PraDa: Privacy-preserving Data-Deduplication-as-a-Service

Boxiang Dong
bdong@stevens.edu

Ruilin Liu
rliu3@stevens.edu

Wendy Hui Wang
Hui.Wang@stevens.edu

Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA, 07030

ABSTRACT

The data-cleaning-as-a-service ($DCaS$) paradigm enables users to outsource their data and data cleaning needs to computationally powerful third-party service providers. It raises several security issues. One of the issues is how the client can protect the private information in the outsourced data. In this paper, we focus on data deduplication as the main data cleaning task, and design two efficient privacy-preserving data-deduplication methods for the $DCaS$ paradigm. We analyze the robustness of our two methods against the attacks that exploit the auxiliary frequency distribution and the knowledge of the encoding algorithms. Our empirical study demonstrates the efficiency and effectiveness of our privacy preserving approaches.

Categories and Subject Descriptors

H.2.0 [**General**]: Security, integrity, and protection

Keywords

Data-cleaning-as-a-service, data deduplication, outsourcing, security, privacy-preserving

1. INTRODUCTION

It has been well known that real-world datasets, particularly those from multiple sources, tend to be *dirty* - inaccurate, incomplete, and inconsistent. According to a recent survey [8], around 40% of companies have suffered losses, problems, or costs due to poor quality data. Dirty data are commonly present in both single data collections (e.g., due to misspellings during data entry, missing information or other invalid data) or multiple data sources. Many data management and decision making applications, e.g., data warehouses and web-based information systems, require extensive support for data cleaning.

Data cleaning is a labor-intensive and complex process [3]. Providing a data cleaning solution that is powerful, reliable, and easy to use is extremely challenging. With corporate data exploding in volume and variety, cleaning of large scale data has grown difficult. However, building in-house tools for data cleaning is highly labor-intensive, and may create maintenance problems down the road. Alternatively, purchasing expensive proprietary data cleaning solutions may not be acceptable by those medium- and small-size organizations that have limited budgets. A possible solution to resolve this dilemma is to outsource the data to a third-party data cleaning service provider for efficient data cleaning. This motivates the *data-cleaning-as-a-service* (DCaS) paradigm that enables organizations with limited computational resources to outsource their data cleaning needs to a third-party service provider. Recently, several academic and industrial organizations have started investigating and developing technologies and infrastructure for cloud-based data cleaning services (e.g., OpenRefine [1]).

In this paper, we focus on *data deduplication*, one of the important data cleaning problems, that aims to identify the records that refer to the same entity. Outsourcing data deduplication to a third-party service provider raises a few issues. For instance, when the outsourced data involves any personal information (e.g., medical or financial details of individuals), the privacy of those information needs to be carefully protected. Indeed, personal information is commonly required for data deduplication to match records from different databases [5]. It is therefore paramount to protect data privacy when outsourcing the data to the $DCaS$ provider.

A concept that is highly related to data deduplication is *record linkage*, which shares the same goal as data deduplication as finding similar records in the datasets. There has been extensive study of the privacy-preserving record linkage ($PPRL$) problem that enables linking large databases across organizations while preserving the privacy of the entities stored in these databases. The existing $PPRL$ solutions consider either the 2-party that consists of two private database owners (e.g., [2, 19]) or 3-party that compromises of a trusted third-party besides the two data owners (e.g., [12, 21]). Unlike these existing $PPRL$ techniques, we consider an asymmetric paradigm that consists of two parties where one is computationally weak and the other one is much more powerful. We call the weak party *client* and the other party *server* in the rest of the paper.

We assume the server is untrusted. Therefore, to protect the sensitive information in the outsourced dataset, the client encodes its data and sends the encoded data to the server. The attacker (i.e., the server) may try to decode the received dataset to infer the private information. We

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661863.

assume that the attacker has (either exact or approximate) frequency distribution information of the outsourced data, and will launch the *frequency analysis attack* by exploiting the frequency knowledge to break the encoding. Furthermore, we assume that the attacker knows the details of the encoding scheme; he can exploit such knowledge to launch the *known-scheme attack*, trying to recover the original data.

In general, the client can use *salt*, a random value that is different for each tuple, as part of the encoding scheme to prevent the frequency analysis attack. However, adding salt may disable to find duplicated records from the encoded data. On the other hand, using hash values [5], Bloom filters [22], and embedding techniques [21] still enable one-to-one mapping between the original and encoded data values, thus cannot defend against the frequency analysis attack.

Contributions. In this paper, we design the Privacy-preserving Data-Deduplication-as-a-Service ($PraDa$) system that enables the client to outsource her data as well as data deduplication needs to a potentially untrusted server. $PraDa$ enables the server to discover near duplicated records from the encoded data, while providing provable privacy guarantee against both the frequency analysis and the known-scheme attacks. To the best of our knowledge, we are the first to design privacy-preserving techniques for the $DCaS$ paradigm against the frequency analysis attack. In particular, our contributions include the following.

First, we design the *locality-sensitive hashing* based ($LSHB$) scheme that utilizes locality-sensitive hashing (LSH) to map strings to hash values such that similar strings share the same value with high probability [15]. To defend against the frequency analysis attack, we design the split-and-duplication method to ensure that the LSH values always behave a uniform frequency distribution. We analyze the privacy guarantee of the $LSHB$ approach against both the frequency analysis and the known-scheme attacks. The $LSHB$ approach can deal with a number of distance measurement metrics for categorical data.

Second, we design the embedding and homophonic substitution (EHS) scheme. EHS first maps string values to a Euclidean space in the way that similar strings are mapped to close Euclidean points. To defend against the frequency analysis attack, all identical Euclidean points are encoded as different but close points. The frequency distribution of the encoded Euclidean points is always flattened, regardless of the frequency distribution of original data. We analyze the robustness of EHS against both the frequency analysis and the known-scheme attacks. The EHS approach can deal with both categorical and numerical data.

Finally, we evaluate and compare the performance of both $LSHB$ and EHS in terms of time complexity and data deduplication accuracy, on real datasets. Our results show that both approaches guarantee high recall (i.e., the percentage of original duplicated records that are returned by the server). In particular, EHS always guarantees 100% recall. $LSHB$ delivers lower recall (around 70%), but it has better time performance than EHS at both client and server sides. For both approaches, the client's computational effort is much cheaper than the server as well as executing data deduplication locally.

The rest of the paper is organized as follows. Section 2 describes the preliminaries. Section 3 and 4 discuss our $LSHB$ and EHS methods respectively. Section 5 presents the post-processing procedure at the client side. Section 6 provides the experiment results. Section 7 discusses the related work. Section 8 concludes the paper.

2. PRELIMINARIES

2.1 Outsourcing Framework

Record linkage techniques are used to link together records which relate to the same entity (e.g. patient or customer) in one or more data sets where a unique identifier is not available. The key of the most record linkage techniques is to measure the similarity of records. Normally two records S_1 and S_2 are *δ-similar*, denoted as $S_1 \simeq S_2$, if their similarity according to some distance measurement metrics is no less than a given threshold δ. In this paper, we consider the *Jaccard* similarity based on q-grams. But we also discuss how to extend our methods to other distance metrics such as edit distance and hamming distance.

We consider a client-server framework that contains a data owner as the client, and the service provider as the server. The client has a private dataset D and would like to find out all record pairs in D that are near-duplicates. To protect the private information in D, she transforms D to D', and sends D' together with the similarity threshold to the server. When the server receives D', it executes the data deduplication on D' to find all data pairs that are similar (e.g., their distance is less than the client's threshold), and returns the results to the client.

2.2 Attack Model

Since the server is potentially untrusted, the client will send the encoded records to the server for data deduplication. We assume the server is *semi-honest*, meaning that it follows the protocols honestly but it is curious to extract private information. In this paper, we consider two specific attacks, namely the *frequency analysis attack* and the *known-scheme attack*.

Frequency analysis attack. We assume the attacker may possess the knowledge of the domain values of the outsourced data, as well as the frequency distribution of the data values at attribute level. As shown previously [22, 7], such adversary knowledge can be easily obtained in real-world applications. We assume that the attacker has the exact frequency occurrence of attribute values. The attacker can launch the frequency analysis attack by matching the encoded data with the original ones of the same frequency. We assume that the attacker has no prior knowledge about the correlations of data values.

Known-scheme attack. We assume the attacker knows the details of the encoding methods that are used by the client. He may try to break the encoding scheme by utilizing the knowledge of the encoding methods.

2.3 Precision and Recall

We define *precision* and *recall* to measure the impact of privacy-preserving techniques to the accuracy of record linkage. Formally, given a dataset D, let R_O be the similar strings in D, and R_E be the pairs of similar strings (possibly in encrypted format) returned by the service provider. The *precision* is measured as $Pre = \frac{|R_E \cap R_O|}{|R_E|}$, and the *recall* is measured by $Rec = \frac{|R_E \cap R_O|}{|R_O|}$. Intuitively, the precision measures the fraction of near-duplicates by the encoding method that are correct, while the recall measures the fraction of original near-duplicates that are preserved by the encoding methods. We aim at designing the privacy-preserving tech-

niques that achieve provable privacy guarantee with high precision/recall.

3. LOCALITY-SENSITIVE HASHING BASED (LSHB) APPROACH

In this section, we propose two locality-sensitive hashing based ($LSHB$) approaches, namely the *duplication* approach (Section 3.1) and the *split-and-duplication* approach (Section 3.2). The key idea of the two $LSHB$ methods is to map the strings to locality-sensitive hashing (LSH) values. LSH is a set of hash functions that map objects into several buckets such that similar objects share a bucket with high probability, while dissimilar ones do not. Formally, let O be the domain of objects, and $dist()$ be the distance measure between two objects. Then,

DEFINITION 3.1. [(d_1, d_2, p_1, p_2)-sensitive hashing] A function family $\mathcal{H}$ is called *(d_1, d_2, p_1, p_2)-sensitive* if for any two objects $o_1, o_2 \in O$: (1) If $dist(o_1, o_2) < d_1$, then $Pr_{\mathcal{H}}[h(o_1) = h(o_2)] \geq p_1$; (2) If $dist(o_1, o_2) > d_2$, then $Pr_{\mathcal{H}}[h(o_1) = h(o_2)] \leq p_2$, where $0 \leq d_1 < d_2 \leq 1$, and $0 \leq p_1, p_2 \leq 1$.

A family is interesting when $p_1 > p_2$. Intuitively, any two similar objects will have the same LSH value with high probability, while any two dissimilar objects will have the same LSH value with a low probability. To date, several LSH families have been discovered for different distance metrics. In this paper, we consider the q-gram based Jaccard metric and use the MinHash[6] as the LSH function family. The family of MinHash functions is a $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive family for any d_1 and d_2, where $0 \leq d_1 < d_2 \leq 1$. Our method can be applied to other distance metrics such as Hamming distance and edit distance.

Intuitively, sending LSH values guarantees privacy as it does not involve any data. However, the LSH values are vulnerable against the frequency analysis attack by which the attacker tries to map the LSH values to strings based on their frequency. Formally, given n unique strings $\mathcal{S} = \{S_1, \ldots, S_n\}$ and $m \leq n$ LSH values of these strings, let f_i and l_i be the frequency of the string $S_i (1 \leq i \leq n)$ and the LSH value $L_i (1 \leq i \leq m)$ respectively. Then for each LSH value $L_i (1 \leq i \leq m)$, the attacker tries to find its matching candidates $\mathcal{S}' \subseteq \mathcal{S}$ s.t. (1) for all $S_i, S_j \in \mathcal{S}'$, $S_i \simeq S_j$, and (2) $\sum_{S_j \in \mathcal{S}'} f_j = l_i$. In other word, the attacker will try to find those similar strings that are mapped to the same LSH value based on their frequency. Then the attacker's probability that the LSH value $L_i (1 \leq i \leq m)$ is mapped to a specific string $S_j \in \mathcal{S}'$ is $prob(L_i \rightarrow S_j) = \frac{1}{|\mathcal{S}'|}$. For any LSH value that has few matching candidates (e.g., those strings have few similar others), it can be mapped to the correct string with high probability (possibly 100%).

3.1 Duplication Approach

To defend against the frequency analysis attack on the LSH-based approach, we define *α-privacy* to quantify the desired privacy guarantee:

DEFINITION 3.2. [α-privacy] Given a set of unique strings $\mathcal{S}$ $\{S_1, \ldots, S_n\}$ and their LSH values $\mathcal{L}$ $\{L_1, \ldots, L_m\}$, we say $\mathcal{L}$ is *α-private* if for each $L_i \in \mathcal{L}$, the probability that maps it to string S_j satisfies that $prob(L_i \rightarrow S_j) \leq \alpha$.

Intuitively, lower α provides higher privacy guarantee.

Next, we present our duplication-based approach that constructs α-private LSH values. It consists of two steps:

Step 1: Grouping. First, we construct the LSH values of all strings in $\mathcal{S}$. Second, we sort LSH values by their frequency in descending order. Third, starting from the most frequent LSH value, we repeat grouping $k = [\frac{1}{\alpha}]$ adjacent values together into one group (called an α-group), until all LSH values are grouped. The last group, if less than k in size, is merged with its previous group.

Step 2: Frequency homogenization. For each group, let l_{max} be the maximum frequency of its LSH values. Then for each LSH value L_i in the group, we add $l_{max} - l_i$ copies of L_i, where l_i is the frequency of L_i.

After the two steps, all LSH values in the same group have the same frequency, regardless their frequency in the original dataset.

3.1.1 Privacy Analysis of Duplication Approach

Frequency analysis attack. The duplication-based approach guarantees that there are always at least $[\frac{1}{\alpha}]$ LSH values that are of the same frequency. Therefore, even for the worst case that these values have no other similar strings, the probability of associating any LSH value with its original string based on their frequency is at most α. In other words, the duplication approach guarantees α-privacy.

Known-scheme attack. When the attacker knows the details of the duplication approach, he can apply the following two-step attack trying to find the mapping between LSH values and strings. First, the attacker tries to find the mapping between string groups and LSH groups. He can execute the grouping step of the duplication approach over the string values to get string groups; the string groups are expected to be similar to the LSH groups of the strings. Then for each string group, he will try to find out the corresponding LSH values based on their frequency. From the knowledge of the encoding algorithm, he knows that with high probability all LSH values of the same frequency belong to the same group. Furthermore, for any string S and its LSH value L, it must be true that $freq(S) \leq freq(L)$. Based on these information, he can map LSH groups to the string groups based on their frequency. Note that all LSH values in the same group have the same frequency, while the strings in the same group do not have to. Apparently, for any string groups of k unique and dissimilar strings, its LSH group must contain exactly k unique LSH values. Furthermore, since it is highly likely that a string group only contains dissimilar strings, the maximum frequency of the LSH group must be no larger than the maximum frequency of its string group. The attacker can use these knowledge to map LSH groups to string groups. Under the worst case, there is only one such mapping. Second, the attacker tries to find the mapping between specific strings and LSH values based on the group mapping result. Consider a mapping that consists of k strings and k LSH values. The attacker constructs all possible mappings between k LSH values and k strings, with each LSH value mapped to exactly one string. There are $k!$ such mappings. Among these mappings, $(k-1)!$ mappings map a specific LSH value $L_j (1 \leq j \leq k)$ to its correct string $S_i (1 \leq i \leq k)$ correctly. Therefore, the probability $prob(L_j \rightarrow S_i) = \frac{(k-1)!}{k!} = \frac{1}{k}$. Since $k = [\frac{1}{\alpha}]$, the duplication-approach is α-private against the known-scheme attack.

3.1.2 Complexity Discussion

The duplication approach requires $O(n)$ to group and add duplicates at the client side, where n is the number of unique strings in D. The complexity of data deduplication at the

server side is $O((|D|+p)^2)$, where $|D|$ is the total number of strings in D ($|D| \gg n$), and p is the total number of duplicated LSH values. In particular, the number of duplicated LSH values that are added to the string S_i is $r_i = (f_{max} - f_i)$. The total number p of duplicated LSH values added to D is $p = \sum_{i=1}^{n} r_i$. It can be large, especially for the datasets of skewed frequency distribution. This will hurt the performance of data deduplication in terms of both accuracy and time performance. Next, we discuss our *split-and-duplication* approach that can improve the duplication-based approach by reducing the number of added LSH copies, while still providing provable guarantee against both the frequency analysis attack and the known-scheme attack.

3.2 Split-and-duplication (SD) Approach

The main idea of the split-and-duplication (SD) approach is to add a *split* step between the grouping step and the frequency homogenization step of the duplication-based approach. The aim of the split step is to convert those LSH values of large frequency to several different ones of smaller frequency, so that the number of added LSH values by the frequency homogenization step can be reduced. Apparently the split copies of the same string cannot have the same LSH values. A naive approach is, for each LSH value L of high frequency, we make several copies of L, so that the frequency of these copies accumulate to the same frequency. Then for each copy of L, we randomly modify a number of its bits, so that all split copies of L will be different after modification. Though simple, this approach may lead to high amounts of accuracy lost due to the fact that the LSH values of similar strings may not share the same bucket anymore. Therefore, instead of making split copies of LSH values, we propose to make split copies of the strings, and construct LSH values of these string split copies. We ensure that for each similar string pair, at least one pair of their split copies share the same LSH value with high probability. Therefore, the server still can identify the similar strings from the received LSH values. Next, we present the details of the split step.

Our split step is based on the *permutation* mechanism. Formally, let S and S' be equal-size strings. We say S is a *permutation* of S' if there exists a bijection π such that for each i, $S[i] = \pi(S'[i])$. For instance, the string "cacb" is a permutation of the string "abac" as there exists the bijection $\pi(a) = c$, $\pi(b) = a$, and $\pi(c) = b$. We call π a permutation function. For any given string S of frequency f, the client defines g unique permutation functions, and constructs g unique split copies of S by applying the g permutation functions on S. Each permutation generates a split copy of frequency $[\frac{f}{g}]$. Continuing our example, consider the string "abac", and two permutation functions π_1 and π_2 such that $\pi_1(a) = c$, $\pi_1(b) = a$, and $\pi_1(c) = b$, as well as $\pi_2(a) = b$, $\pi_2(b) = c$, and $\pi_2(c) = a$, it has two split copies "cacb" and "bcba". We have the following theorem to show that permutation preserves Jaccard similarity.

LEMMA 3.1. Given any two strings S_1 and S_2, let π be a permutation function. Let S_1' and S_2' be the strings after applying π on S_1 and S_2 respectively. Then it must be true that $jaccard(S_1, S_2) = jaccard(S_1', S_2')$.

The proof of Lemma 3.1 can be found in Appendix. Following Lemma 3.1, we have the following theorem to show that for any two (dis)similar strings, their permutations using the same permutation function are always (dis)similar. Indeed, this applies to not only Jaccard similarity measurement but also other distance metrics such as Hamming distance and edit distance.

THEOREM 3.1. Given any two strings S_1 and S_2 s.t. $S_1 \simeq S_2$ ($S_1 \not\simeq S_2$ resp.). Let S_1' and S_2' be the strings after applying the permutation function π on S_1 and S_2 respectively. Then it must be true that $S_1' \simeq S_2'$ ($S_1' \not\simeq S_2'$ resp.).

Theorem 3.1 shows that the permutation-based split procedure will lead to neither *false negatives* (i.e., the split copies of two similar strings turn dissimilar) nor *false positives* (i.e., the split copies of two dissimilar strings become similar).

Next, we discuss how to decide the total number of split copies. Our goal is to ensure that the total amounts of duplicated LSH values by the SD approach is no larger than that of the duplication approach. First, we calculate the total number of the duplicated values that are to be added by the SD approach. For each group that consists of k strings $\{S_1, \ldots, S_k\}$, let f_i be the frequency of the string $S_i (1 \le i \le k)$. Assume that the string S_i is split into g_i copies. Apparently, the frequency of each split copy of S_i is $[\frac{f_i}{g_i}]$. Let $\overline{f_{max}}$ be the maximum frequency of the split copies in the group. Then for each string S_i, its number $\bar{r_i}$ of duplicated LSH values is $\bar{r_i} = g_i(\overline{f_{max}} - [\frac{f_i}{g_i}])$. To find out when the SD approach incurs smaller amounts of duplicate values than the duplication approach, we compare r_i and $\bar{r_i}$. Apparently $\bar{r_i} = g_i(\overline{f_{max}} - [\frac{f_i}{g_i}]) = g_i\overline{f_{max}} - f_i$. Compared with $r_i = f_{max} - f_i$, the SD approach wins the duplication approach if $f_{max} > g_i\overline{f_{max}}$. Note that the split copy of the string that has $\overline{f_{max}}$ may not be the same string of f_{max}. Therefore, for each string, we always pick g_i, the number of split copies, in the way that ensures $f_{max} > g_i\overline{f_{max}}$.

3.2.1 Privacy Analysis of SD Approach

Frequency analysis attack. The SD approach guarantees that for each LSH value, there are always at least $[\frac{1}{\alpha}] - 1$ other LSH values that are of the same frequency. Thus, it guarantees α-privacy against the frequency analysis attack.

Known-scheme attack. If the attacker knows the details of the SD approach, he can launch the following two-step attack trying to decode the received LSH values. By the first step, the attacker tries to find mapping between string and LSH groups based on the frequency of strings and LSH values. From the analysis of the SD algorithm, he knows that all LSH values of the same frequency belong to the same group with high probability. Furthermore, for any string S and its split copy S', $freq(S) \ge freq(S')$. Apparently, for any string groups of k unique strings, its LSH groups must contain at least k unique LSH values. Furthermore, the maximum frequency of the LSH group must be no larger than the maximum frequency of its string group. The attacker can use these two facts to map LSH groups with string groups. In the worst case, there is only one such mapping. Then by the second step, the attacker tries to find the mapping between LSH values and strings. Consider a mapping that consists of k strings and $\ell \ge k$ LSH values, in which LSH values are of the same frequency while the strings may have different frequency. The attacker tries to construct all possible mappings of ℓ LSH values to k strings. This is equivalent to the problem of distributing ℓ distinguishable balls into k distinguishable boxes where $\ell > k$ and no box is empty. The number of such mapping equals: $M(\ell, k) = \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^\ell$. Among these mappings, there are $M(\ell-1, k) + M(\ell-1, k-1)$ correct

mappings between a LSH value $L_j (1 \leq j \leq \ell)$ and its string $S_i (1 \leq i \leq k)$. Therefore, the probability of finding the correct mapping $L_j \rightarrow S_i$ is

$$prob(L_j \rightarrow S_i) = \frac{M(\ell-1,k) + M(\ell-1,k-1)}{M(\ell,k)} = \frac{1}{k}.$$

Since $k \geq \frac{1}{\alpha}$, the SD approach can provide α-privacy against the known-scheme attack. Interestingly, varying the number of split copies will not impact the privacy guarantee.

3.2.2 Complexity Discussion

The SD approach requires $O(ns)$ to construct all split copies at the client side, where n is the number of unique strings in D, and s is the average number of split copies of all strings. The complexity of the data deduplication at the server side is $O((|D|+p)^2)$, where $|D|$ is the total number of strings in D, and p is the total number of duplicates. Note that $|D| \gg n$. Thus the computational effort at the client side is much cheaper than that at the server side.

4. EMBEDDING & HOMOPHONIC SUBSTITUTION (EHS) APPROACH

In this section, we present a different approach called Embedding & Homophonic Substitution (EHS) that constructs the encoding of the input data without adding any additional data. In a nutshell, EHS has two steps: (1) conversion of categorical data to Euclidean space, and (2) 1-to-many homophonic substitution of Euclidean points. The EHS approach can be easily adapted to numerical data for which the conversion step is not necessary, while the substitution step will be applied on the original data.

For the first step, many mapping techniques (e.g., $FastMap$ [9] and $StringMap$ [16]) can be used to map strings into a multidimensional Euclidean space. The mapping functions guarantee that the similar strings are mapped to close Euclidean points. In this paper, we use $FastMap$ [9] for string conversion. By $FastMap$, the string similarity threshold δ will be adjusted to a new similarity threshold δ_E for Euclidean points. $FastMap$ is not reversible [9], thus the original strings cannot be reconstructed from the embedded Euclidean points. The complexity of the embedding step is $O(d^2|D|)$, where d is the number of dimensions of the Euclidean points, and $|D|$ is the number of strings in D. We omit the details of the $FastMap$ algorithm for space issue.

$FastMap$ embedding is one-to-one, simply mapping strings to Euclidean points fails to defend against the frequency analysis attack. Therefore, the second step of EHS is to apply a homophonic substitution encoding scheme on the Euclidean points to defend against the frequency analysis attack. Homophonic substitution schemes are one-to-many, meaning that multiple encoded symbols can map to one plaintext symbol. Next, we will first discuss our basic homophonic substitution encoding scheme and its weakness against both the frequency analysis and the known-scheme attacks (Section 4.1). We then present our grouping-based homophonic substitution (GHS) scheme that can defend against these attacks with provable guarantee (Section 4.2).

4.1 Basic Approach

By the basic approach, any Euclidean point P of frequency f is transformed to f unique points $E_1, \ldots, E_f$ in the same Euclidean space, each point of frequency 1. Therefore, the frequency distribution of the Euclidean points is always uniform, regardless of the frequency distribution of the original data. The substitution scheme is one-to-many as one string maps to multiple Euclidean points. To preserve similarity, for each point P in the d-dimension Euclidean space, we construct a d-sphere H^r of radius r centered at P, where r is a user-specified parameter. Let f be the frequency of P. Then we pick f points from H^r in a uniformly random manner, with each point of frequency 1. These f points are the encoded versions of P. The distance between the constructed Euclidean points satisfies the following theorem.

THEOREM 4.1. Given any two original Euclidean points P_1 and P_2, let $\{E_1^1, \ldots, E_1^k\}$ and $\{E_2^1, \ldots, E_2^t\}$ be the Euclidean points of P_1 and P_2 constructed by the above procedure (t and k are not necessarily the same). Then for any pair of Euclidean points E_1^i and E_2^j ($i \in [1,k], j \in [1,t]$), $dist(E_1^i, E_2^j) \leq dist(P_1, P_2) + r_1 + r_2$, where $dist()$ is the Euclidean distance function, and r_1, r_2 are the length of the radius of the corresponding d-spheres of P_1 and P_2.

The proof of Theorem 4.1 is included in Appendix. Based on Theorem 4.1, we adjust the similarity threshold of the encoded Euclidean points accordingly. We say two encoded values E_i and E_j are similar if $dist(E_i, E_j) \leq \delta_E + r_i + r_j$, where δ_E is the distance threshold after embedding, r_i and r_j are the length of the radius of spheres of P_i and P_j, the corresponding original Euclidean point of E_i and E_j.

4.1.1 Privacy Analysis

Frequency analysis attack. Given n strings, each of frequency f_i, and m encoded Euclidean points, each of frequency f ($f = 1$ for the basic approach), we define $sf = \sum_{i=1}^{n} f_i$. With further computation, the probability $prob_F$ that the encoded point E_j is mapped to its string S_i by the frequency analysis attack is

$$prob_F(E_j \rightarrow S_i) = \frac{\binom{f_i}{f}}{\binom{sf}{f}}. \quad (1)$$

As an example, consider two strings S_1 and S_2 of frequency 10000 and 10. These two strings are encoded as 10000 and 10 Euclidean points respectively by the basic approach. Then for any encoded Euclidean point E_j of S_1, $prob_F(E_j \rightarrow S_1) = \binom{10000}{1} / \binom{10010}{1} = \frac{10000}{10010} \approx 1$.

Known-scheme attack. If the attacker possesses the details of the basic approach, he knows that all Euclidean points of the same string must fall in a circle of a small radius in the Euclidean space. Given n unique strings and $m \geq n$ Euclidean points, he can apply the following 2-step attack to find the mapping between the Euclidean points and the strings: By the first step, it uses a clustering algorithm (e.g., k-means clustering) to group m Euclidean points to n groups based on their closeness, so that the groups are of low intra-group distances and high inter-group distances. By the second step, it maps n clusters of Euclidean points to n unique strings based on their similarities. In particular, the attacker maps the n cluster centroids to n strings in the way that for any three centroids c_i, c_j and c_k, their corresponding strings S_i, S_j and S_k satisfy that if $dist(c_i, c_j) \leq dist(c_i, c_k)$, then $jaccard(S_i, S_j) \geq jaccard(S_i, S_k)$. After the two-step attack, some Euclidean points of the same string may be covered by multiple clusters (i.e., mapped to multiple strings). Given n unique strings and $m \geq n$ Euclidean points by the basic approach, the probability $prob_S$ that the encoded point E_j is mapped to its corresponding string S_i by the

known-scheme attack is

$$prob_S(E_j \rightarrow S_i) = \frac{1}{t}, \quad (2)$$

where t is the number of clusters that include E_j. If $t = 1$, the mapping $E_j \rightarrow S_i$ is broken with 100% certainty.

Now we are ready to define the privacy model that can defend against both the frequency analysis attack and the known-scheme attack.

DEFINITION 4.1. [(α_1, α_2)-privacy] Given n strings $\mathcal{S}$ $\{S_1, \ldots, S_n\}$ and m encoded Euclidean points $\mathcal{E}$ $\{E_1, \ldots, E_m\}$, we say $\mathcal{E}$ satisfies (α_1, α_2)-*privacy* if for any Euclidean point $E_i \in \mathcal{E}$ (whose corresponding string is $S_j \in \mathcal{S}$), $prob_F(E_j \rightarrow S_i) \leq \alpha_1$, and $prob_S(P_i \rightarrow S_i) \leq \alpha_2$, where $prob_F$ and $prob_S$ are measured by Eqn. 1 and Eqn. 2 respectively.

As we have shown, the basic approach may fail to meet the (α_1, α_2)-privacy requirement.

4.2 Grouping-based Homophonic Substitution (GHS) Scheme

To address the privacy weakness of the basic approach, we design the grouping-based homophonic substitution encoding scheme (GHS) that satisfies (α_1, α_2)-privacy. The GHS takes n Euclidean points that are transformed from the original n strings and outputs m ($m \geq n$) transformed Euclidean points, each of frequency 1. It satisfies that $\sum_{i=1}^{n} f_i = m$, where $\sum_{i=1}^{n} f_i$ is the sum of the frequency of all Euclidean points. Before we explain the details of GHS method, we first define (α_1, α_2)-*bounded clusters*.

DEFINITION 4.2. [(α_1, α_2)-bounded clusters] For any given cluster C and two user-specified thresholds α_1, α_2, C is (α_1, α_2)-*bounded* if it satisfies: (1) for each point $P \in C$, $\frac{f_P}{\sum_{X \in C} f_X} \leq \alpha_1$, where f_P (f_X, resp.) is the frequency of point P (point X, resp); and (2) C contains at least $[\frac{1}{\alpha_2}]$ points.

The GHS procedure consists of 2 steps. First, the Euclidean points are grouped into (α_1, α_2)-bounded clusters. For each cluster C, a d-sphere H is constructed to cover all points of C, where d is the dimension of the Euclidean space. Second, for each sphere H that covers the points $\{P_1, \cdots, P_k\}$, $f_1 + \cdots + f_k$ points in total are constructed within H in a uniform random fashion, where f_i ($1 \leq i \leq l$) is the frequency of the point P_i. Each constructed point is of frequency 1. Theorem 4.2 below shows how to update the similarity threshold for the Euclidean points constructed by the GHS approach.

THEOREM 4.2. Given any two original Euclidean points P_1 and P_2 that are similar according to the threshold δ', let $\{E_1^1, \ldots, E_1^k\}$ and $\{E_2^1, \ldots, E_2^t\}$ be the Euclidean points of P_1 and P_2 constructed by the above procedure (t and k are not necessarily the same), and r_1 and r_2 be the top-2 maximum radius length of all spheres. Then for any pair of Euclidean points E_1^i and E_2^j ($i \in [1, k], j \in [1, t]$), $dist(E_1^i, E_2^j) \leq \delta' + 2r_1 + 2r_2$, where $dist()$ is the Euclidean distance function.

The proof of Theorem 4.2 is in Appendix. Note that different from Theorem 4.1 that considers the distance between the centers of spheres, Theorem 4.2 considers the distance between any two points within the spheres. Following Theorem 4.2, the new similarity threshold for the points constructed by GHS approach is $\delta_{new} = \delta_E + 2r_1 + 2r_2$. Next, we discuss how to construct (α_1, α_2)-bounded clusters.

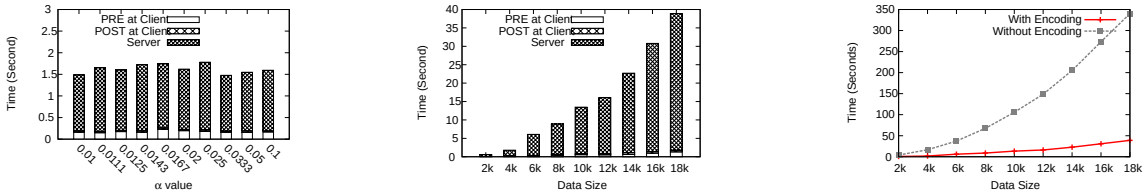
Our algorithm consists of the following steps. First, each point is initialized as a cluster. Second, for each cluster C_i that is not (α_1, α_2)-bounded, we look for other cluster(s) to merge with C_i. In particular, for each cluster candidate C_j, we compute the radius length of the sphere by merging C_j with C_i, and pick the candidate that leads to the smallest radius length. We design an efficient algorithm that computes the *approximate* radius length in $O(1)$ time. In particular, given two clusters C_i, C_j of centers c_i and c_i respectively, let r_i and r_j be the radius length of the spheres of C_i and C_j, and $c_i[x]$ be the value of the x-th dimension of the vector c_i. The new center c_{new} of the sphere that covers all points in $C_i \cup C_j$ is computed as $c_{new}[x] = \frac{c_i[x]s_i + c_j[x]s_j}{s_i + s_j}, \forall x \in [1, d]$, where s_i and s_j are the number of points of C_i and C_j, and d is the dimension of the Euclidean space. Then the approximate radius length of the sphere that covers all points in $C_i \cup C_j$ is computed as $r_{new} = max(dist(c_i, c_{new}) + r_i, dist(c_j, c_{new}) + r_j)$. We look for the cluster C_j whose merge with C_i will lead to the smallest r_{new}. After that, we check whether $C_i \cup C_j$ is a (α_1, α_2)-bounded cluster. If not, we repeat the merging procedure, until all clusters are (α_1, α_2)-bounded.

4.2.1 Privacy analysis

The GHS approach provides (α_1, α_2)-privacy against the frequency analysis attack, as each point belongs to a (α_1, α_2)-cluster. Next, we discuss the robustness against the known-scheme attack. If the attacker knows the details of the GHS method, he can group the received Euclidean points into clusters of high intra-cluster similarity. Next, he tries to map Euclidean points to the strings based on their frequency. In particular, a cluster of f points is highly likely to be mapped to the strings $\{S_1, \ldots, S_t\}$ if $\sum_{i=1}^{t} f_i = f$. It is possible that the attacker can find a unique mapping between the clusters and the strings. After that, for each cluster C_i that consists of ℓ Euclidean points $\{E_1, \ldots, E_\ell\}$ (each of frequency $f = 1$), assume C_i is mapped to $k \leq \ell$ strings $\{S_1, \ldots, S_k\}$. He can apply the frequency analysis attack (Section 4.1.1) on C_i. Since each cluster is (α_1, α_2)-bounded, the probability $prob_F$ that the encoded point E_j is mapped to its corresponding string S_i by the frequency analysis attack equals to $prob_F(E_j \rightarrow S_i) = \binom{f_i}{f} / \binom{mf}{f} = \frac{f_i}{mf} \leq \alpha_1$. The attacker also computes the probability of mapping each Euclidean point to a specific string by trying all possibilities. The reasoning of the attack probability is similar to the attack as described in Section 3.2.1, where $M(\ell, k) = \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k - i)^\ell$, and $prob_S(E_j \rightarrow S_i) = \frac{M(\ell-1,k)+M(\ell-1,k-1)}{M(\ell,k)} = \frac{1}{k}$. Since $k \geq [\frac{1}{\alpha_2}]$, the probability $prob_S \leq \alpha_2$. Therefore, the GHS approach can provide (α_1, α_2)-privacy guarantee.

4.2.2 Complexity analysis

The complexity of constructing (α_1, α_2)-bounded clusters is $O(ny)$, where n is the number of Euclidean points, and y is the number of required iterations. Our empirical study shows that y is normally a small portion of n (at most half of n). The complexity of constructing the spheres for each cluster is $O(1)$, and the complexity of constructing all transformed Euclidean points is $O(n)$. Thus the total complexity of GHS is $O(ny)$. The complexity of data deduplication at the server side is $O(|D|^2)$, where $|D|$ is the total number of strings of D. Since identical string values are always mapped to the same Euclidean point, $|D| \gg n$. Therefore, the computations at the client side are much cheaper than that at the server side.



(a) Various α values ($FEMALE$ dataset) (b) Various data sizes ($LAST$ dataset) (c) Time overhead by encoding ($LAST$ dataset)

Figure 1: Time Performance of $LSHB$ Approach

5. POST-PROCESSING

After the server returns the near-duplicates, the client maps the received results to strings, then eliminates the false positives by measuring the similarity of all near-duplicate pairs. Assume that the similarity of each near-duplicate pair can be measured in constant time. Then the complexity of the post-processing is $O(|R^S|)$, where $|R^S|$ is the number of near-duplicates that are returned by the server. Compared with the complexity of data deduplication of the original dataset D (quadratic to $|D|$), the post-processing is much cheaper than running the data deduplication locally (note that $|R^S| \ll |D|$).

6. EXPERIMENTS

6.1 Setup

Experiment environment. We implemented both of our $LSHB$ and EHS approaches in Java. All experiments were executed on a PC with a 2.4GHz Intel Core i5 CPU and 8GB memory running Linux.

Datasets. We use two real datasets[1] from US Census Bureau in our experiments: (1) the *female first name* dataset ($FEMALE$ for short) that contains 4275 unique female first names and their frequencies, and (2) the *last name* dataset ($LAST$ for short) that contains 88799 last names and their frequencies. We also use subsets of $LAST$ dataset of various sizes for scalability measurement. To study the impact of pairwise distance distribution to the performance, we also prepared two types of datasets: (1) the *thick* dataset in which a large potion of strings are similar to each other; and (2) the *sparse* dataset in which most strings are dissimilar.

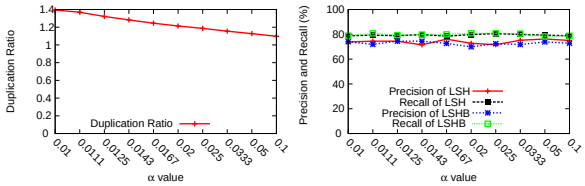
Parameters. We use $q = 2$ for q-gram setting and set the jaccard similarity threshold to be 0.4 for both $FEMALE$ and $LAST$ datasets. For $LSHB$ approach, we apply 100 random permutations to generate the MinHash signatures.

6.2 Performance of $LSHB$ Approach

6.2.1 Time performance

First, we measure the impact of various α values (for α-privacy) to the time performance of $LSHB$ approach (Figure 1 (a)). We measure preparation time at the client side, data deduplication time at the server side, and post-processing time at the client side. First, the preparation time at the client side is stable. This is because the complexity of encoding is decided by the number of unique strings, which is not affected by α value. The post-processing time at client side is also stable, because different α values do not change the number of similar string pairs. Second, the time performance at the server side is also stable. The reason is the complexity at the server side only relies on the number of unique LSH values (and thus the number of unique strings), thus changing α values does not affect the running time at server side. Third, the computational efforts at the server side dominates the whole data deduplication process (including both at the client and the server side). This observation consolidates our claim that the $LSHB$ approach suits the outsourcing paradigm very well.

[1]available at `http://goo.gl/cjbN`.



(a) Amounts of added copies (b) Precision & recall

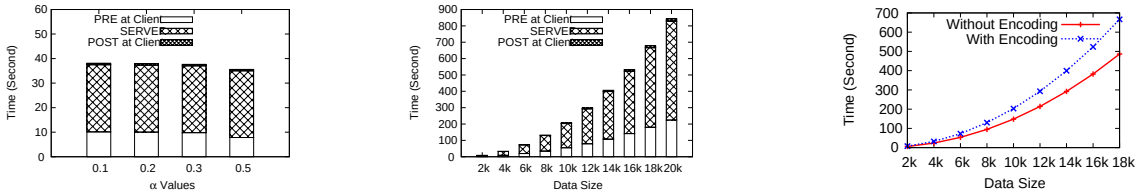
Figure 2: Impact of $LSHB$ Encoding to Accuracy

Second, we measure the scalability of $LSHB$ approach on datasets of various sizes (Figure 1 (b)). We observe that the preparation time at the client side increases slowly with the growth of the data size, while the post-processing time at the client side and the time performance at the server side increases sharply. This is because the complexity of preparing LSH values at the client side is linear to the data size, while the complexity of deduplication and data post-processing is quadratic to data size. Second, compared to the time performance at the server side, the post-processing time at the client side is very small, as the server has filtered many dissimilar string pairs. We also observe that the time performance of preparation at the client side is much cheaper than that of the server side. Indeed, the LSH construction time at client side never exceeds 2 seconds. We also observe that the difference between the performance at the client and the server sides becomes more significant with the growth of data size. This is because the complexity of preparation at the client side is linear to the data size, while the complexity at the server side is quadratic to the data size. This proves the advantage of the outsourcing paradigm that handles large datasets most of the time.

Third, we measure the impact of $LSHB$ encoding to the time performance of data deduplication. We compare the time of the $LSHB$ approach (including the time at the client and the server side) with the time of the data deduplication on the original data (Figure 1 (c)). We observe that the time performance of both scenarios increases with the growth of data size. However, the time performance of the $LSHB$ encoding is always cheaper than that of the original data. This speed-up is mainly due to the use of LSH values. This shows that although the privacy-preserving methods may bring computational overhead in general, picking a right encoding method (e.g., LSHs) may enable faster data deduplication than the brute force approach of measuring the pairwise similarity of all string pairs.

6.2.2 Impact to Accuracy of Data Deduplication

Storage overhead. We measure the amounts of the added LSH copies by using *duplication ratio*. We define



(a) Various α values ($FEMALE$ dataset) (b) Various data sizes ($LAST$ dataset) (c) Time overhead by encoding ($LAST$ dataset)

Figure 3: Time Performance of EHS Approach

the duplication ratio as $\frac{|D'|}{|D|}$, where $|D|$ and $|D'|$ are the number of strings in original D and after applying $LSHB$ respectively. In Figure 2 (a), the ratio increases with the decrease of α (i.e., higher privacy requirement). The reason is that smaller α requires larger groups, and thus more inserted copies to make their frequency homogenized. This is the price we need to pay for higher privacy.

Precision/recall We compare the precision/recall of applying our $LSHB$ approach with the precision/recall of directly applying LSH hashing over the original data (Figure 2 (b)). In all the experiments, the precision varies around 75%, and the recall is around 80%. Our $LSHB$ approach is able to provide comparable accuracy to LSH. This convinces us the good utility of the $LSHB$ approach. We note that after post-processing, we can ensure 100% precision.

6.3 Performance of EHS Approach

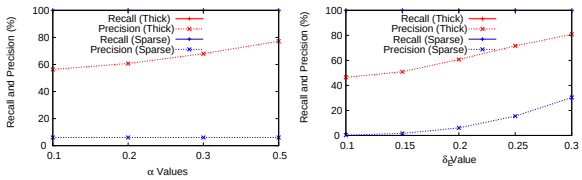
6.3.1 Time performance

First, we measure the time performance of EHS encodings for various α_1 and α_2 settings of (α_1, α_2)-privacy (Figure 3 (a)). We use the similarity threshold $\delta_E = 0.2$ and various α values ($\alpha = \alpha_1 = \alpha_2$). We measure: (1) the preparation time at the client side, (2) the data deduplication time at the server side, and (3) the post-processing time at the client side. First, we observe that the preparation time decreases when α increases. The reason is that larger α value requires smaller clusters and points of lower frequency, leading to less merge during the (α_1, α_2)-bounded cluster. Second, the data deduplication time is stable for all α values, because the complexity at the server side is decided by the dataset size, which is unchanged for these settings. Third, the post-processing time decreases when α increases. That is when α value is larger, the clusters get smaller, which leads to more tightened similarity threshold and thus fewer false positives. For all experiments, the post-processing time is always less than 0.6 seconds. Similar to $LSHB$ approach, we observe that the server's computation efforts are much higher than that of the client.

Second, we measure the scalability of EHS approach for various data sizes. As shown in Figure 3 (b), the preparation time increases with the growth of the data size. That is because larger dataset requires more time to construct (α_1, α_2)-bounded clusters. On the other hand, the post-processing time increases because the size of R^S increases with the size of data. We also compare the time performance at both the client and the server side. We observe that the client requires much less time than the server. When the size of the dataset increases, the running time at the server side grows faster than the client side. This observation shows that our EHS approach suits the outsourcing paradigm well.

Third, we compare the time performance of data deduplication by our EHS encoding with that on the original data (Figure 3 (c)). For EHS encoding, we add up the preparation time, the post-processing time, and the data deduplication time. We observe that the processing time of both cases increases with the size of the data. However, applying EHS encoding only adds a small potion of time overhead compared with the performance of the original data.

6.3.2 Impact to Accuracy of Data Deduplication



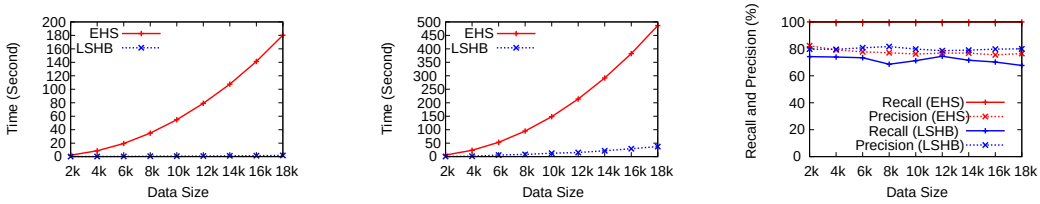
(a) Various α values ($\delta_E = 0.2$)(b) Various δ_E values ($\alpha = 0.2$)

Figure 4: Impact of EHS Encoding to Accuracy of Deduplication (*FEMALE* dataset, $d = 10$)

We measure the precision and recall of our EHS approach for various α values ($\alpha = \alpha_1 = \alpha_2$) and various δ_E values (Figure 4). First, we observe that recall is always 100%. In other words, all the near-duplicates in the original dataset are returned. Second, from Figure 4 (a), we observe that precision increases with α value. This is because larger α value leads to smaller clusters, which yield more tightened δ_{new}. One interesting observation is, the precision on the *sparse* dataset increases much slower than that of the *thick* dataset. That is because the δ_{new} value of the *sparse* dataset is large; thus the change of δ_{new} (from 1.00 to 0.89) makes less impact than the change of δ_{new} on the *thick* dataset (from 0.23 to 0.21). As shown in Figure 4 (b), the precision raises with δ_E, because large δ_E results in more similar pairs. Thus, the ratio of false positive decreases. In all experiments, we observe that the precision of the *thick* dataset (at least 0.4) is much better than the *sparse* dataset. To understand why, we measured the average pair-wise distances of both datasets. It turned out that the average pair-wise distance of the *thick* and the *sparse* datasets are 0.27 and 0.31 respectively. We use 0.2 as the value of δ_E, and measure the new similarity threshold δ_{new}. For the *thick* dataset, δ_{new} is less than 0.23, which is less than 15% change of the similarity threshold. While for the *sparse* dataset, δ_{new} is changed to be between 0.9 and 1, which is too loose compared with the original threshold and thus concludes almost all pairs as similar. Therefore, the (α_1, α_2)-bounded clustering method introduces fewer false positives on the *thick* dataset than the *sparse* dataset.

6.4 $LSHB$ Versus EHS

First, we compare the time performance of encoding at client side for both approaches (Figure 5 (a)). We observe that the $LSHB$ approach is much faster than the EHS approach. This is because constructing $LSHs$ is much faster than the $FastMap$ embedding and (α_1, α_2)-bounded clustering. Second, we compare the time performance of data



(a) Time performance at client side (b) Time performance at server side (c) Precision and recall

Figure 5: Comparison of $LSHB$ Approach and EHS Approach ($LAST$ dataset, $r = 0.4$)

deduplication at the server side. The result (Figure 5 (b)) shows that although $LSHB$ adds additional values to the outsourced dataset, it is again faster than EHS, due to the fact that dealing with LSH values is much faster than the computation of Euclidean distances. Third, we compare the precision and recall of the two approaches (Figure 5 (c)). We observe that $LSHB$ yields comparable precision with EHS but EHS yields much higher recall (100%). In summary, there exists the trade-off between the time performance of the privacy-preserving encoding methods and the accuracy of data deduplication on encoded data. The use of LSH values enables faster time performance at both the client and the server sides, but it introduces lower recall rate, while EHS guarantees 100% recall.

7. RELATED WORK

Data deduplication is highly related to the record linkage problem. Privacy-preserving record linkage has been paid much attention recently. The existing approaches can be classified based on the privacy-preserving techniques they use. We classify these techniques into the following types: (1) encoding and embedding, (2) differential privacy, (3) secure multi-party computation (SMC).

Substitution encoding and embedding. Churches et al. [5] propose a three-party protocol based on hash values of q-grams. In particular, for each string, its power set (i.e. the subsets of the original bigram set) is constructed. Each subset of the power set is hashed using a common secret key shared among database owner A and B. Party A and B form tuples containing the hash values, the number of q-grams in the hashed subset and the total number of q-grams and an encryption of the identifiers to a third party C. Party C computes the similarity based on the hash values. To prevent the frequency analysis attack, Churches et al. [5] propose to: (1) use an additional trusted party, thereby extending the number of parties to four; and (2) recommend to use chaffing and winnowing [20] to insert dummy records to obfuscate the frequency information in the hash values. A trusted party may not exist in practice. Furthermore, besides the substantial increase of computational and communication costs, inserting dummy records is still prone to frequency attacks on the hashes of the q-gram subsets with just one q-gram [24]. Schnell et al. [22] and Durham et al. [7] 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. [22] observed that the BFs are still open to the frequency attack. [18] formalized the frequency attack on the Bloom filters as a constraint satisfaction problem, and pointed out that the probability of mapping BF values to original strings is high even when only the frequency of samples is used for the attack. To fix this problem, [7] propose composite BFs by which the attribute-level bloom filter values are to be combined into a single bloom filter for the entire record. Compared with our work, Kuzu et al. [18] consider the attack that tries to map BF bits back to the attributes from which they were constructed by utilizing the frequency of bits, while we consider the attack that maps the encoded values to their original strings based on their frequency. Storer et al. [23] 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. This one-to-one encryption cannot defend against the frequency analysis attack.

Scannapieco et al. [21] focus on the three-party scenario and use embedding techniques to transform original strings into a Euclidean space by using the $SparseMap$ method [11]. Since the mapping between each distinct string and each distinct Euclidean point is still one-to-one, the embedded Euclidean points can be easily mapped to their strings easily based on their frequency.

Differential privacy. Bonomi et al. [4] 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. However, as the embedding is still one-to-one, it cannot defend against the frequency-based attack. Inan et al. [12, 13, 14] propose a three-party protocol that first runs the *blocking* step by utilizing anonymized data sets to accurately match or mismatch a large portion of record pairs. Then the blocking step is followed by the SMC step, where unlabeled record pairs are labeled using cryptographic techniques. However, the SMC step still involves high computational cost [4].

Secure multi-party (SMC) computations. Most of the aforementioned protocols involve 3 parties. Atallah et al. [2] design a SMC protocol based on homomorphic encryption for record linkage based on edit distance similarity. Ravikumar et al. [19] propose a SMC protocol for two-party record matching using TF-IDF distance and Euclidean distance. Since we consider a client-server framework, we aim to design privacy-preserving encoding methods that are more efficient than SMC computations.

8. CONCLUSION

In this paper, we designed two data encoding methods for privacy-preserving data-deduplication-as-a-service ($DDaS$) paradigm. Our two methods can defend against the frequency analysis attack and the known-scheme attack with provable guarantee, while enabling to find near-duplicates from the encoded data. For the future work, we are interested in extending our work to other similarity measurement metrics, e.g., edit distance. We also plan to investigate other security issues related to $DDaS$, e.g., efficient result integrity verification of the returned near-duplicates.

9. REFERENCES

[1] Openrefine. http://openrefine.org/.

[2] M. J Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proceedings of ACM workshop on Privacy in the electronic society*, 2003.

[3] P. Bohannon, W. F. Fan, F. Geerts, X.B. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, 2007.

[4] L.Bonomi, L. Xiong, R. Chen, and B. Fung. Frequent grams based embedding for privacy preserving record linkage. In *CIKM*, 2012.

[5] T. Churches and P. Christen. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making*, 4(1):9, 2004.

[6] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D Ullman, and C. Yang. Finding interesting associations without support pruning. *TKDE*, 13(1):64–78, 2001.

[7] E. Ashley Durham, M. Kantarcioglu, Y. Xue, C. Toth, M. Kuzu, and Bradley Malin. Composite bloom filters for secure record linkage. *TKDE*, 1(1), 2013.

[8] W. W Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. *The Data Warehousing Institute*, 2002.

[9] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *SIGMOD*, 1995.

[10] L. Gravano, P. G Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[11] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transaction of Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[12] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco. A hybrid approach to private record linkage. In *ICDE*, 2008.

[13] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *EDBT*, 2010.

[14] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. A hybrid approach to private record matching. *IEEE Transactions on Dependable and Secure Computing*, 9(5):684–698, 2012.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the symposium on Theory of computing*, 1998.

[16] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, 2003.

[17] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006.

[18] M. Kuzu, M. Kantarcioglu, E. Durham, and B. Malin. A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In *Privacy Enhancing Technologies*, 2011.

[19] P. Ravikumar, W. W Cohen, and S. E Fienberg. A secure protocol for computing string distance metrics. In *Proceedings of the Workshop on Privacy and Security Aspects of Data Mining at the International Conference on Data Mining*, 2004.

[20] R. L Rivest. Chaffing and winnowing: Confidentiality without encryption. *CryptoBytes (RSA laboratories)*, 4(1):12–17, 1998.

[21] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In *SIGMOD*, 2007.

[22] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):41, 2009.

[23] M. W Storer, K. Greenan, D. DE Long, and E. L Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, 2008.

[24] S. Trepetin. Privacy-preserving string comparisons in record linkage systems: a review. *Information Security Journal: A Global Perspective*, 17(5-6):253–266, 2008.

APPENDIX

A. PROOF OF LEMMA 3.1

PROOF. Let Σ be the alphabet. Since π is a bijection, for any $y_1, y_2 \in \Sigma$, $\pi(y_1) = \pi(y_2)$ if and only if $y_1 = y_2$. Therefore, for any q-gram $Y \in G(S_1, q) \cap G(S_2, q)$, it must be true that $\pi(Y) \in G(S_1', q) \cap G(S_2', q)$. Also for any q-gram $Y \in G(S_1', q) \cap G(S_2', q)$, it must be true that $\pi^{-1}(Y) \in G(S_1, q) \cap G(S_2, q)$. Therefore, $|G(S_1, q) \cap G(S_2, q)| = |G(S_1', q) \cap G(S_2', q)|$. Similarly, we can prove that $|G(S_1, q) \cup G(S_2, q)| = |G(S_1', q) \cup G(S_2', q)|$. Therefore, $jaccard(S_1', S_2') = jaccard(S_1, S_2)$. ∎

B. PROOF OF THEOREM 4.1

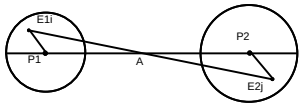


Figure 6: Encoding of Euclidean Points

PROOF. Given any two points P_1 and P_2, we pick a point A between these two points. Figure 6 illustrates the positions of P_1, P_2 and A. Apparently, it must be true that $dist(P_1, E_1^i) \leq r_1 (1 \leq i \leq k)$. Similarly, $dist(P_2, E_2^j) \leq r_2$ $(1 \leq j \leq t)$. According to triangle inequality, $dist(E_1^i, A) \leq dist(P_1, E_1^i) + dist(P_1, A) \leq r_1 + dist(P_1, A)$, and $dist(E_2^j, A) \leq dist(P_2, E_2^j) + dist(P_2, A) \leq r_2 + dist(P_2, A)$. It's easy to infer that $dist(E_1^i, E_2^j) = dist(E_1^i, A) + dist(E_2^j, A) \leq r_1 + r_2 + dist(P_1, A) + dist(P_2, A) = r_1 + r_2 + dist(P_1, P_2)$ ∎

C. PROOF OF THEOREM 4.2

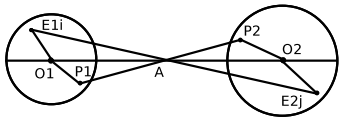


Figure 7: Similarity Preservation of Euclidean Points

PROOF. Given any two Euclidean points P_1 and P_2, assume that they are in the different clusters whose centers are O_1 and O_2. Let A be the intersection point of the line (P_1, P_2) and (O_1, O_2). By triangle inequality, $dist(P_1, A) + dist(O_1, P_1) \geq dist(O_1, A)$, and $dist(P_2, A) + dist(O_2, P_2) \geq dist(O_2, A)$. Combining them we have: $dist(P_1, P_2) = dist(P_1, A) + dist(P_2, A) \geq dist(O_1, A) + dist(O_2, A) - dist(O_1, P_1) - dist(O_2, P_2) \geq dist(O_1, O_2) - r_1 - r_2$ Since $dist(O_1, P_1) \geq r_1$ and $dist(O_2, P_2) \geq r_2$, $dist(O_1, O_2) \leq dist(P_1, P_2) + r_1 + r_2$. On the other hand, for any points E_1^i constructed for P_1 and any points E_2^j constructed for P_2, we have $dist(E_1^i, A) \leq dist(O_1, A) + dist(O_1, E_1^i)$, and $dist(E_2^j, A) \leq dist(O_2, A) + dist(O_2, E_2^j)$. Combining them we have: $dist(E_1^i, E_2^j) \leq dist(O_1, O_2) + dist(O_1, E_1^i) + dist(O_1, E_2^j)$. Given the fact that $dist(O_1, E_1^i) \leq r_1$, $dist(O_2, E_2^j) \leq r_2$ $(1 \leq j \leq t)$, and $dist(P_1, P_2) \leq \delta_E$, we have: $dist(E_1^i, E_2^j) \leq dist(O_1, O_2) + dist(O_1, E_1^i) + dist(O_2, E_2^j) \leq dist(P_1, P_2) + r_1 + r_2 + r_1 + r_2 \leq \delta_E + 2r_1 + 2r_2$. ∎