is to prevent the server from cheating on the non-candidate nodes by including insound \([N_b, N_c]\) in VO. More details of the robustness of our authentication procedure can be found in Section 6.

### 4.3 Authentication Phase

For a given target string \( s_q \), the server returns \( VO \) together with the matching set \( M^S \) to the client. The verification procedure consists of three steps. In Step 1, the client re-constructs the MB-tree from \( VO \). In Step 2, the client re-computes the root signature \( \text{sig}^{\prime} \) and compares \( \text{sig}^{\prime} \) with \( \text{sig} \). In Step 3, the client re-computes the edit distance between \( s_q \) and a subset of strings in \( VO \). Next, we explain the details of these steps.

**Step 1: Re-construction of MB-tree:** First, the client sorts the strings and string ranges (in the format of \([N_b, N_c]\)) in \( VO \) by their mapping values according to the string ordering scheme. String \( s \) is put ahead of the range \([N_b, N_c]\) if \( s < N_b \). It returns a total order of strings and string ranges. If there exists any two ranges \([N_b, N_c]\) and \([N'_b, N'_c]\) that overlap, the client concludes that the \( VO \) is not sound. If there exists a string \( s \in M^S \) and a range \([N_b, N_c]\) in \( VO \) such that \( s \in [N_b, N_c] \), the client concludes that \( M^S \) is not sound, as \( s \) is indeed a dissimilar string (i.e., it is included in a non-candidate node). Second, the client maps each string \( s \in M^S \) to an entry in a leaf node in \( T \), and each pair \([N_b, N_c], h_N\) 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 signature:** After the MB-tree \( T \) is re-constructed, the client computes the root signature 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 \( M^S \)-tree. For each internal node that corresponds to a pair \([N_b, N_c], h^{1-f}\) in \( VO \), the client computes the hash \( h_N \) of \( N \) as \( h_N = h(h(N_b)||h(N_c)||h^{1-f}) \). Finally, the client re-computes the hash value of the root node, and rebuilds the root signature \( \text{sig}^{\prime} \) by signing \( h_{root} \) using the data owner’s public key. The client then compares \( \text{sig}^{\prime} \) with \( \text{sig} \). If \( \text{sig}^{\prime} \neq \text{sig} \), the client concludes that the server’s results are not sound.

**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 C-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., \( s \) is a similar string indeed), the client concludes that the server fails the completeness verification. Third, for each range \([N_b, N_c]\) in \( VO \), the client verifies whether \( DST_{\min}(s_q, N) > \theta \), where \( N \) is the corresponding MB-tree node associated with the range \([N_b, N_c]\). If it is not (i.e., \( N \) is indeed a candidate node), the client concludes that the server fails the completeness verification.

**Example 4.2** Consider the MB-tree in Figure 7 as an example, and the target string \( s_1 \). Assume the similar strings \( M^S = \{s_3, s_5\} \). Consider the VO shown in Example 4.1. The C-strings are \( C = \{s_2, s_4, s_6\} \). After the client re-constructs the MB-tree, it re-computes the hash values of strings \( M^S \cup C \). It computes the root signature \( \text{sig}^{\prime} \) from these hash values. It also performs the following distance computations: (1) for \( M^S = \{s_3, s_5\} \), compute the edit distance between \( s_1 \) and string in \( M^S \); (2) for \( C = \{s_2, s_4, s_6\} \), compute the edit distance between \( s_1 \) and any C-string in \( C \); and (3) for the pair \([[s_7, s_{12}], h_{N_7}] \) in \( VO \), compute \( DST_{\min}(s_q, N) \).

### 4.4 Complexity Analysis

We formally analyze the time and space complexity of the three phases of \( V S^2 \), and present the result in Table 1.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Measurement</th>
<th>( V S^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-processing</td>
<td>Time</td>
<td>( O(n) )</td>
</tr>
<tr>
<td></td>
<td>Space</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>VO construction</td>
<td>Time</td>
<td>( O(n) )</td>
</tr>
<tr>
<td></td>
<td>VO Size</td>
<td>( O((n_C + n_R)\sigma_S + n_M\sigma_M) )</td>
</tr>
<tr>
<td>Verification</td>
<td>Time</td>
<td>( O((n_R + n_M + n_C)C_{Ed}) )</td>
</tr>
</tbody>
</table>

Table 1: Complexity analysis of \( V S^2 \)

(\( n \): # of strings in \( D \); \( \sigma_S \): AVG. length of string; \( \sigma_M \): AVG. size of MB-tree node; \( n_M \): # of MF nodes; \( n_R \): # of strings in \( M^S \); \( n_C \): # of C-strings; \( C_{Ed} \): the complexity of an edit distance computation)}
which results in $O(n)$ time complexity, where $n$ is the number of records in the dataset $D$. As the total number of nodes in the MB-tree is $O\left(\frac{n}{f}\right)$, where $f$ is the fanout of the nodes, the total time complexity of the pre-processing phase is $O(n)$. Regarding the space complexity, as the data owner only needs to keep the root signature of the MB-tree locally, the storage overhead is only $O(1)$.

Regarding the VO construction phase, the server traverse the MB-tree to construct the VO. In the worst case, the server scans all the MB-tree nodes to find $C$-strings and MF nodes. Thus, the time complexity is $O(n)$. The VO size of the $VS^2$ approach is calculated as the sum of two parts: (1) the total size of the similar strings and $C$-strings (in string format), which is $O((n_R + n_C)\sigma_S)$ and (2) the size of MF nodes, which is $O(n_M F\sigma_M)$. Note that $\sigma_M = 2\sigma_S + |h|$, where $|h|$ is the size of a hash value. In our experiments, it turned out that $\sigma_M/\sigma_S \approx 20$.

Regarding the authentication phase, in order for the client to authenticate the matching results based on the VO, the main time complexity comes from calculating the string edit distance, as it is very efficient to compute the hash values and the root signature of the MB-tree. Specifically, for each similar string and $C$-string, the client calculates its edit distance to the target string. Besides, the completeness verification requires to compute the edit distance $DST_{\text{min}}(s_i, N)$, for each $MF$ node $N$. Thus, the complexity of VO verification is $O((n_R + n_M F + n_C)C_{Ed})$.

5 VO Construction for Multiple Target Records

So far we discussed the authentication of record matching of a single target string (record). To authenticate the matching result for multiple target strings (records) $S = \{s_1, \ldots, s_t\}$, a straightforward solution is to create VO for each string $s_i \in S$ and its similarity result $M^S(s_i)$. Apparently this solution may lead to $VO$ of large sizes in total. Thus, we aim to reduce the size of VO. Our VO optimization method consists of two main strategies: (1) optimization by triangle inequality; and (2) optimization by overlapping dissimilar strings.

5.1 Optimization by Triangle Inequality

It is well known that the string edit distance satisfies the triangle inequality, i.e., $|DST(s_i, s_k) - DST(s_j, s_k)| \leq DST(s_i, s_j) \leq DST(s_i, s_k) + DST(s_j, s_k)$. Therefore, consider two strings $s_i, s_j \in D$, assume the the server has executed the approximate matching of $s_i$ and prepared the VO of the results. Then consider the authentication of the search results of string $s_j$, for any string $s \in D$ such that $DST(s_i, s) - DST(s_j, s) > \theta$, there is no need to prepare the proof of $s$ showing that it is a false hit for $s_j$. A straightforward method is to remove $s$ from the $MB$-tree $T$ for $VO$ construction for the string $s_j$. Now the question is whether removing $s$ always lead to VO of smaller sizes. We have the following theorem.

**Theorem 5.1** Given a string $s$ and a $MB$-tree $T$, let $N \in T$ be the corresponding node of $s$, and $N_p$ be the parent of $N$ in $T$. Let $C(N_p)$ be the set of children of $P$ in $T$, and $N'_p$ be the node constructed from $C(N_p) \setminus \{N\}$ (i.e., the children nodes of $N_p$ excluding $N$). Then if $N_p$ is a non-candidate, it must be true that $N'_p$ must be a non-candidate.

Let $[N_b, N_c]$ be the range of $N_p$. The proof of Theorem 5.1 considers two cases: (1) $s \neq N_b$ and $s \neq N_c$. Removing $s$ will not change $N_b$ and $N_c$, which results in that both $N'_b$ and $N'_c$ have the same range $[N_b, N_c]$. (2) $s = N_b$ or $s = N_c$. Removing $s$ from $C(N_p)$ changes the range of $N_p$ to $[N'_b, N'_c]$. Note that it must be true $[N'_b, N'_c] \subseteq [N_b, N_c]$. Therefore, $N'_p$ must be a non-candidate.

Based on Theorem 5.1 removing the MB-tree nodes that correspond to any dissimilar string does not change the structure of the MB-tree. Therefore, we can optimize the VO construction procedure by removing those strings covered by the triangle inequality from the MB-tree.

Another possible optimization is that for any two target strings $s_i$ and $s_j$, for any string $s \in D$ such that $DST(s, s_i) + DST(s, s_j) \leq \theta$, the client does not need to re-compute $DST(s, s_j)$ to prove that $s_i$ is a similar string in the results. We omit the details due to the space limit.

5.2 Optimization by Overlapped Dissimilar Strings

Given multiple target records (strings), the key optimization idea is to merge the VOs of various target strings. This is motivated by the fact that any two strings $s_i$ and $s_j$ may share a number of dissimilar strings. These shared dissimilar strings can enable to merge the VOS of $s_i$ and $s_j$. Note that simply merging all similar strings of $s_i$ and $s_j$ into one set and constructing MB-tree of the merged set is not correct, as the resulting MB-tree may deliver non-leaf MFs that are candidates to both $s_i$ and $s_j$. Therefore, given two target strings $s_i$ and $s_j$ such that their false hits overlap, let $NC_i$ ($NC_j$, resp.) be the $NC$-strings of $s_i$ ($s_j$, resp.), the server finds the overlap of $NC_i$ and $NC_j$. Then the server constructs $VO$ that shares the same data structure on these overlapping strings. In particular, given the overlap $O = NC_i \cap NC_j$, the server constructs the non-leaf MFs from $O$, and include the constructed MFs in the VOs of both $s_i$ and $s_j$.

6 Security Analysis

Given a target string $s_q$ and a similarity threshold $\theta$, let $M$ ($F$, resp.) be the similar strings (false hits, resp.) of $s_q$. Let $M^S$ be the similar strings of $s_q$ returned by the server. An untrusted server may perform the following cheating behaviors the real results $M^S$: (1) tampered values: some
The server constructs the $M^S$, $10, 100, 1000, 10000$, $2, 3, 4, 5, 6, 7$  

Theorem 6.1  
Given a target string $s_q$ and the non-candidate node $N$ of range $[N_b, N_e]$, including any string $s'$ into $N$ such that $s' \approx s_q$ will change $N$ to be a candidate node.

The proof is straightforward. It is easy to see that $s' \notin [N_b, N_e]$. Therefore, including $s'$ into $N$ must change the range to be either $[s', N_e]$ or $[N_b, s']$, depending on whether $s' < N_b$ or $s' > N_e$. Now it must be true that $\text{DST}_{\text{min}}(s_q, N) \leq \theta$, as $\text{DST}(s', s_q) \leq \theta$.

Following Theorem 6.1 for any string $s \in SS$ that is considered a NC-string in $VO$, the client can easily catch it by verifying whether $\text{DST}_{\text{min}}(s_q, N) > \theta$, for any non-candidate node (i.e., Step 3 of the authentication procedure).

7 Experiments  

In this section, we report the experiment results.  

7.1 Experiment Setup  

Datasets and queries. We use two real-world datasets: (1) the Authors dataset from DBLP that contains 260K last names. The maximum length of a name is 63, while the average is 14.627; and (2) the Authors dataset from DBLP including 1,000,000 researcher names. The maximum length of a name is 99 while the average length is 14.732. 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.

Experimental environment. We implement the $VS^2$ 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 brute-force approximate string matching algorithm that calculates the edit distance between every string in the dataset and the target string $s_q$.

Parameter setup. The parameters include: (1) string edit distance threshold $\theta$, and (2) the fanout $f$ of $MB$-tree nodes (i.e., the number of entries that each node contains). The details of the parameter settings can be found in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity threshold $\theta$</td>
<td>2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>MB-tree fanout $f$</td>
<td>10, 100, 1000, 10000</td>
</tr>
</tbody>
</table>

Table 2: Parameter settings  

In the discussions, we use the following notations: (1) $n$: the number of strings in the dataset, (2) $n_{MB}$: the number of similar strings, (3) $n_{MFS}$: the number of $MF$s, and (4) $n_{C}$: the number of $C$-strings. We use $\sigma_S$ to indicate the average string size, and $\sigma_M$ as the size of $MB$-tree node.
7.2 VO Construction Time

We measure the VO construction time at the server side. The impact of \( \theta \). In Figure 2, we show the VO construction time with regard to different \( \theta \) values on both datasets. First, in all our experiments, the VO construction time is very short; it never exceeds 11 seconds for both datasets. It only needs around 3 seconds for the Actor dataset. Second, we observe that the VO construction time grows when \( \theta \) increases. This is because \( n_{MF} \) reduces with the increase of \( \theta \). When \( \theta \) is small, a node \( N \) may be a \( MF \), because \( DST_{min}(s_q, N) > \theta \). While with a larger \( \theta \) values, the node \( N \) becomes a candidate. Thus, the server needs to search deep in the \( MB \)-tree to find the \( MF \) nodes, if there is any. Third, we observe the dramatic increase in VO construction time when \( \theta \) changes from 2 to 3. With further analysis, we discover that the change in VO construction time is consistent with how \( n_{MF} \) varies. For example, on the Actors dataset, when \( \theta = 2 \), \( n_{MF} = 2,958 \). However, \( n_{MF} \) decreases to 649 when \( \theta = 3 \). The abrupt reduction in \( n_{MF} \) leads to the immense increment effort that the server devotes to traverse the \( MB \)-tree to find \( MFs \). Fourth, the VO construction time on the Authors dataset is larger than that on the Actors dataset. This is because with the same fanout \( f \), the \( MB \)-tree’s size of the Authors dataset is larger, due to the greater data size.

![Figure 2: VO construction time w.r.t. \( \theta \) (\( f = 100 \))](image)

(a) The Actors dataset  (b) The Authors dataset

Figure 3: VO construction time w.r.t. \( f \)

The impact of \( f \). From Figure 3, we observe that 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. The observation from Figure 3 suggests that small fanout values fit our \( VS^2 \) method better.

7.3 VO Size

We measure the size of the VO constructed by \( VS^2 \).

![Figure 4: VO size w.r.t. \( \theta \) (\( f = 100 \))](image)

(a) The Actors dataset  (b) The Authors dataset

Figure 5: VO size w.r.t. \( f \)

The impact of \( \theta \). The results are shown in Figure 4. The observation is that the VO size increases with the growth of \( \theta \) value. Apparently, larger \( \theta \) values lead to more similar strings (i.e., larger \( n_R \)), fewer \( MFs \) (i.e., smaller \( n_{MF} \)), and fewer \( C \)-strings (i.e., smaller \( n_C \)). The VO size is decided by \( (n_R + n_C)\sigma_S + n_{MF}\sigma_M \), where \( \frac{\sigma_M}{\sigma_S} \approx 20 \). When we increase \( \theta \), even though \( n_{MF} \) reduces, the intensive growth of \( n_R \) and \( n_C \) leads to the increase of VO size. For example, on the Actors dataset, when \( \theta = 2 \), \( n_R + n_C = 126,858 \), \( n_{MF} = 2,958 \). While when \( \theta = 3 \), \( n_R + n_C = 230,821, n_{MF} = 649 \).

The impact of \( f \). As shown in Figure 5, the VO size increases with the growth of the fanout \( f \). Recall that the VO size is \( (n_R + n_C)\sigma_S + n_{MF}\sigma_M \). When \( f \) increases, \( n_R \) is unchanged. Meanwhile, \( n_C \) dramatically increases with \( f \) (e.g., on the Actors dataset, when \( f \) increases from 100 to 1,000, \( n_C \) increases by 122,515). Furthermore, when \( f \) increases, \( n_{MF} \) decreases by a small degree (when \( f \) changes from 100 to 1,000, \( n_{MF} \) decreases by 2,937). Consider that \( \frac{\sigma_M}{\sigma_S} \approx 20 \). The intensive increase of \( n_C \) leads to a larger VO size. We observe the sharp increase in VO size when \( f \) increases from 100 to 1,000. This is because at this point, \( n_C \) experiences the maximum extent of growth.

7.4 VO Verification Time

In this section, we measure the VO verification time at the data owner side. We compare the VO verification time with the performance of our baseline algorithm, aiming to show that the client’s verification effort is less than by doing string matching locally.
The impact of \( \theta \). In Figure 6, we report the VO verification time of \( VS^2 \). First, we observe that the VO verification time is always smaller than the baseline. In particular, \( VS^2 \) can save up to 87\% computational effort at the data owner side if she outsources the string matching to the server and does verification locally. This proves that our verification method is suitable for the outsourcing setting. Second, the VO verification time increases when \( \theta \) increases. Recall that the complexity of VO verification is \( O((n_R + n_C + n_{MF})C_{Ed}) \). This is because when \( \theta \) increases, \( n_{MF} \) decreases, while both \( n_R \) and \( n_C \) increase. As \( n_R \) and \( n_C \) increase faster than the decrease of \( n_{MF} \), it leads to more edit distance calculations for the verification. Therefore, the total verification time increases.

The impact of \( f \). We also measure the VO verification time of \( VS^2 \) with various fanout values. On both datasets, the VO verification is always more efficient than the baseline method. When the fanout is 10, \( VS^2 \) can save up to 90\% computational effort at the client side. Besides, the verification time increases when \( f \) grows. This is straightforward as larger \( f \) results in the growth of the number of C-strings. This requires more calculations for the edit distance between C-strings and the target string.

In sum, the experiment results suggest that \( VS^2 \) is an efficient authentication approach to verify the result of outsourced record matching computations. In particular, \( VS^2 \) is more suitable for the setting in which the similarity threshold is small and the MB-tree’s fanout is small.

8 Related Work

The security problem of outsourced computations caught much attention in recent years. Based on the the perspective to enforce security and the type of computations that are executed on the data, we classify these techniques into the following types: (1) authentication of outsourced SQL query evaluation, (2) authentication of keyword search, and (3) privacy-preserving record linkage. There are also other work on authentication of data integration \cite{10} and network aggregation in distributed systems (e.g. \cite{18,27}). None of these work considered the authentication of outsourced approximate string matching.

Authentication of outsourced SQL query evaluation. The issue of providing authenticity for outsourced database was initially raised in the database-as-a-service (DaS) paradigm \cite{10}. The aim is to assure the correctness of SQL query evaluation over the outsourced databases. The proposed solutions include Merkle hash trees \cite{13,15}, signatures on a chain of paired tuples \cite{17}, and authenticated B-tree and R-tree structures for aggregated queries \cite{14}. The key idea of these techniques is that the data owner outsources not only data but also the endorsements of the data being outsourced. These endorsements are signed by the data owner against tampering with by the service provider. For the cleaning results, the service provider returns both the results and a proof, called the verification object (VO), which is an auxiliary data structure to store the processing traces such as index traversals. The client uses the VO, together with the answers, to reconstruct the endorsements and thus verify the authenticity of the results. An efficient authentication technique should minimize the size of VO, while requiring lightweight authentication at the client side. In this paper, we follow the same VO-based strategy to design our authentication method.

Authentication of keyword search. Pang et al. \cite{16} targeted at search engines that perform similarity-based document retrieval, and designed a novel authentication mechanism for the search results. The key idea of the authentication is to build the Merkle hash tree (MHT) on the inverted index, and use the MHT for VO construction. Though effective, it has several limitations, e.g., the MHT cannot deal data updates efficiently \cite{9}. To address these limitations, Goodrich et al. \cite{9} designed a new model that considered conjunctive keyword searches as equivalent with a set intersection on the underlying inverted index data structure. They use the authenticated data structure in \cite{20} to verify the correctness of set operations.

Privacy-preserving record linkage. The privacy of sensitive information in the outsourced record matching has been the focus of research for two decades. Schnell et al. \cite{23} and Durham et al. \cite{8} propose to store the q-grams in the Bloom Filters (BFs); the BFs of two similar strings (i.e., many q-grams in common) have a large number of identical bit positions set to 1. However, Schnell et al. \cite{23} observed that the BFs are still open to the frequency attack. Storer et al. \cite{24} consider the duplicates as exact identical contents. They use convergent encryption that uses a function of the hash of the plaintext of a chunk as the encryption key, so that any identical plaintext values will be encrypted to identical ciphertext values. However, the convergent encryption
disables the approximate matching on outsourced records. Bonomi et al. propose a new transformation method that integrates embedding methods with differential privacy. Its embedding strategy projects the original data on a base formed by a set of frequent variable length grams. The privacy of the gram construction procedure is guaranteed to satisfy differential privacy. Atallah et al. design a SMC protocol based on homomorphic encryption for record linkage based on edit distance similarity.

9 Conclusion

In this paper, we designed an efficient authentication method, namely ARM, for outsourced approximate record matching. ARM is based on a novel authentication data structure named MB-tree that integrates both B-ed-tree and Merkle hash tree. Experimental results show that our methods can authenticate the record matching result efficiently. In the future, we will investigate how to reduce the VO size as well as the verification time.

10 Acknowledgment

This material is based upon work supported by the U.S. National Science Foundation under Grant No. 1350324.

References