Undefined 0 (2016) 1
IOS Press



# Private Record Linkage

*An analysis of the accuracy, efficiency, and security of selected techniques for name matching*

Pawel Grzebala and Michelle Cheatham *
*Wright State University, 3640 Colonel Glenn Hwy., Dayton, OH 45435, USA*
*E-mail: {grzebala.2\michelle.cheatham}@wright.edu*

**Abstract.** The rise of Big Data Analytics has shown the utility of analyzing all aspects of a problem by bringing together disparate data sets. Efficient and accurate private record linkage algorithms are necessary to achieve this. However, records are often linked based on personally identifiable information, and protecting the privacy of individuals is critical. This paper contributes to this field by studying an important component of the private record linkage problem: linking based on names while keeping those names encrypted, both on disk and in memory. We explore the applicability, accuracy, speed and security of three different primary approaches to this problem (along with several variations) and compare the results to common name-matching metrics on unprotected data. While these approaches are not new, this paper provides a thorough analysis on a range of datasets containing systematically introduced flaws common to name-based data entry, such as typographical errors, optical character recognition errors, and phonetic errors.

Keywords:,

## 1. Introduction and Motivation

Data silos, in which organizations keep their data tightly isolated from other systems, are a major barrier to the effective use of data analytics in many fields. Unfortunately, when the data in question involves information about people, integrating it often necessitates querying or joining based on personally identifiable information (PII) that could be used to explicitly identify an individual. As recent security breaches at organizations ranging from Target to the United States Postal Service have made clear, it is important to protect PII, both while it is at rest on a system and when it is read into memory. The goal of this effort is to explore the applicability, accuracy, speed and security of existing algorithms for querying and joining databases while keeping the PII within those databases protected.

This work focuses particularly on the situation in which a data provider maintains a database which authorized subscribers are able to query. For instance, consider a company that maintains a database containing its customer data. The company wishes to allow third party entities who have contracted with it to access the information in this database.[1] At the same time, the company wants to limit its vulnerability to data breaches by keeping the data encrypted as much as possible, including while it is stored in the database and when it is loaded into memory to do query processing. For instance, if an attacker gets access to the system on which the database resides, he should not be able to see the raw data values, either on disk or in memory.

Even though this situation occurs frequently, research on private record linkage tends to focus more on a different use case, in which the data provider and the data consumer do not fully trust one another. This typically leads to solutions involving trusted third parties or asymmetric cryptography that go beyond the requirements of this ubiquitous application scenario, and these additional, unneeded capabilities negatively im-

*Corresponding author, E-mail: michelle.cheatham@wright.edu

[1] A standard access control system to allow authorized consumers to query the database while preventing unauthorized users from doing so is assumed to be in place.

pact performance. For instance, because access control mechanisms are already in place, this work is not concerned about the data consumer (who has paid to access the information in the database) gaining knowledge of any, or even all, of the records in the database. Furthermore, this project is not concerned about the data consumer preventing the data provider from gaining knowledge about what queries are being made. Rather, the present use case allows a system in which the data consumer submits a query containing the raw PII values, these values are encrypted using symmetric key cryptography[2], and the encrypted values are then used to query the database.

This work focuses on supporting privacy-preserving querying and merging on string attributes and does not consider numeric data. While private record linkage based on numeric fields of is course an important capability to establish, the techniques involved for this are distinctly different than for string-based linking. Furthermore, string attributes, in particular person names, are a particularly common linkage point between datasets. We therefore leave the challenge of numeric attributes for future work and focus on name-based linking here. The requirements of our target application scenario require PRL methods that support encryption and do not need to act directly on the raw field values, so approaches that utilize the original string values at any stage in the process are not suitable in this case. Because names are frequently misspelled, mispronounced, or mistyped, it is important for the approach to support fuzzy (approximate) matching as well as exact matching. This fuzzy matching should be particularly tailored to support the types of lexical variations specific to names. No data should be decrypted, even in memory, until a match is ensured. In this paper we analyze the accuracy and efficiency of several metrics that meet these requirements and compare those results to that of standard name-matching methods employed on unencrypted data. The paper focuses entirely on technical considerations of the targeted use case. Laws and regulations also have a bearing on this application, but that aspect is not addressed here due to wide variance between legal jurisdictions and the authors' lack of legal expertise.

Note that nothing in this application scenario places any restrictions upon the infrastructure in which the data records are stored. In particular, the results presented here can be applied directly, with no modification, to data stored as RDF triples in accordance with the linked data principles. This work therefore joins a growing body of literature regarding how linked data can be secured while retaining its utility for those authorized to access it [6,9].

This paper is an extension of our work presented in [7]. That paper compared the selected name matching techniques based on accuracy and efficiency. Here, we add an analysis of the accuracy impact of varying the value of $q$ in the $q$-grams method (Section 4.6), an in-depth security analysis (Section 5), and an overall recommendation (in Section 6).

The main contributions of this paper are:

- An analysis of the accuracy of several name-based similarity metrics as a function of threshold value on a large dataset containing various types of realistic errors.
- A comparison of the accuracy and computational efficiency of privacy-preserving similarity metrics with those of standard string metrics on unprotected data, which clarifies the price paid in support of data privacy.
- A frequency analysis attack against the best-performing private name matching technique, which provides insight into the level of security achieved through the approach.

The rest of this paper is organized as follows: In Section 2 we provide an overview of some related work and briefly discuss the challenges that make record linkage on names difficult. Section 3 introduces the metrics and algorithms used to perform record linkage in this study. This includes the string similarity metrics for unencrypted data which are used as a baseline for comparison purposes and the metrics relevant to private record linkage. Section 4 analyzes and evaluates the performance of the algorithms mentioned in Section 3 in terms of accuracy, computational efficiency, and security. Finally, Section 6 concludes the paper by summarizing the results and provides an outlook to future work.

## 2. Background

There have been numerous approaches to solving the problem of record linkage based on person names. A comprehensive overview of several name matching techniques was provided by Snae in [12]. Snae describes four different types of name matching al-

[2]Note that this exposes the raw PII values in memory, though only those in the query, not those in every database record.

gorithms and compares them in terms of accuracy and execution time: spelling analysis based algorithms (Guth and Levenshtein), phonetic based algorithms (Soundex, Metaphonez, and Phonex), composite algorithms (a combination of sound and spelling based methods, e.g. ISG), and hybrid algorithms (a combination of phonetic and spelling based approaches, e.g. LIG). The hybrid algorithms were recommended for many name based record linkage applications because of their flexibility, which allows them to be easily tuned for specific use cases. However, the results indicated that there is no single best method for name matching. In the conclusion, the author suggests that the choice of the name matching algorithm should depend on the specific application needs. Moreover, this work doesn't take into consideration the important aspect of our study, which is linking records while keeping them encrypted.

As mentioned previously, many existing techniques for private record linkage assume that the two parties involved do not want to reveal their data to the other party. One way this is commonly achieved is by developing algorithms that avoid directly comparing the records to be linked. For example, in the two-party protocol presented by Vatsalan and his colleagues in [15], two database owners compute similarity values between the records in their dataset and public reference values. Then, the similarity values are binned into intervals and the bins are exchanged between the two database owners. Based on the exchanged bins the protocol uses the reverse triangular inequality of a distance metric to compute the similarity values of two records without revealing the records themselves. Another, somewhat similar, two-party protocol was proposed by Yakout et al. in [17]. In this approach, each database owner converts all of their records into vector representations that are later mapped to points in a complex plane. The planes are then exchanged between the owners in order to identify pairs of points that are in proximity of each other. To calculate similarity values between the candidate vectors, the Euclidean distance of two records is computed using a secure distance computation. These two approaches are typical of many existing PRL techniques and, like the majority of those techniques, they implicitly assume that the records to be linked are not encrypted. We now turn our attention to examples of the few approaches that do not make this assumption.

One important thing to note is that not all string similarity metrics can be applied to the problem of name-based private record linkage. In order for a metric to be usable in this scenario, the metric must not require access to individual characters within this string. This is because any such metric would have to "encrypt" a string character-by-character, which is essentially a classical substitution cipher that is not at all secure. This eliminates common metrics such as Levenshtein and Monge Elkan from consideration.

Among the techniques that support approximate matching for linking records are the Soundex and *q*-gram string similarity metrics. The Soundex metric was originally designed as a phonetic encoding algorithm for indexing names by sound [2]. Soundex encodes a string representing a name into a code that consists of the first letter of the name followed by three digits, by applying a set of transformation rules to the original name. When two Soundex encodings are compared, the comparison is an exact match rather than approximate comparison but common name mispronunciations will not cause the algorithm to miss a match[3]. To use Soundex for private record linkage, both the name and the phonetic encoding are stored in the database in encrypted form for each record, but the encrypted phonetic encoding is the one used to respond to queries. The comparison is still an exact rather than fuzzy comparison, but because it is now being done on a phonetic encoding, common misspellings or other slight differences will not cause the algorithm to miss matching records. This was the approach suggested in [4].

Another of the string similarity metrics that *can* be used is *q*-grams. A *q*-gram is created by splitting a string into a set of substrings of length $q$. An example of a *q*-gram, given *q*=2 and the input string $Alice$, is $\{"Al", "li", "ic", "ce"\}$. As with the Soundex approach, in order to use *q*-grams for name-based private record linkage additional information must be stored with each record. In the case of *q-grams*, the person's name is divided into *q*-grams, each of the substrings in the set of *q*-grams is encrypted, and those encrypted substrings are also stored as part of the record. The amount of similarity between two records is then computed as the degree of overlap between these sets of en-

[3] In 1990 Lawrence Philips created a phoenetic algorithm called Metaphone that improves upon Soundex by considering numerous situations in which the pronunication of English words differs from what would be anticipated based on their spelling [10]. Metaphone was not considered for this effort because the extensions that it makes beyond Soundex are primarily intended to improve the performance on regular words rather than on names; however, the metric does fit the requirements for use in this application, and will be considered during our future work on this topic.

crypted $q$-grams for each record. Each individual substring is compared based on exact match. The degree of overlap is computed using a traditional set similarity metric such as Jaccard or Dice, which are calculated as follows:

$$\text{Jaccard} = \frac{\text{grams}_{\text{common}}}{\text{grams}_1 + \text{grams}_2 - \text{grams}_{\text{common}}}$$

$$\text{Dice} = \frac{2 \text{ x } \text{grams}_{\text{common}}}{\text{grams}_1 + \text{grams}_2}$$

, where $\text{grams}_{\text{common}}$ corresponds to the number of $q$-grams that are common to both strings, $\text{grams}_1$ to the number of $q$-grams in the first string, and $\text{grams}_2$ to the number of $q$-grams in the second string. The intuition behind using $q$-grams to compare two names is that a typo, misspelling, or other variation will only impact a limited number of substrings and therefore similar strings will still have a high degree of overlap and thus a high similarity value. The downside is that the order of the substrings is not considered, so it is possible for two very different strings, such as "stop" and "post" to have very high similarity according to this metric.

A more in-depth review of techniques proposed to achieve private record linkage can be found in [3].

In this work, we have evaluated the performance of the Soundex and $q$-gram algorithms for name-based private record linkage in the scenario described in the introduction. Because it is unrealistic to expect a privacy-preserving record linkage algorithm to perform better than a linkage method that does not provide any protection for the data, we have compared the performance of Soundex and $q$-gram to the performance of some traditional string similarity metrics on unencrypted data. Specifically, we have used Levenshtein and Jaro-Winkler, two of the most commonly used string similarity metrics, as a baseline. Levenshtein is an edit distance metric. It simply counts the number of edits (insertions, deletions, or substitutions) that must be applied to one string in order to transform it into another one. For example, the Levenshtein distance between "Michelle" and "Micheal" is 2. Jaro-Winkler is based on the Jaro metric, which counts the number of "common" characters of two strings. Characters are considered common when the difference between their indexes is no greater than half of the length of the longer string. Jaro also takes into consideration the number of character transpositions. The Jaro-Winkler version of the algorithm increases the similarity value returned by Jaro if the two strings begin with the same sequence of characters and differences appear only in the middle or at the end of string [3].

Name matching has been researched for many years and numerous studies have proven that it is not an easy task. This is because a name's spelling can be malformed in a wide variety of ways, including punctuation, abbreviation, pronunciation, spelling, the order of writing, use of prefixes, typos, or optical recognition errors to name a few. In addition, privacy concerns have made it very difficult to find publicly available data that can be used for benchmark purposes, particularly a collection of names that accurately reflects worldwide name distribution rather than being US-centric. This lack of suitable benchmarks was a considerable challenge during this study, leading to the use of a newly-available name matching benchmark generation system, described in Section 4.

## 3. Approach

We analyzed the performance of several string similarity metrics for linking encrypted records. The metrics considered were Soundex and several variations of the $q$-gram technique. This performance was compared against those of Jaro and a normalized version of Levenshtein on the unencrypted data. The data and Java source code are available from https://github.com/prl-dase-wsu/prl-technique-comparison.

We used two metrics based on $q$-grams. The first is $q$-grams with $q$=2 (also called bigrams). Because studies have shown [8] that padding the input string with a special character (one that never appears as part of any string in the dataset) at the beginning and the end of string can increase the accuracy when comparing two different $q$-grams we also tried padded $q$-grams with $q$=2. Both $q$-grams and padded $q$-grams were compared using two different similarity coefficient methods, Jaccard and Dice.

The string metrics that were used on unencrypted data, Jaro and Leveshtein, were introduced in Section 2. To formally define both algorithms, the similarity value of two strings returned by the Jaro algorithm is calculated as follows:

$$\text{Jaro} = \frac{\frac{c}{s_1} + \frac{c}{s_2} + \frac{c - t}{c}}{3}$$

, where $c$ is the number common characters in both strings, $s_1$ is the length of the first string, $s_2$ is the length of the second string, and $t$ is the number of transposi-

tions (the number of common characters that are not in sequence order, divided by 2). Since all of the metrics used in this study return a value between 0.0 (when strings are completely different) and 1.0 (when string are the same) we modified the original Levenshtein algorithm so that it returns a similarity value that falls in the same range. The Normalized Levenshtein formula is defined as follows:

$$\text{NormalizedLevenshtein} = 1 - \frac{\text{Levenshtein}}{\max(s_1, s_2)}$$

, where *Levenshtein* is the number of replacements needed to transform the first string into the second string, $s_1$ is the length of the first string, and $s_2$ is the length of the second string.

The benchmark datasets used in this study were created by using the advanced personal data generation tool called "GeCo" and developed by K-N. Tran et al. [13] The tool was created to address the issue of lack of publicly available data that contains PII information. GeCo has two main functionalities: data generation and data corruption. The data generation module provides the user with an interface capable of producing records with five different attribute generation mechanisms. The first two can be used to generate individual attributes such as credit card number, social security number, name, age, etc. The attribute values are created by either user-defined functions or based on frequency look-up files that specify the set of all possible values of an attribute and their relative frequencies. The other three types of attribute generation mechanisms allow the user to produce compound attributes where the attributes' values depend on each other. For example, a compound attribute with fields such as: city, gender, and blood pressure can be created, where the value of the blood pressure depends on the previously generated city and gender values. The second module of GeCo provides users with a sophisticated interface allowing them to corrupt the generated data using six different corruption techniques that simulate real-world errors that can occur during data processing. Those techniques include introducing: (1) missing values (one of the record's fields gets lost), (2) character edits (a random character of a string attribute is inserted, deleted, substituted, or transposed), (3) keyboard edits (simulates a human mistake during typing), (4) optical character recognition (OCR) errors (simulates OCR software mistakes), (5) phonetic edits (replaces substrings with their corresponding phonetic variations), and (6) categorical value swapping (replaces an attribute value with one of its possible variations). The user can also specify numerous other parameters such as: the number of records to corrupt, the number of corruptions applied to a record or single attribute, or the probability of corruption of a particular attribute.

For benchmark purposes we generated a dataset of 10,000 records where each of the records had the following attributes: first name, last name and credit card number. Then, we used the GeCo tool to introduce various types of realistic corruption to the generated dataset. The corrupted datasets produced by the GeCo tool were categorized using three parameters: type of applied corruption technique (Character Edit, Keyboard Edit, OCR Edit, Phonetic Edit, or mix of all), the percentage of original record corruption (high - 10%, medium - 5%. or low - 2%), and the number of corruptions applied to either the first name, last name, or both (1 or 2). This resulted in 30 variations of the dataset. Once the datasets were corrupted we added additional attributes to each of the records from all datasets to be able to perform record linkage using encrypted *q*-grams and Soundex encodings. Each *q*-gram array and Soundex encoding were encrypted using 256-bit AES password-based encryption.

To evaluate the performance of the string metrics, the uncorrupted dataset was cross joined with each of the corrupted datasets using each of the string metrics discussed in the previous section. During the join operation only the pairs of records with the highest similarity score that exceeded a threshold value were joined. If the an individual pair of joined records corresponded to the same individual we counted it as a "Correct" join, otherwise the join was counted as a "False Positive". If none of the scores returned by a string metric exceeded a threshold value we incremented the "False Negative" count by one to indicate that a corresponding record was not found in the other dataset. In a special case, when more than one pair of records had the same highest score, the pair of records that corresponded to the same individual was marked as "Correct" and the rest of the pairs were counted as "False Positive".

## 4. Evaluation and Analysis

Performing record linkage using each of the seven string metrics (*q*-grams compared using Jaccard coefficient, *q*-grams compared using Dice coefficient, padded *q*-grams compared using Jaccard coefficient,

padded $q$-grams compared using Dice coefficient, Soundex, Levenshtein, and Jaro) between the uncorrupted dataset and the 30 corrupted datasets resulted in a massive amount of statistical data. Instead of presenting the outcome of every single cross join operation, this section summarizes our key findings, with an emphasis on practical advice related to selecting a metric, setting the threshold, and conveying the type of performance that can be expected by someone attempting to do name-based private record linkage.

### *4.1. Observation 1: Soundex is not viable, but (padded) q-grams are*

Figure 1 Shows the results of all of the string metrics on a version of the data in which 10 percent of the names have had one (the solid lines) or two (the dotten lines) characters edited. In the single edit case, all versions of the $q$-gram metric are able to achieve the same, nearly perfect, accuracy on the encrypted data that Levenshtein and Jaro achieve on the encrypted data. The performance of all metrics is lower for the two character edit case, with a top accuracy of 90 percent rather than the completely accurate results possible in the single edit situation. However, we again see that the performance of at least the padded versions of the $q$-gram approach on the ciphertext can match that of Levenshtein and Jaro on the plaintext.

The results for the Soundex metric are not included in Figure 1 because the results showed that comparing names based on encrypted Soundex encodings is not viable in most of the cases as a record linkage technique. The only exception was noted when the datasets containing records with the phonetic type of record corruption were joined. Still, in the best case scenario only 60.71% of corrupted data was successfully matched using this technique. Table 1 presents the accuracy of record linkage on all types of corrupted datasets using the Soundex technique.

Table 1

Performance of record linkage based on encrypted Soundex encodings. The percentage values reflect the number of corrupted records that were successfully matched with their uncorrupted versions.

| Corruption type | Number of corruptions per record | |
|---|---|---|
| | 1 | 2 |
| Character Edit | 47.24% | 24.06% |
| Keyboard Edit | 48.06% | 21.94% |
| OCR Edit | 38.29% | 13.71% |
| Phonetic Edit | 60.71% | 43.88% |
| Mix | 50.82% | 25.47% |

### *4.2. Observation 2: Dice is preferable to Jaccard for calculating q-gram similarity*

Out of the four similarity metrics based on $q$-grams, the ones using the Dice coefficient to measure the similarity between the sets of encrypted $q$-grams were more accurate. This was the case with $q$-grams as well as padded $q$-grams. This can be explained by the fact that Dice favors the occurrences of common $q$-grams more than Jaccard. As a result, a pair of similar records is likely to have a higher similarity score when calculated using Dice coefficient. To illustrate this, in Table 2 we provide a sample results from record linkage performed against a dataset with the phonetic type of corruption, where 10% of original records had two phonetic errors introduced. Similar results were recorded for datasets with other types of corruptions.

### *4.3. Observation 3: Lower thresholds are better for q-grams*

Figure 1 illustrates that the threshold value for the Levenshtein and Jaro metrics can be set relatively high without sacrificing accuracy when linking the unencrypted data, which was not the case when the $q$-gram techniques were used to link the encrypted data. For instance, to achieve an accuracy of 99.5% when performing linkage against datasets where records contain one corruption of any type, the threshold value applied to the Jaro or Levenshtein metric was set to 0.8 whereas the threshold value applied to $q$-grams based metrics needs to be set to a value between 0.55 and 0.75 to achieve the same result, depending on the type of corruption applied to the datasets.

Table 2 makes the point that the padded versions of the $q$-gram metric in particular have better performance when the threshold value is kept low, which as explained in the previous paragraph is the optimal approach. For threshold values up to 0.7 for the Jaccard coefficient and 0.8 for the Dice coefficient, padding the $q$-grams produces better results. For higher threshold values, the unpadded version is slightly better. The reason behind this is that similarity scores calculated using padded $q$-grams are higher when the differences between the strings used to generate the $q$-grams appear in the middle of the strings. [3] When the differences appear at the beginning or at the end of strings the similarity scores are lower because the number of common $q$-grams is smaller. Statistically, the differences between strings appear more often in the middle, which explains why the padded $q$-grams can pro-

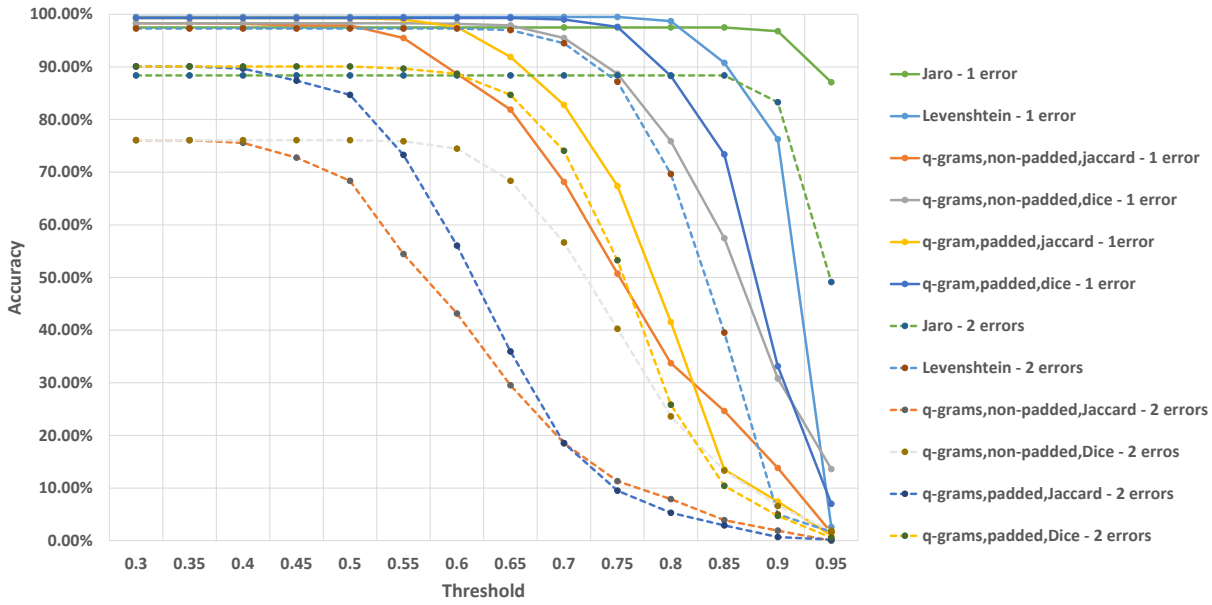



Fig. 1. Illustration of the decrease in accuracy of record linkage of selected string metrics. Solid lines correspond to accuracy when linkage was performed on datasets corrupted with 1 Character Edit, dotted lines with 2 Character Edits.

Table 2

Sample results of record linkage performed against phonetically corrupted dataset showing the performance of *q*-grams based string similarity metrics

| | unpadded q-grams | | | | | | padded q-grams | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Jaccard | | | Dice | | | Jaccard | | | Dice | | |
| Threshold | Correct | FP | FN | Correct | FP | FN | Correct | FP | FN | Correct | FP | FN |
| 0.30 | 9837 | 60 | 163 | 9837 | 60 | 163 | 9949 | 45 | 51 | 9949 | 45 | 51 |
| 0.35 | 9837 | 60 | 163 | 9837 | 60 | 163 | 9949 | 45 | 51 | 9949 | 45 | 51 |
| 0.40 | 9834 | 57 | 166 | 9837 | 60 | 163 | 9948 | 45 | 52 | 9949 | 45 | 51 |
| 0.45 | 9814 | 56 | 186 | 9837 | 60 | 163 | 9937 | 43 | 63 | 9949 | 45 | 51 |
| 0.50 | 9778 | 48 | 222 | 9837 | 60 | 163 | 9910 | 38 | 90 | 9949 | 45 | 51 |
| 0.55 | 9614 | 29 | 386 | 9834 | 57 | 166 | 9790 | 27 | 210 | 9949 | 45 | 51 |
| 0.60 | 9506 | 25 | 494 | 9826 | 57 | 174 | 9589 | 18 | 411 | 9946 | 45 | 54 |
| 0.65 | 9329 | 21 | 671 | 9778 | 48 | 222 | 9337 | 18 | 661 | 9910 | 38 | 90 |
| 0.70 | 9198 | 18 | 802 | 9637 | 32 | 363 | 9170 | 18 | 830 | 9792 | 27 | 208 |
| 0.75 | 9101 | 18 | 899 | 9467 | 22 | 532 | 9081 | 17 | 919 | 9538 | 18 | 462 |
| 0.80 | 9046 | 18 | 954 | 9264 | 19 | 736 | 9032 | 17 | 968 | 9233 | 18 | 767 |
| 0.85 | 9023 | 18 | 977 | 9125 | 18 | 875 | 9011 | 17 | 989 | 9097 | 17 | 903 |
| 0.90 | 9009 | 18 | 991 | 9036 | 18 | 964 | 9002 | 17 | 998 | 9021 | 17 | 979 |
| 0.95 | 9002 | 18 | 998 | 9009 | 18 | 991 | 9000 | 17 | 1000 | 9002 | 17 | 998 |

duce higher similarity scores for the majority of corrupted data. This pattern occurred in all of the results produced during this study.

### *4.4. Observation 4: Some types of errors are worse than others*

Out of all corrupted datasets the worst performance in terms of accuracy and the number of false positives found was "achieved" when the datasets with OCR Edits were linked. This is most likely due to the fact that

some of the mistakes that OCR Edits introduce are replacements of two characters in place of one character, or vice versa. For instance, character "m" can be replaced with "rn" and the string "cl" can be replaced by the character "d". Those kind of replacements can have a significant negative impact on the string similarity scores produced by all of the metrics. The best performance results were recorded when the datasets corrupted with Character Edits were linked, those are presented in Figure 1. Figure 2 illustrates the accuracies of linking datasets corrupted with OCR Edits errors. The accuracies of datasets corrupted with Keyboard Edits, Phonetic Edits, and a mix of all types of edits fall in between the accuracies presented in Figure 1 and Figure 2.

Another pattern common for the results obtained from linking all types of corrupted datasets was a significant drop in accuracy when the corrupted records contained more than one error of any type. For instance, for *Threshold*=0.85 the accuracies of the Jaro, Levenshtein, unpadded $q$-grams compared using the Dice coefficient, and padded $q$-grams compared using the Dice coefficient were 97.5%, 90.79%, 57.46%, and 73.37% respectively when there was only one error of the Character Edit type per record. When the number of errors per corrupted record increased to two, the accuracies decreased to 88.39%, 39.54%, 13.31%, and 10.41%. Figure 1 presents a full overview of the accuracy degradation for datasets corrupted with Character Edits where 10% of all original records were corrupted.

### 4.5. *Observation 5: The efficiency penalty for these privacy-preserving string similarity metrics is small*

The Jaro, Levenshtein, and Jaccard and Dice variants of the $q$-grams metric all have a $O(nm)$ time complexity, where $n$ and $m$ are the lengths of the strings to be compared. Because the Soundex metric is only checking for equality of the Soundex representation of the strings, its time complexity just $O(n)$. When determining whether a particular name is in the dataset, the query name is compared against all of the names in the dataset. It should be noted that the Soundex algorithm, because it is an exact rather than fuzzy match, could be made more efficient by indexing the database on the Soundex representation of the name. Also, there has been some work on eliminating the need to consider all names in the dataset when querying using the Jaro metric through the user of character and length-based filters to quickly determine if it is possible for a particular name to match the query within a specified threshold [5]. Neither of these optimizations were considered in this work.

While most of the string metrics considered have the same computational complexity, constant factors differ between the approaches. For example, because Jaro only looks for matching characters within a window that is half the length of the longer string, it is generally faster than Levenstein. To evaluate the computational efficiency of each of the string metrics in a practical setting, the time taken to perform the join operation between the original dataset and the corrupted dataset was measured. We explored the impact on the performance when the number of characters in names increases. In this case, the datasets always consisted of 10,000 records but the number of characters in each name was equal to 10, 15, or 20. These tests were done on datasets with a record corruption of 10%, where the records were corrupted using the Character Edit technique and contained one corruption per record. The results are shown in Figure 3. The timing results of linkage performed on the other corrupted datasets were very similar to the ones presented in this figure.

The results show that the $q$-grams approaches are very slightly faster than Jaro in these tests, and significantly faster than Levenshtein. The average time taken to perform the join operation on the datasets using Levenshtein was more than five times the magnitude of the time taken by the other string metrics. Of course, the best speed was observed when the datasets were linked using the Soundex metric. In those cases the linkage was performed almost instantly, averaging only about one second.

Additionally, we have investigated the impact on the performance when the number of records increases. Three datasets of different volumes (10, 20, and 30 thousand records) were linked to conduct the tests. The results shown in Figure 3 indicate that as the number of records to be linked increases, the time required to link all the records is again very similar for Jaro and the $q$-grams techniques, significantly greater for Levenstein, and very low for Soundex.

### 4.6. *Observation 6: As q increases, record linkage quality degrades.*

We also have studied the effects of the value of $q$ on the accuracy of the $q$-grams approach. As $q$ increases, fewer $q$-grams are needed to encode a string. This implies that every difference between two strings en-

Fig. 2. Accuracy of string metrics used to perform record linkage on dataset with 10% of the records corrupted using OCR Edits with 1 corruption per record.

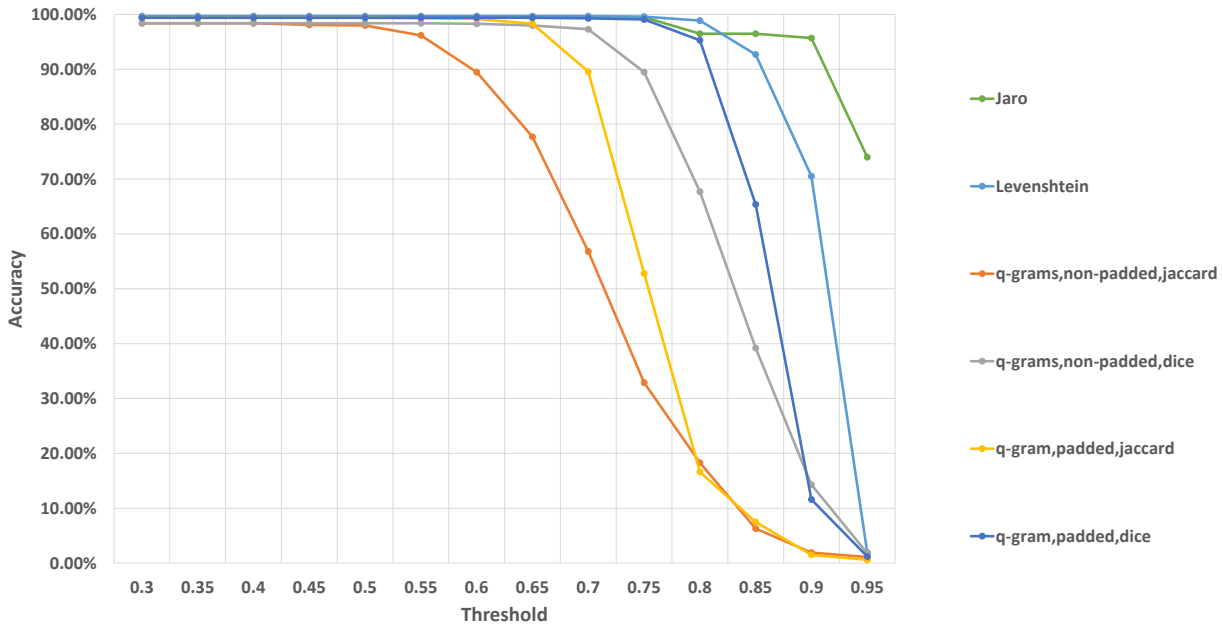
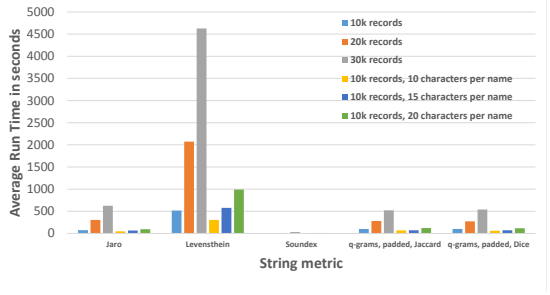



Fig. 3. The average time of record linkage using selected string metric techniques.




coded using $q$-grams and compared using the Dice or Jaccard coefficient will be penalized more and more as $q$ increases. It is therefore expected that performance will drop as $q$ increases. Our goal was to investigate the magnitude of this drop. We therefore studied the accuracy of the $q$-grams approach for $q$ equals 2, 3 and 4.

As expected, bi-grams ($q$=2) yielded the best results for all types of datasets. Figure 4 presents the accuracy of record linkage of datasets containing keyboard errors (10% of records were corrupted with one random insertion, deletion, substitution, or transposition of a character). The records were linked based on Dice and Jaccard coefficient scores. The solid lines represent the results of linking records using unpadded $q$-grams, the dotted lines indicate that the $q$-grams were padded. All versions of $q$-grams demonstrated a very similar degradation pattern in linkage quality. For example, when the records containing unpadded $q$-grams were linked based on Dice coefficient scores with a threshold value set to 0.65 the percentages of successfully linked records (that contained the aforementioned corruption) for $q$=2, 3, and 4 were equal to 99%, 84%, and 57% respectively. This pattern holds for all types of dataset corruptions (character, keyboard, OCR, or phonetic errors). When the number of corruptions per record increases to two, the record linkage quality suffers even more. Figure 5 illustrates the accuracy of record linkage of datasets where 10% of records were corrupted with two random keyboard errors (Figures 4 and 5 were created using the same color-coded series representing the linkage performed using the same $q$-gram, padding, and $q$-gram similarity coefficient combination to allow easy comparison between the results). For low threshold values, the record linkage using $q$-grams with $q$=4 resulted in correct linkage of only about 15% of corrupted records, and even the usage of padded versions of $q$-grams didn't significantly improve the results. The linkage utilizing tri-grams and bi-grams performed 3 to 6 times better depending on the particular combination of $q$ value, padding, and $q$-gram similarity coefficient. As before, the same linkage quality degradation pattern was reported for all types of data corruption.

Fig. 4. Illustration of accuracy of record linkage based on different types of $q$-grams. Linkage was performed on datasets corrupted with 1 Keyboard Edit.

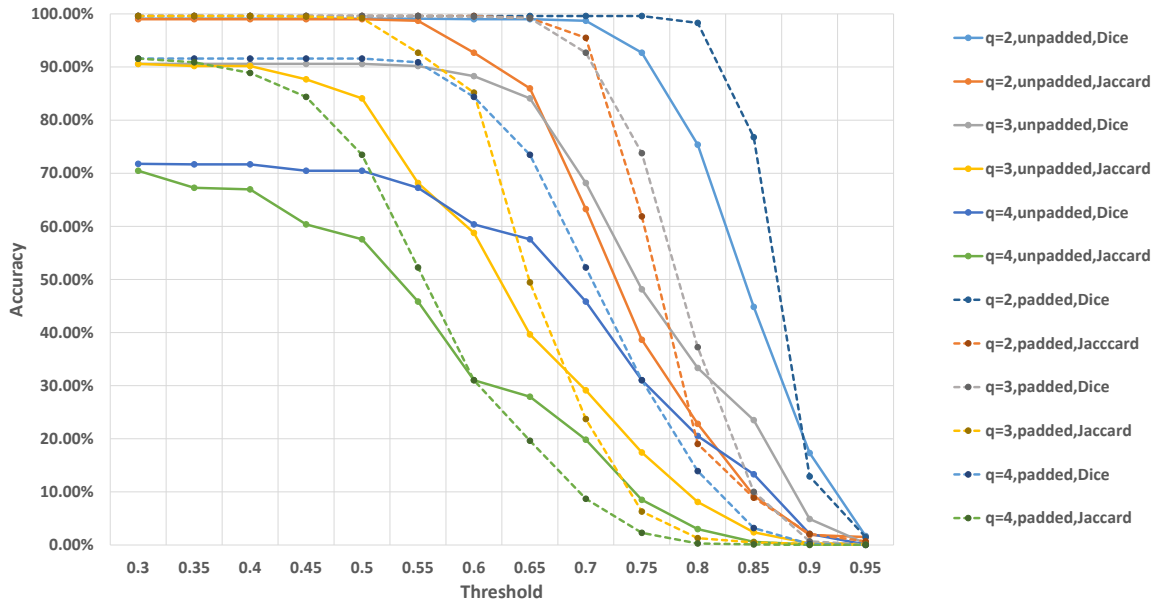
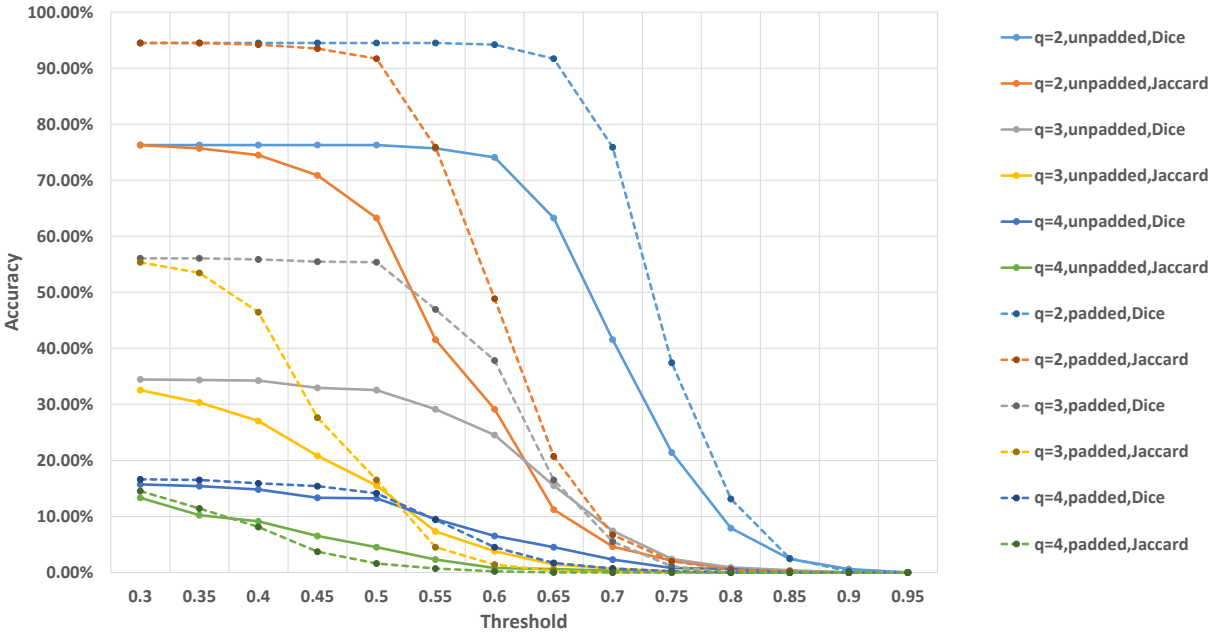



Fig. 5. Illustration of accuracy of record linkage based on different types of $q$-grams. Linkage was performed on datasets corrupted with 2 Keyboard Edits.




## 5. Evaluation of the Privacy Level of $q$-grams

An important measure of any PRL technique, in addition to its performance and scalability, is the level of privacy. In general, the PRL privacy requirements assume that once the record linkage is complete only a limited amount of information will be shared between the interested parties [16]. The "limited amount" term usually means revealing data such as the number of linked records, explicit identifiers of the linked records, or linked records along with their selected

attributes. The work presented here assumes that the querying entity can discover as much information about the records in the database as it wishes. Here, the emphasis is instead on making sure the records (more specifically the PII attributes) are always encrypted in order to protect the identities of persons associated with the records in the event of a data breach. For the $q$-grams based techniques, each of the $q$-grams was encrypted with an AES password-based encryption algorithm. A brute force attack (i.e. dictionary attack) on this kind of encryption algorithm has proven to be infeasible [1]. However, as other researchers have noted, $q$-gram based techniques are prone to a frequency based attack. [11,4] A frequency attack can be mounted by comparing the frequency of each encrypted $q$-gram with its relative frequency derived from data that is likely to have a similar distribution of $q$-gram occurrences. In the case of first and last names, such data can be very easily found online. For example, a dataset that contains more than 5,000 first names and nearly 90,000 last names was published by the U.S. Census Bureau in 1990 [14]. To evaluate the privacy level of $q$-gram based techniques we want to answer the following question: What percentage of records can be discovered by conducting a frequency based attack on first and last names? This section describes an overview of the approach taken to prepare the attack and presents our findings and observations.

### *5.1. Frequency Attack Approach*

The ultimate goal of the attack is to decrypt as many first and last names that are stored in the form of encrypted $q$-grams as possible. When designing the attack the following assumptions were made:

- The attacker managed to obtain access to all encrypted records from the database, along with all attributes of each record, but he/she doesn't have access to their unencrytped version.
- The attributes corresponding to first and last names can be identified by the attacker. In other words, the attacker knows the schema of the record and can identify which $q$-grams correspond to first name or last name. Additionally, the $q$ number used to form $q$-grams is also known.
- To fully explore the vulnerabilities of $q$-gram based techniques none of the names (first or last) in the attacker's dataset was corrupted since random errors in the dataset could influence the results of the attack.
- Since the attacker can distinguish between $q$-grams corresponding to first name or last name, the attack will be conducted on first and last names separately.
- The lists of first and last names published by the U.S. Census Bureau will be used by the attacker to calculate relative frequencies of $q$-grams.

The simplest and most intuitive approach to conduct the frequency analysis attack would be as follows. The attacker creates a list of encrypted $q$-grams containing the frequency of occurrences of each of the $q$-grams. Using publicly available sources of first and last names, the attacker can prepare similar lists with unencrypted versions of the $q$-grams. Finally, the two lists can be ordered by the $q$-gram frequencies and merged together in order to substitute the encrypted $q$-grams with their unencrypted equivalents in all records from the database. However, such a naive approach will not work. Since the attacker has no knowledge about the name distribution in the encrypted database, the two lists of $q$-gram frequencies are not likely to provide enough correctly guessed $q$-gram decryptions to decrypt a significant number of names. Table 3 presents a selected part of the two lists created by analyzing $q$-gram frequencies of first names, where the $q$ number used to form the $q$-grams was equal to 2. Out of 323 encrypted $q$-grams there were only eight matches where the $n^{th}$ $q$-gram from the encrypted dataset had the exact same frequency rank as the $q$-gram derived from the public dataset. A brute force solution to substitute every single possible decryption for each of the encrypted $q$-grams is not feasible. Such an attempt will result in creating $323^3$ variations of $q$-gram decryptions. Evaluating each of the variations would require too many computational resources. Therefore, it is clear that a more sophisticated approach is needed.

To conduct the frequency attack in an efficient manner we decided to take an advantage of the fact that the frequencies of $q$-grams derived from the public dataset are usually not too far off from the frequencies of the encrypted $q$-grams. For example, the encrypted $q$-gram "ar" in Table 3 was classified as the $6^{th}$ most frequent in the encrypted database and as the $2^{nd}$ most frequent among $q$-grams derived from the public dataset of names. This information can significantly reduce the number of possible decryption variations for the encrypted $q$-grams. To reduce it even further, we propose the following technique that analyzes each of the encrypted records and identifies the most probable $q$-gram translation.

Table 3
Top 10 most commonly occurring $q$-grams in encrypted and public datasets of first names where $q$=2.

| | encrypted dataset | | | public dataset | |
|---|---|---|---|---|---|
| | q-gram | decrypted version | frequency | q-gram | frequency |
| 1 | BMNjNlgaU6MYKOYS7jEbcQ== | an | 1769 | an | 819 |
| 2 | WNd7LpS8QCJZQbxTI5CWnQ== | el | 1086 | ar | 646 |
| 3 | iWCuQ5xviCTYO6B+LGBLDA== | li | 1002 | el | 639 |
| 4 | LuzDuWRkbEwZK8xmRqA49A== | le | 972 | ri | 530 |
| 5 | tJIjS5UyZTKCiw3XkEYQMg== | ia | 942 | na | 498 |
| 6 | TEdzDzlf+TzuFA4+wrJ3og== | ar | 924 | le | 494 |
| 7 | 9pHgQ2qQC1XGJ4u8SKZDmQ== | la | 838 | in | 491 |
| 8 | Tbm4IHyAi7hwjB2sZAL6ag== | na | 799 | ne | 488 |
| 9 | TvNEdMt1kMGkccoSPJzeMA== | ha | 764 | en | 473 |
| 10 | +Sbh5wED8+a/lMhao3C6GQ== | en | 744 | er | 467 |

Once a table similar to the one present in Table 3 is obtained, the attacker iterates through each of the encrypted records. Based on the assumption that the $q$ number is known to the attacker, the length of the encrypted name can be deduced using the formula: $L = n + q - 1$, where $L$ is the length of the name, $n$ is the number of $q$-grams used to form the name and $q$ is the $q$ number. The length of the name is used to pull all instances of names from the publicly available dataset of the same length. Those names will serve as guesses of the encrypted name. It is likely that the number of names of length $L$ will be in the hundreds or even thousands. To reduce this number, in the next step the attacker prepares another table with the number of rows equal to the number of encrypted $q$-grams in the record. Each row contains an encrypted $q$-gram and a list of potential translations of this $q$-gram obtained from a table such as Table 3. The list is populated with the $q$-gram translation that matches the frequency rank of the encrypted $q$-gram as well as additional $q$-gram translations that are within a specified range (later on referred to as the "Guessing Offset") above and below the frequency matching $q$-gram. An example of such a table is presented in Table 5, where each possible $q$-gram translation includes the information about the offset between a translation itself and the translation that matches the frequency of the encrypted $q$-gram. The offset represents how close the frequency rank of a potential $q$-gram translation is to that of the frequency of the encrypted $q$-gram. For example, the second encrypted $q$-gram in Table 5 “TEdzDzlf+TzuFA4+wrJ3og==” includes possible translations such as “le:0”. “le” is the translation that matches the frequency rank of the encrypted $q$-gram (see Table 3), so there is no offset. Another possible translation “ar:4” indicates that the $q$-gram “ar” has an offset of 4 between “ar” and the $q$-gram that matches the frequency rank of the encrypted $q$-gram (“le”). This table is used to eliminate the guesses of the encrypted name that are not valid due to the fact that they cannot be formed given the lists of possible $q$-gram translations for each of the encrypted $q$-grams. For example, in Table 5, the possible $q$-gram translation “ri” can be eliminated from the second row because none of the possibilities in the first row end with “r”. The corresponding list of possible guesses for the encrypted name can then be further reduced.

In the final step, each of the names in the guesses list is assigned a penalty score. The score is the sum of the offsets that had to be taken to include all $q$-grams used to form the name. Based on Table 5, the name “tere” would receive the penalty score of 1 (“te:1”) + 4 (“er:4”) + 2 (“re:2”) = 7. The name with the lowest score is identified as the best possible guess for the encrypted name. The $q$-grams used to form this guess are stored in a separate table (called the Translation Table) that contains the encrypted $q$-grams and their corresponding translations obtained from the name with the lowest penalty score. When the attacker is done iterating through all of the encrypted records, the translation for an encrypted $q$-gram is identified by taking its most common translation from the Translation Table. The attacker can now substitute the encrypted $q$-grams with their unencrypted versions in all encrypted names in the dataset.

### *5.2. Sample Frequency Attack Step*

We now provide an example iteration of the frequency attack approach described above. Table 4 illustrates one of the encrypted records that the attacker obtained from the database during the security breach.

The attribute called 'First Name' represents a set of encrypted $q$-grams separated by commas that were used to encrypt the name "sara".

Table 4
An example of an encrypted record.

| Record Id | First Name | Last Name |
|---|---|---|
| rec-1081 | uSpcZZ71KiNrD8Av/FS3cA==, TEdzDzlf+TzuFA4+wrJ3og==, 2gP9jD8Zg5ugC6YTQg3UMQ== | ... |

The attacker can infer the length of the name behind the encrypted $q$-grams by applying the following formula: $length = 3 + (2 - 1) = 4$ since he knows that the $q$ number is equal to 2. From a publicly available dataset of names, he or she can obtain the list of all names of length 4: john, hera, sara, carl, tera, tere, etc. to serve as the guesses list. To trim the list, Table 5 is prepared where each of the encrypted $q$-grams has a list of its potential decryptions obtained from a table similar to Table 3. Assume that the maximum "guessing offset" is equal to 5 (i.e. each $q$-gram translation no further than 5 places up or down the list is considered similar).

Table 5
Lists of possible translations for encrypted $q$-grams used to encrypt the name "sara"

| encrypted q-gram | possible translations (q-gram:offset) |
|---|---|
| uSpcZZ71KiNrD8Av/FS3cA== | nd:5,il:4,de:3,ro:2,te:1,he:0, mi:1,sa:2,th:3,st:4,tt:5 |
| TEdzDzlf+TzuFA4+wrJ3og== | an:5,ar:4,el:3,ri:2,na:1,le:0, in:1,ne:2,en:3,er:4,la:5 |
| 2gP9jD8Zg5ugC6YTQg3UMQ== | li:5,ie:4,ma:3,ha:2,on:1,ra:0, ia:1,re:2,da:3,ta:4,al:5 |

Based on Table 5 the attacker can eliminate most of the names from the guesses list. For example, the name "john" is not a valid option because there is no "jo" $q$-gram as a possible translation for the first encrypted $q$-gram. The only names that are left are: tera, tere, hera, and sara. Each of the names is assigned a penalty score that is the sum of the offset associated with each $q$-gram required to form a particular name in the list of possible translations. For example, the name tera (te:1, er:4, ra:0) will be assigned a score of 5. The name with the lowest score, which in this example is hera (score of 4), will be considered the correct translation of the encrypted name. The attacker will keep the corresponding $q$-gram translations in the separate table (see Table 6).

Table 6
List of encrypted $q$-gram translations after one iteration

| encrypted q-gram | possible translation (q-gram:occurrences) |
|---|---|
| uSpcZZ71KiNrD8Av/FS3cA== | he: 1 |
| TEdzDzlf+TzuFA4+wrJ3og== | er: 1 |
| 2gP9jD8Zg5ugC6YTQg3UMQ== | ra:1 |

Once all records have been processed by the attacker, the translation for an encrypted $q$-gram will be selected from Table 6. If there is more than one option for the translation, the one with the higher number of occurrences will be considered the correct one.

### *5.3. Frequency Attack Analysis*

The frequency attack was conducted on a dataset containing 10,000 records. Each record included two sets of encrypted $q$-grams: one set corresponding to the $q$-grams that represented a first name and another one corresponding to a last name. To study the effectiveness of the attack on different type of $q$-grams, four variations of the dataset were generated, each using a different value for $q$, from 1 to 4. The attack was carried out on first and last names separately. This section provides an overview of our most important findings.

#### *5.3.1. Observation 7: Security increases as q increases (with one exception)*

To convey the results of the frequency analysis attack, we present three figures that illustrate the key statistics. Figure 6 shows the percentage of names that were fully decrypted. In the case in which the dataset of first names were encoded with unigrams (i.e. $q$-grams with $q$ equal to 1) we were able to correctly decrypt 66% (about 550 names) of all first names in the database (blue solid line). When first names are encrypted with $q$ equal to 2 (the solid orange line), only about 14% of names were successfully decrypted. For $q$ equal to 3 and 4, the percentage of decrytped names drops near zero. The pattern for last names is similar, with the exception of when $q$ equals 1, which will be discussed shortly. When $q$ equals 2, 28% of last names were fully decrypted, but when $q$ equals 3 or 4, this value never reaches 2%.

The performance degradation can be explained by looking at the number of unique $q$-grams in the dataset as $q$ increases. Table 7 provides detailed information regarding the exact number of unique $q$-grams in each

dataset. For instance, with $q$ equal to 2 there are 323 unique $q$-grams in the case of first names, and 644 in the case of last names. When $q$ is equal to 4 the number of unique $q$-grams increases to 1757 and 15146 respectively. With higher values of $q$ the chance of identifying a correct translation for an encrypted $q$-gram is smaller, and the potential attacker would likely have to conduct a brute force attack to decrypt a significant number of names.

The reason the frequency analysis attack was unsuccessful when $q$ equaled 3 or 4 is because our benchmark dataset of 10,000 records didn't contain enough samples of many $q$-gram to decrypt them. Some of the $q$-grams occurred only a few times across all of the records, making the $q$-gram distribution very uniform. The proper way to attack a dataset with this kind of distribution is a brute force attack (i.e. substituting every single possible $q$-gram translation for each of the encrypted $q$-grams). Such an attack would require a very high amount of computational resources. For instance, the number of unique $q$-grams for our dataset of last names when $q$=4 is equal to 15,146. This results in more than 230 million possible combinations of $q$-gram substitutions.

Table 7
Number of unique $q$-grams in datasets of first and last names.

| | number of unique q-grams in dataset of | |
|---|---|---|
| q number | first names | last names |
| 1 | 28 | 29 |
| 2 | 323 | 644 |
| 3 | 1256 | 5378 |
| 4 | 1757 | 15146 |

Fully decrypted names are not the only ones the attacker might be interested in. A name that is partially decrypted, for example where eight out of ten $q$-grams were correctly decrypted, might also have a high value to the attacker. Figure 7 presents the average rate of decryption of first and last names. We define the rate of decryption as the percentage of correctly decrypted $q$-grams in a name. In the aforementioned example, correct decryption of eight out of ten $q$-grams equates to an 80% rate of decryption. By looking at the series in Figure 7, one can see that the average rates of decryption for both first and last names are similar for corresponding $q$ values. The maximum average rates of decryption for first and last names are very high for $q$=1 and $q$=2: 93% and 63% for first names, and 62% and 77% for last names. Partially decrypted names might be easy to guess and can provide a huge aid in discovering decrypted values of encrypted $q$-grams, which may consequently lead to discovering the identity of all entities in the encrypted database.

#### 5.3.2. *Observation 8: The Guessing Offset must be increased as q increases, making the attack more expensive*

Figure 8 shows the percentage of correctly decrypted $q$-grams out of all of the $q$-grams used to form first or last names as a function of the Guessing Offset. With $q$=1 the decryption percentage increases very quickly even for low ranges of the Guessing Offset. It is worth noting that in the case of $q$=1 and Guessing Offset greater than 20, the $q$-gram decryption percentage is at its highest (63% and 83% for first and last names respectively). The reason it never reaches 100% is because of the nature of the frequency analysis attack we took. Some of the $q$-grams occur so rarely (or very evenly when compared to others) that their potential translations result in too many possible options and they are never identified correctly. This could possibly be improved by using a different source of names (i.e. other than ones coming from the U.S. Census Bureau). The series for $q$=2 reach their maximum with the Guessing Offset greater than 160 (not shown on the graph). Presumably, the series that correspond to $q$ equal to 3 and 4 would eventually also reached their maximum, though at an impractically high Guessing Offset. Another interesting observation that can be made by analyzing the data in Figure 8 is that the $q$-gram decryption ratio is not consistent with the percentage of fully or partially decrypted names. For example, with the $q$-gram decryption rate between 30% and 40% for $q$=1, we were able to fully decrypt only 2% of last and 1% of first names, and achieve a 36% and 56% average rate of decryption. For the same $q$-gram decryption rate for $q$=2, the numbers increase to 23%, 9%, 75%, and 56% respectively. As the $q$ number increases from 1 to 2 the distribution of the $q$-grams becomes less uniform, which makes it easier to identify $q$-grams required to decrypt (or partially decrypt) a significant number of names.

#### 5.3.3. *Observation 9: Some name collections are more vulnerable than others*

The exception mentioned in section 5.3.1 occurs when the $q$-gram approach with $q$ equal to one is used to protect the collection of last names. In this case the dataset is surprisingly resilient to a frequency analysis attack. Based on Figures 6, 7, and 8 the percentage of decrypted last names should be relatively close to the percentage of decrypted first names. The percentages

Fig. 6. The percentage of decrypted first and last names for different values of $q$.

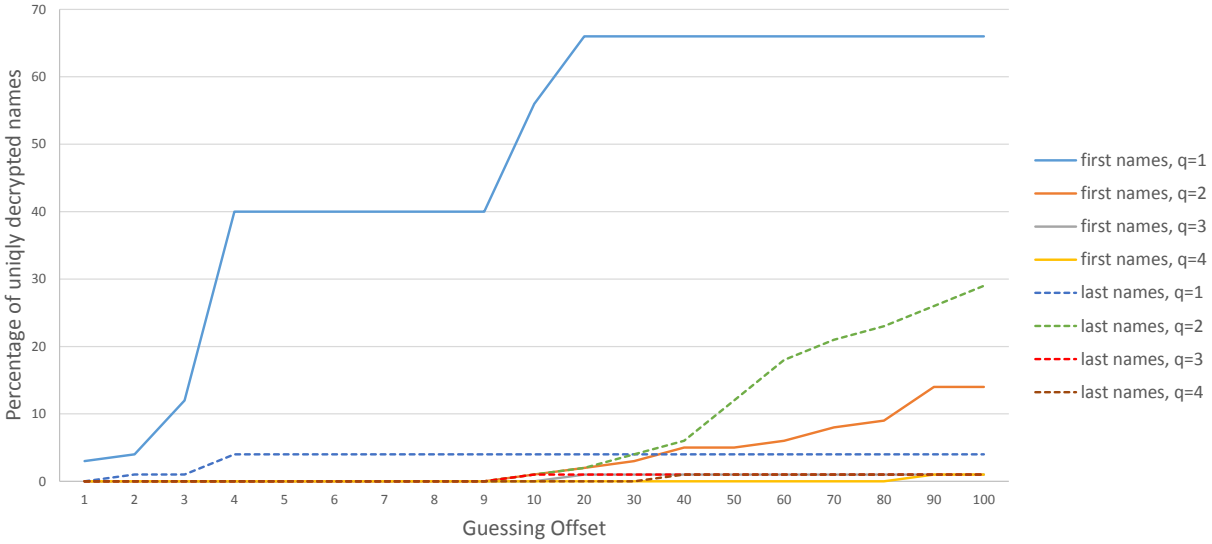
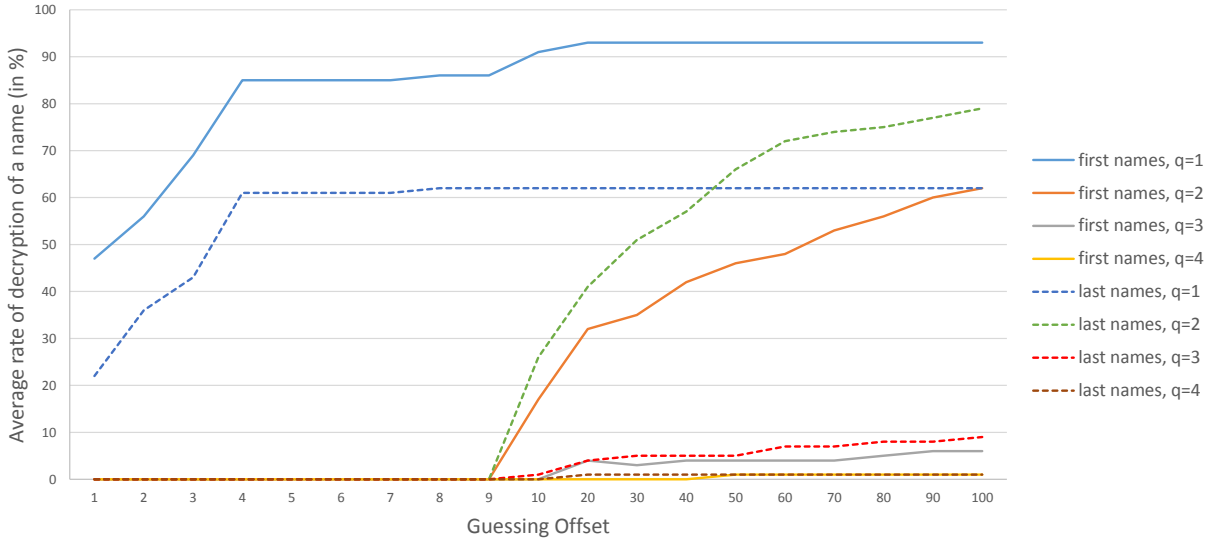



Fig. 7. The average rate of decryption of first and last names for different values of $q$.




of decrypted q-grams occuring in first and last names when $q$=1 were equal to 83% and 63% respectively, and the average rates of first and last name decryption were equal to 93% ad 62%. Yet, only 4% of last names were fully decrypted, compared to 66% of first names. The surprisingly low percentage of fully decrypted last names can be explained by two properties of the datasets of first and last names that were used to conduct the frequency analysis attack. Firstly, the average length of a first name is shorter by 1.5054 characters than the average length of a last name. Shorter length means that fewer $q$-grams are required to fully decrypt a name. Additionally, Figure 9 shows that more than 40% of all last names are longer than eight characters, while 40% of all first names contain fewer than six characters.

Another property that makes the last names harder to decrypt is a more uniform unigram distribution across all of the records. The distributions are shown in Figure 10. The less uniformly $q$-grams are distributed, the easier it is to guess their correct decryption. In the case of first names, the two most frequent unigrams

Fig. 8. The rate of $q$-gram decryption for first and last names encoded with different values of $q$.

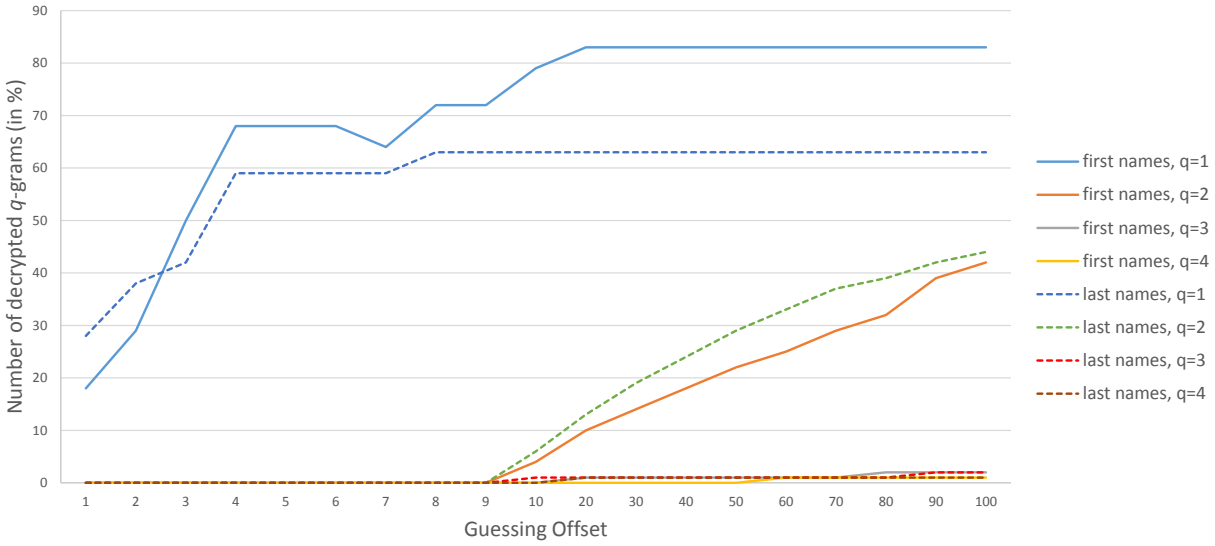
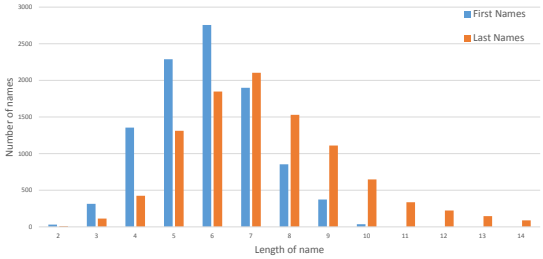



Fig. 9. First and last name distribution based on the length of the name.




can be clearly identified ('a' and 'e'). In case of last names, 'a' and 'e' have almost identical frequency. Moreover, the second most frequent unigram in last names is also hard to select because unigrams 'i', 'l', 'n', 'o', 'r', and 's' have very similar frequency of occurrence. These two properties have a strong influence on the number of successfully decrypted $q$-grams.

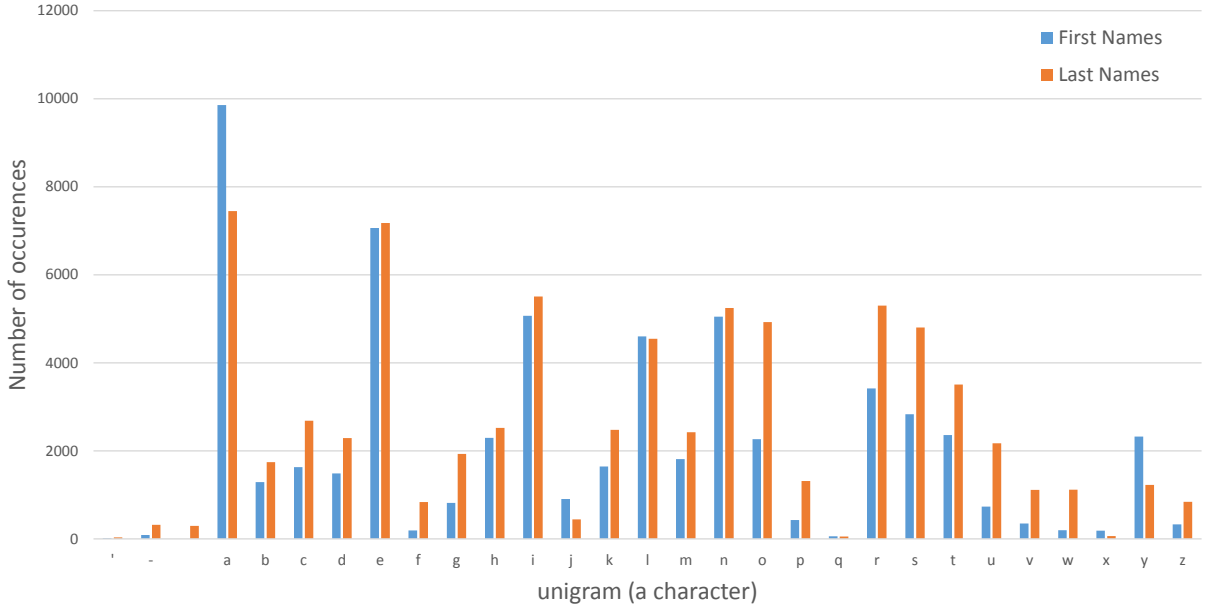
## 6. Conclusions and Future Work

In this work we evaluated the accuracy, speed and security of selected string metrics that support approximate matching for querying and joining databases based on encrypted names. An advanced benchmark generation tool, "GeCo", was used to produce sample datasets with records containing common mistakes in name spelling such as typographical errors, optical recognition errors, and phonetic errors. The performance of several string metrics that support approximate matching on encrypted data (four variations of $q$-grams based techniques and one technique based on encodings produced by the Soundex algorithm), was compared against commonly used string metrics, such as Jaro and Levenshtein, employed on unencrypted data.

Joining databases based on Soundex encodings did not prove to be a feasible option since it failed to find a correct match for more than 50% of records when the name in a corrupted record contained one error, and for almost 75% when corrupted records could contain two errors in a single name. $Q$-grams based techniques seem to be viable option for joining databases on encrypted names. While their performance in terms of precision is slightly worse than the performance of metrics such as Jaro or Levenshtein on unencrypted data, this can be easily dealt with by adjusting the threshold value that determines when two $q$-grams are likely to correspond to the same name. In order to achieve acceptable privacy while maintaining an reasonable level of performance, $q$ should be set to at least three.

In future work we plan to extend the range of attribute types that can be used to perform record linkage. In this study we focused on linking records based only on an individual's first and last name. However other types of attributes, such as numeric or categorical ones, can also carry PII. We want to be able to integrate those kind of attributes into private record linkage queries.

Fig. 10. First and last names distribution based on the length of the name.




## References


[1] Abdullah Al Hasib and Abul Ahsan Md Mahmudul Haque. A comparative study of the performance and security issues of aes and rsa cryptography. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 2, pages 505–510. IEEE, 2008.

[2] Peter Christen. A comparison of personal name matching: Techniques and practical issues. In *Proceedings of the Sixth International Conference on Data Mining Workshops*, pages 290–294. IEEE, 2006.

[3] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

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

[5] Kevin Dreßler and Axel-Cyrille Ngonga Ngomo. Time-efficient execution of bounded jaro-winkler distances. In *Proceedings of the 9th International Conference on Ontology Matching*, volume 1317, pages 37–48. CEUR-WS.org, 2014.

[6] Mark Giereth. On partial encryption of rdf-graphs. In *Proceedings of the International Semantic Web Conference*, pages 308–322. Springer, 2005.

[7] Pawel Grzebala and Michelle Cheatham. Private record linkage: A comparison of selected techniques for name matching (in press). In *Proceedings of the 13th European Semantic Web Conference*, 2016.

[8] Heikki Keskustalo, Ari Pirkola, Kari Visala, Erkka Leppänen, and Kalervo Järvelin. Non-adjacent digrams improve matching of cross-lingual spelling variants. In *String Processing and Information Retrieval*, pages 252–265. Springer, 2003.

[9] Juan C Muñoz, Gabriel Tamura, Norha M Villegas, and Hausi A Müller. Surprise: user-controlled granular privacy and security for personal data in smartercontext. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, pages 131–145. IBM Corporation, 2012.

[10] Lawrence Philips. Hanging on the metaphone. *Computer Language*, 7(12), 1990.

[11] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. Privacy-preserving record linkage using bloom filters. *BMC medical informatics and decision making*, 9(1):41, 2009.

[12] Chakkrit Snae. A comparison and analysis of name matching algorithms. *International Journal of Applied Science, Engineering and Technology*, 4(1):252–257, 2007.

[13] Khoi-Nguyen Tran, Dinusha Vatsalan, and Peter Christen. Geco: an online personal data generator and corruptor. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2473–2476. ACM, 2013.

[14] U.S. Census Bureau. Frequently occurring surnames from census 1990 - names files. `http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html`. Accessed: 2016-03-18.

[15] Dinusha Vatsalan, Peter Christen, and Vassilios S Verykios. An efficient two-party protocol for approximate matching in private record linkage. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pages 125–136. Australian Computer Society, Inc., 2011.

[16] Dinusha Vatsalan, Peter Christen, and Vassilios S Verykios. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969, 2013.

[17] Mohamed Yakout, Mikhail J Atallah, and Ahmed Elmagarmid. Efficient private record linkage. In *Proceedings of the 25th International Conference on Data Engineering*, pages 1283–1286. IEEE, 2009.