Blind Data Linkage using n-gram Similarity Comparisons

Tim Churches[1] and Peter Christen[2]

[1] Centre for Epidemiology and Research, New South Wales Department of Health, Locked Mail Bag 961, North Sydney NSW 2059, Australia, tchur@doh.health.nsw.gov.au
[2] Department of Computer Science, Australian National University, Canberra ACT 0200, Australia, peter.christen@anu.edu.au

Abstract. Integrating or linking data from different sources increasingly becomes an important task in the preprocessing stage of many data mining projects. The aim of such linkages is to merge all records relating to the same entity, such as a patient or a customer. If no common unique entity identifiers (keys) are available in all data sources, the linkage needs to be performed using the available identifying attributes, like names and addresses. Data confidentiality often limits or even prohibits successful data linkage, as either no consent can be gained (for example in biomedical studies) or the data holders are not willing to openly provide their data. We present methods for confidential data linkage based on hash encoding, public key encryption and n-gram similarity comparison techniques, and show how *blind data linkage* can be performed.

Keywords: privacy preserving data mining, hash encoding, data matching, public key infrastructure, n-gram indexing.

1 Introduction

The ability to find matches between confidential data items held in two (or more) separate databases is an increasingly common requirement for many applications in data processing, analysis and mining.

A medical research project, for example, may need to determine which individuals, whose identities are recorded in a population-based register of people suffering from hepatitis C infection, also appear in a separate population-based register of cases of liver cancer. Clearly such linkage between these two databases involves the invasion of privacy of the individuals whose details are stored in these databases. It is usually infeasible to obtain consent from each individual identified in each of the register databases – instead one or more ethics committees or institutional review boards provide consent for the linkage of the two databases on the behalf of the individuals involved, after weighing the public good which is expected to accrue from the research against the unavoidable invasion of individual privacy which the research involves.

Traditionally the linkage of records requires that identified data on every registered individual be copied from one of the databases to the other, or from both databases to a third party (often the researchers or their proxy) [8]. The invasion of privacy which this entails could be minimised if there were some method of determining which records in the two databases matched, or were likely to match on more detailed comparison, without either database revealing any identifying information to each other or to a third party. If this were possible, then the researchers only need to be given access to a small subset of records from each of the databases, and it may even be feasible to obtain direct consent from the individuals concerned.

Similarly, consider two research databases of genetic sequences of DNA, each of which are commercially valuable in their own right. The owners or custodians of these two databases may wish to determine whether their databases contain any sequences in common before deciding to collaborate and share their data, without having to first reveal the contents of their database to the other party, or to a third party.

In both cases, comparisons need to be made between sequences of symbols – alphanumeric characters like names and addresses in the first example, nucleic bases in the second – in order to measure their similarity, without revealing what those sequences are.

This paper describes methods for achieving this goal by using hash encoding, n-gram similarity comparison techniques and public key infrastructure. After an overview of related work in the following Section 2, in Section 3 we describe two methods of how to perform such confidential comparisons, and in Section 4 these methods are extended to *blind data linkage* of two data sets. A discussion is then given in Section 5 before conclusions and an outlook to future work is presented in Section 6.

2 Related Work

Berman [1] has described a *threshold protocol* designed to address the following problem (note that the names *Alice*, *Bob* and *Carol* – as traditionally used in the cryptography literature – do not necessarily represent humans, but may represent organisations or information systems):

> "Bob has a file containing the medical records of millions of patients. Alice has secret software that can annotate Bob's file, enhancing its value many fold. Alice won't give Bob her secret algorithm, but is willing to demonstrate her algorithm if Bob gives her his database. Bob won't give Alice the database, but he can give her little snippets of the database containing insufficient information for her to infer patient identities." [1]

As stated in this problem definition, the threshold protocol is only suitable for cases in which: (a) it is possible to reliably extract non-confidential fragments from the database in question, and (b) those non-confidential fragments are of interest or utility to the other party. These conditions precludes its application in the situations outlined in the previous section.

A number of protocols which allow for the all-or-nothing disclosure of secrets have appeared in the cryptology literature [3, 4, 11], but to our best of knowledge no-one has described a method of using these protocols for the comparison of strings or sequences. The methods described in this paper do not make use of these all-or-nothing disclosure protocols.

Anonymous data linkage based on SHA [13] hashed identifiers are used in Switzerland [2] and France [10]. In the French system spelling transformations are performed before the identifiers are hashed (with an added *pad* to prevent dictionary attacks), and probabilistic linkage techniques are then applied based on exact matches only. The Swiss system works in a similar way and is described in more details in Section 3.1.

Schadow et. al. [12] describe a system for privacy preserving distributed queries. They state that "large accumulations of patient data are commonly viewed as threatening patients' privacy, and therefore may not be scalable." Having distributed databases and performing confidential data linkage therefore becomes important for successful biomedical research. Their approach consists of a mediator (to which a query poser submits a query), and a large number of distributed data sources. The mediator sends the queries to the data sources and then performs the linkage on the returned results. Patient identifiers are key-hashed (using the HMAC algorithm), and deterministic linkage techniques are applied using four different combinations of identifiers (which include first and last names, dates of birth, and social security numbers). While fairly good sensitivity and specificity are reported, this work does not address the important issue of (typographical) errors in the patient identifiers (which occur in most real world databases).

3 Methods

Alice holds a database, **A**, which contains one or more attributes (columns, variables), denoted **A.a**, **A.b** and so on, containing confidential strings (like names and addresses) or sequences (of perhaps nucleotide bases or amino acids). The need for confidentiality may arise from the fact that the values in **A.a** identify individuals, or because the information has some commercial value (such as the base sequence for particular genes of known function). Bob holds a similar but quite separate database, **B**, also containing one or more confidential columns, **B.a**, **B.b** and so on.

Alice and Bob wish to determine whether any of the values in **A.a** match any of the values in **B.a** without revealing to each other or to any other party what the actual values in **A.a** and **B.a** are.

The problem is simple when "matching" is defined as exact equality of the pair of strings or sequences being compared, but becomes more complicated if the strings contain errors (for example typographical variations in names). The following section presents the protocol for the exact matching and Section 3.2 deals with the case of inexact matching using n-grams.

All transfers of data between parties are assumed to be secured and authenticated using a public key infrastructure (PKI) [13] to sign and encrypt all messages.

3.1 Exact Matching of Secret Strings or Sequences

Carol is a third party who is trusted by Alice and Bob to (a) perform the processing described in this protocol, and (b) not to divulge the data which she receives from Alice and from Bob to any other party, including Alice and Bob. We will refer to this protocol as *Protocol I.*

1. Alice and Bob mutually agree on a secret random key, K_{AB}, which they share only with each other. They also agree on a secure one-way hashing function k_{owh} (e.g. MD5 or SHA [13]), and on a standard protocol for preprocessing strings to render them into a standard form (such as converting all characters to lower case, removal or substitution of punctuation and extraneous whitespace, and so on). They also agree on a standard method of transforming values using K_{AB} – this might be as simple as concatenation of a string representation of K_{AB} to a string representation of the value to be transformed, or it might be symmetrical encryption of the value to be transformed using an agreed algorithm with K_{AB} as the encryption key.
2. Using K_{AB}, Alice transforms each of the secret strings in **A.a** in the agreed manner. Alice then further encodes the transformed values using the secure one-way hash function k_{owh}. The resulting set of hashes, H_A, is sent to Carol. These hashes carry no recoverable information about the values from which they were derived (the hash functions are one-way). The use of K_{AB} to transform the "raw" values of **A.a** before hashing renders a dictionary attack by Carol infeasible (a dictionary attack is one in which a large number of possible values for **A.a** are hashed by Carol to discover whether the hashed value matches any of those contained in H_A).
3. Bob does the same with the values of **B.a**, and also sends the resulting set of hashes, H_B, to Carol.
4. Carol finds the intersection of H_A and H_B, which we will denote H_{AB}.

What happens next depends on the overall intent. Carol can report the number of hashes in common (that is $|H_{AB}|$) to Alice and Bob, who may then enter into further negotiation between themselves before revealing further information to each other about those matching values (or records). Carol may just return H_{AB} to both Alice and Bob, in which case Alice immediately acquires information about which of her values match Bob's values, and vice-versa. Alternatively, Carol may pass the list of intersecting hashed values to another party (say, a researcher) who may make further use of the information.

An obvious limitation of this protocol is that only strings which are exactly equal are matched. Even a single character difference between the strings will result in different hash values (in which, if the one-way hash function is a good one, a majority of bits will be different).

One method of overcoming this problem is to reduce the dimensionality of the secret strings in **A.a** and **B.a** before they are hashed, using, for example, a phonetic encoding function such as Soundex [9]. This renders the match which Carol performs blindly on the hashed values somewhat more robust, at the expense of a greater number of false matches (homonym errors). A minor variation of this technique is used in Switzerland and France to permit anonymous linkage of patient records [2, 10]. A compound "linkage key" is formed from the Soundex (or other phonetic encoding) code of the patient's surname, the Soundex of their given name, their date of birth and their sex, and this value is hashed (using SHA) and the hash value is then encrypted with a secret key (to prevent dictionary attacks against it).

However, Soundex and other phonetic transformations are not perfect – in particular they are not robust to errors in the initial character, and to truncation differences. For example, "Christopher", "Christine", "Christen" and "Cristina" all have a Soundex code of C623 whereas "Chris" has a code of C620; and "Kristine" has a code of K623. In some cases it may be possible to mitigate these problems by also hashing known alternatives, such as "Liz" and "Beth" for "Elizabeth", in an attempt to get a match, but the generality of this approach is limited and the rate of false matches will inevitably increase.

Ideally, a protocol is required which permits the blind calculation by Carol, and/or other trusted third parties, of a more general and robust measure of similarity between the pairs of secret strings. A description of such a protocol, which implements blind n-gram comparisons, follows.

3.2 n-gram Similarity Comparison of Secret Strings or Sequences

In this example, bigrams (2-grams, $n = 2$) are used, but the extension to trigrams ($n = 3$) and other n-grams is direct. We will refer to the protocol described below as *Protocol II.* Although the primary aim of this protocol is to permit Carol to produce a set of similarity scores for pairs of Alice's and Bob's secret strings without Carol being able to determine the nature of those strings, a secondary aim is to prevent information about Bob's secret strings leaking to Alice, and vice versa.

Protocol II assumes that Carol is trusted by Alice and Bob to (a) adhere to the protocol, (b) not reveal information to other parties except where permitted by the protocol, and (c) not try to determine the values of Alice's or Bob's source strings using cryptologic techniques. The effects of violations of these assumptions are considered in Section 3.3. There is no assumption that Alice trusts Bob or vice versa. Note that Alice and Bob do need to share with each other meta-data about the nature of the information contained in their databases – in order to decide which columns/attributes can be validly compared – but they do not need to share the actual values of those columns, nor summary measures (such as frequency counts) derived from those values.

1. As in Protocol I, Alice and Bob mutually agree on (a) a secret random key, K_{AB}, (b) a secure one-way hashing function k_{owh}, (c) a standard protocol for preprocessing strings, and (d) a standard method of transforming and encoding values using K_{AB} and k_{owh}.
2. Alice computes a sorted list of bigrams for each preprocessed (as described in step 1 above) value in the column **A.a** – for example if a value of **A.a** is "Peter" then the sorted list of bigrams is ("er","et","pe","te"). Note that duplicate bigrams are removed, so each bigram is unique in each list. Alice next calculates all possible sub-lists of all lengths greater than zero for each bigram list – in other words, the

power-set of bigrams minus the empty set. For the example given above, Alice computes bigram sub-lists ranging from length 1 to 4.

("er"), ("et"), ("pe"), ("te"),
("er","et"), ("er","pe"), ("er","te"), ("et","pe"), ("et","te"), ("pe","te"),
("er","et","pe"), ("er","et","te"), ("er","pe","te"), ("et","pe","te"),
("er","et","pe","te")

Assuming a bigram list contains b bigrams, the resulting number of sub-lists is $2^b - 1$. Alice then transforms each of the calculated bigram sub-lists using K_{AB} and hashes the result using the agreed one-way hash function k_{owh}. These hashes are stored in column **A.a_hash_bigr_perm**. Alice also creates an encrypted version of the record identifier (key) for the string from which each value in **A.a_hash_bigr_perm** was derived – she stores this in **A.encrypt_rec_key**. She also places the length (that is, number of bigrams) of each **A.a_hash_bigr_perm** in a column called **A.a_hash_bigr_perm_len**, and the length (that is, the number of bigrams) of each original secret string in **A.a**, in a column **A.a_len**. Alice then sends the set of quadruplets (**A.a_hash_bigr_perm**, **A.a_hash_bigr_perm_len**, **A.encrypt_rec_key**, **A.a_len**) to Carol. Note that the number of quadruplets is going to be much larger than the number of original records in **A.a**.

3. Bob carries out the same steps as in step 2 with his column **B.a**, and also sends the resulting set of quadruplets to Carol.
4. Carol determines the set intersection of the values of **A.a_hash_bigr_perm** and **B.a_hash_bigr_perm** which she has been sent by Alice and Bob respectively. For each value of **a_hash_bigr_perm** shared by **A** and **B**, for each unique pairing of (**A.encrypt_rec_key**, **B.encrypt_rec_key**), Carol calculates a bigram score

$$bigr\_score = \frac{number\_of\_bigrams\_in\_common}{0.5 \cdot (number\_of\_bigrams\_in\_A.a + number\_of\_bigrams\_in\_B.a)}$$

and selects the maximum bigram score value for each possible unique pairing of (**A.encrypt_rec_key**, **B.encrypt_rec_key**) – that is, the highest score for each pair of strings from **A.a** and **B.a**. Note that a bigram score of 1.0 corresponds to an exact match between two values.

What happens next again depends on the context. Carol may report the number of strings with a bigram score above an agreed threshold to Alice and Bob, who may then negotiate further steps, or Carol may simply report the similarity scores and the encrypted record keys back to Alice and Bob. Alternatively, Carol may send this information to another third party, *David*, who oversees an over-arching *blind data linkage* protocol involving a number of different columns from Alice's and Bob's databases (that is, not just **A.a** and **B.a**, but also **A.b** and **B.b**, **A.c** and **B.c** and so on). This will be described in Section 4 below.

3.3 Defects in Protocol II, and Possible Remedies

There is no doubt that Protocol II meets its design aims of blinding Carol to the values of Alice's and Bob's secret strings, while still permitting her to calculate a measure of the similarity between those strings. The hashes of the transformed bigram permutations which are sent to Carol do not, per se, carry any information about the secret strings from which they were derived. However, Carol also has quite a lot of additional information which she can use as clues or "cribs" (to use the cryptologic term) to help her infer some or all of the values of Alice's and Bob's secret strings. Given sufficient data, Carol may be able to mount a number of statistical attacks on the information hidden in the hashes.

One approach is for Alice and Bob just to trust Carol not to undertake such attacks. This means that Carol must implement mechanisms to protect the data which she processes from misuse by both insiders (Carol's own staff) and external "attackers", with protection against the former being particularly difficult. However, the processing which Carol is required to carry out can be completely automated, and for all but the largest data sets it could run entirely in RAM (with needed disc-based swap space). Thus, all the data handled by Carol could be volatile, and, if implemented on a secure operating system (such as the NSA Security Enhanced Linux[1]), it may be possible to make it very difficult to misappropriate Carol's data or suborn her processing.

Another approach is to hide Alice's and Bob's real data amongst dummy data. For example, Alice might add a large number of dummy records to her database, and keep a list of the identifiers of these records. Bob does the same. Carol is unaware which of the values she is processing are from dummy records, and which are from real records. Ideally, both Alice and Bob will draw the values for these dummy records from a shared pool of values, so that some of them will match, just like the real data. However the proportion of Alice's and Bob's values which match is usually unknown in advance, so it is difficult to ensure that the degree of overlap between Alice's and Bob's dummy values is indistinguishable from the overlap between the real values. Nevertheless, this approach would make it significantly more difficult to obtain useful information from the data supplied to Carol by Alice and Bob.

A third approach, which we will refer to as *Protocol III*, is as follows. Note the use of an additional trusted third party, *David*. This protocol is suitable when Alice and Bob both intend to share with each other the information which matches, but where it is vital that third parties such as Carol cannot infer information about Alice's and Bob's secret strings.

1. As in Protocols I and II, Alice and Bob mutually agree on (a) a secret random key, K_{AB}, (b) a secure one-way hashing function k_{owh}, (c) a standard protocol for preprocessing strings, and (d) a standard method of transforming and encoding values using K_{AB} and k_{owh}.
2. As in Protocol II, Alice creates the quadruplet of columns **A.a_hash_bigr_perm**, **A.a_hash_bigr_perm_len**, **A.encrypt_rec_key** and **A.a_len**. Unlike Protocol II, Alice then sends only the set of hashes in **A.a_hash_bigr_perm** to Carol.
3. Bob carries out the same steps as in step 2 with his column **B.a**, and also sends only the contents of the **B.a_hash_bigr_perm** to Carol.
4. Carol determines the set intersection of the values of **A.a_hash_bigr_perm** and **B.a_hash_bigr_perm** which she has been sent by Alice and Bob respectively. Carol reports back to Alice the hash values in **A.a_hash_bigr_perm** which are shared by **B.a_hash_bigr_perm**, and she reports back to Bob the hash values in **B.a_hash_bigr_perm** which are shared by **A.a_hash_bigr_perm**. In other

[1] `http://www.nsa.gov/selinux/`

words, both Alice and Bob learn which bigram permutations they share with each other. This obviously leaks some information about the nature of Bob's secrets to Alice, and vice versa. However, Carol has much less information on which to base a cryptologic attack on the information she has received from Alice and Bob.

5. Alice now prepares a set of column quadruplets comprising **A.a_hash_bigr_perm**, **A.a_hash_bigr_perm_len**, **A.encrypt_rec_key** and **A.a_len**, but only for those records whose value for **A.a_hash_bigr_perm** appears in the list of hashes returned to Alice by Carol in step 4. Alice sends this set of quadruplets to David.
6. Bob does the same with his columns.
7. David now joins the sets of quadruplets he has received from Alice and Bob on the values of **a_hash_bigr_perm**, and for each joined record, calculates the bigram score. Then, just as Carol did in step 4 of Protocol II, for each unique pairing of (**A.encrypt_rec_key**, **B.encrypt_rec_key**), David determines the maximum bigram score.

Clearly Protocol III can also be combined with the other strategies discussed above for Protocol II, such as hiding information amongst many dummy records, and "hardening" Carol's and David's system to make misuse of the information which they are sent much more difficult.

3.4 Protection through Last-Minute Election of Third Parties

One other strategy which would reduce the risk of misuse of the information sent to Carol (or David) would be to have many Carols and Davids available, all functionally equivalent, and for Alice and Bob to decide on which of these Carols and Davids to use only at the very last moment. This would mean that a potential attacker would need to suborn or compromise a large number of the Carols in order to have a reasonable chance of gaining access to the information provided to one particular Carol by Alice and Bob. The fully automated nature of the processing carried out by Carol and David lends itself to replication on multiple hosts on a network – which fits very neatly with the Grid model of distributed computing [7].

3.5 Dealing with Non-String Data

Identifying data not only consists of strings or sequences, but often also contains numerical, or date and time attributes. The similarity comparison of scalar quantities (absolute or relative *delta* values) can be implemented quite easily. Say Alice has a column **A.age** containing a value 35 (years) and she wants to match it with values held by Bob which are within 1 year of that. Alice just generates three hashed values 34, 35, and 36 of age for this particular record. This principle can be extended to other quantities, including dates and times, provided they are expressed in a suitable format – such as ticks since an epoch as in Unix date/time. For comparisons of dates which take into account typographical errors, a type of modified bigram comparison would be more appropriate, but it is still amenable to the general similarity comparison scheme described above.

4 Blind Data Linkage

So far, we have demonstrated how blind similarity or distance comparisons of strings and scalar quantities can be achieved. How can these be combined into a method for blind data linkage?

The essence of modern data or record linkage techniques [6, 14] is the independent comparison of a number of partial identifiers (data elements) between pairs of records, and the combination of the results of these comparisons into a compound or summary score (called *matching weight*) which is then judged against some criterion (or thresholds) to classify that pair of records as a match (link), a non-match, or as undecided (potential match). Usually the result of the comparison between individual data elements is weighted in some way – in *deterministic* systems these weights are often binary, whereas in *probabilistic* systems the weights are continuous and determined empirically based on the relative reliability of that data element in deciding matches and non-matches [6], and on the relative frequency of the values of that data element [14]. The classification criteria for the summary score are often determined heuristically or statistically, for example using expectation maximisation (EM) techniques [15].

Thus, the first task is to compare each of the partially-identifying data elements and return a similarity score for each pair. We have demonstrated how this can be done blindly – in the following discussion, we will assume the use of Protocol II as described in Section 3.2, and thus refer to Carol as the third party undertaking the blind comparisons. Note that because this comparison is being done by finding the intersection between sets of 128-bit (or longer) hash values, the intersection operation carried out by Carol can be very efficient. Also, the processing is distributed, with Alice and Bob doing much of the work in creating the n-gram permutations and the hashed values thereof to be compared – the task for Carol is quite small, even though, effectively, multiple hashes for every record in **A** are being compared to multiple hashes for every record in **B**.

So, for each of the partially-identifying (or partially-discriminating) data elements, **a**, **b**, ..., ***i***, in their databases **A** and **B**, Alice and Bob dispatch the similarity (or distance) comparison task to different instances of Carol, which we will term Carol$_a$, Carol$_b$, ..., Carol$_i$. Each of these tasks is independent of the others, and should use a different shared secret key K_{AB}. Each instance of Carol sends the results back to another third party which oversees the entire data linkage task between **A** and **B** – we will call this party *Edith*. Thus, Edith accumulates a series of data sets containing comparison values (or similarity scores) **comp_val** from the Carols of the form:

(A.encrypt_rec_key, B.encrypt_rec_key) : *data_item*.comp_val

where ***data_item*** is a column, **a**, **b**, ..., ***i*** shared by **A** and **B**. Note that not every possible combination of the tuple (**A.encrypt_rec_key**, **B.encrypt_rec_key**) will be present in the data sent to Edith by each instance of Carol – only those record pairs for which the comparison value was greater than 0. In other words, they are a sparse matrix of comparison values, with **A.encrypt_rec_key** the row index and **B.encrypt_rec_key** the column index. Edith joins these data sets and forms a sparse matrix where each entry contains all the comparison values **a.comp_val**, **b.comp_val**, ..., ***i*.comp_val** for a record pair, with a ***data_item*.comp_val** taken to be 0 if it is undefined.

It is now a simple matter for Edith to multiply this matrix by a vector of weights (for each data item), and then sum across each row to create a summary *matching weight*, which is compared to some criterion (thresholds) which have been determined heuristically. Where the weights are binary, this is equivalent to *deterministic* data linkage. Alternatively, the matrix of comparisons can be used as input for a number of machine learning classification methods. Where these methods are supervised, a pair of data sets which have already been linked (and for which the match status of each pair of records is not secret) would need to be used as the *gold standard* to train the classifier models.

If the full *Fellegi & Sunter* [6] model of probabilistic data linkage was required, then Alice and Bob would need to communicate the relative frequency of the value of each data item for each record in their databases to Edith (but not to Carol), so Edith could use this information to further weight the similarity comparison matrix. Note that Edith has no information about the length of the values – just the similarity score between pairs of values. Nevertheless, it would be important that Carol and Edith do not share any information other than that described above.

We will assume that Edith arrives at a set of linked records – that is, pairs of (**A.encrypt_rec_key**, **B.encrypt_rec_key**) – by some means. How can a researcher, *Freddy*, be supplied with de-identified (or anonymous) data from Alice and Bob's databases for just these linked records without telling Alice which of her records match Bob's, and vice versa? There is a straightforward solution using public key encryption [13]: Alice assembles the columns required by researcher Freddy for every record in her database, and encrypts each record, but not the **encrypt_rec_key**, with Freddy's public key. Alice send this data to Edith. Bob does the same. Edith now joins and subsets this data using her list of matching records, and forwards the joined subsets to Freddy. Edith is unable to see the data values contained in these records because they have been individually encrypted with Freddy's public key. Freddy decrypts the data for each joined record using his private key. Thus, Freddy obtains just the columns he needs for the linked records only, Edith knows the record keys of the linked records but never sees the data (because it is encrypted with Freddy's key), and Alice and Bob learn nothing.

5 Discussion

Proof-of-concept implementations of the protocols described in this paper have been developed as a series of Python[2] programs. In these implementations, Alice generates the secret random key, K_{AB}, using a random number generator operating over a large number space, then transforms the result into an MD5 hash

[2] `www.python.org`

Table 1. Overhead of bigram similarity comparisons and hash encoding.

	Surnames		Suburb names	
	ACT	NSW	ACT	NSW
Number of records	115,558	2,323,355	115,558	2,323,355
Average string length (characters)	6.52	6.39	7.01	9.28
Average number of bigram sub-lists	210.5	166.2	236.3	2,521.0
Uncoded strings (KBytes)	736	14,500	791	21,047
Exact hashed matching (KBytes)	1,806	36,302	1,806	36,302
Bigram hashed matching (KBytes)	380,101	6,032,223	426,695	91,518,954
Ratio exact hash / uncoded	2.45	2.50	2.28	1.72
Ratio bigram hash / uncoded	516.7	416.0	539.3	4,348.3

value and transmits this value to Bob. The values in **A.a** and **B.a** (example surnames) are transformed by simply concatenating K_{AB} to a string representation of the values, followed by applying the one-way hash function encoding, i.e. $k_{owh}(\mathbf{A.a} + K_{AB})$.

Compared to transferring the raw identifying strings between the parties of a data linkage system, bigram similarity comparisons and hash encoding require a much larger volume of data to be transfered. We performed simulations using surnames and suburb names taken from two Australian phone directories, and calculated the average lengths of the surname and suburb name strings, the resulting number of bigram sub-lists, as well as the overhead resulting from using bigrams and hash encoding. We assumed MD5 hash encoding which uses 128-bit (16 byte) hash codes.

Table 1 shows the amount of data (in Kilobytes) transfered and the resulting overhead. The overhead can be reduced somewhat by only calculating and transferring bigram sub-lists of certain lengths (for example, bigram sub-lists of length 1 are very unlikely part of a successful match of longer strings or sequences).

6 Conclusions and Future Work

In this paper we have presented methods for blind fuzzy linkage of records using hash encoding, public key encryption and n-gram similarity comparison techniques. Proof-of-concept implementations have demonstrated the feasibility of our approach, albeit at the expense of very high data transmission overheads. On modern high-bandwidth research networks, we do not believe this is a fundamental problem. We are planning to include these blind data linkage techniques into our open source data linkage system *Febrl* [5], and to perform large scale linkage experiments using real world data sets.

References

1. Berman, J.J.: Threshold protocol for the exchange of confidential medical data. BMC Medical Research Methodology, Nov. 2002, 2:12.

2. Borst, F., Allaert, F.A. and Quantin, c.: The Swiss Solution for Anonymous Chaining Patient Files. MEDINFO 2001.
3. Brassard, G., Cr'epeau, C. and Robert, J.M.: All-or-Nothing Disclosure of Secrets. Advances in Cryptology (Crypto'86), Lecture Notes in Computer Science, no. 263, Springer Verlag, 1987, pp. 234-238.
4. Brassard, G., Chaum, D. and Cr'epeau, C.: Minimum Disclosure Proofs of Knowledge. Journal of Computer and System Sciences (JCSS), vol. 37, no. 2, 1988.
5. Freely extensible biomedical record linkage (Febrl) project web page, URL: `http://sourceforge.net/projects/febrl`
6. Fellegi, I. and Sunter, A.: A Theory for Record Linkage. Journal of the American Statistical Society, 1969.
7. Foster, I. and Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann; 1st edition, Nov. 1998.
8. Kelman, C.W., Bass, A.J. and Holman, C.D.J.: Research use of linked health data – a best practice protocol. Australian and New Zealand Journal of Public Health, 26:3, 2002, pp. 251-255.
9. Lait, A.J. and Randell, B.: An Assessment of Name Matching Algorithms, Technical Report, Department of Computing Science, University of Newcastle upon Tyne, UK 1993.
10. Quantin, C., Bouzelat, H., Allaert, F.A.A., Benhamiche, A.M., Faivre, J. and Dusserre, L.: How to ensure data quality of an epidemiological follow-up: Quality assessment of an anonymous record linkage procedure. Intl. Journal of Medical Informatics, vol. 49, pp. 117-122, 1998.
11. Salomaa, A. and Santean, L.: Secret selling of secrets with many buyers. EATCS Bulletin 42, 178–186, 1990.
12. Schadow, G., Grannis, S.J. and McDonald, C.J.: Discussion paper: privacy-preserving distributed queries for a clinical case research network. Proceedings of the IEEE international conference on privacy, security and data mining, Maebashi City, Japan, 2002, pp. 55-65.
13. Schneider, B.: Applied Cryptography. John Wiley & Sons, second edition, 1996.
14. Winkler, W.E.: The State of Record Linkage and Current Research Problems. Research Report RR99/03, US Bureau of the Census, 1999.
15. Winkler, W.E.: Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. Research Report RR00/05, US Bureau of the Census, 2000.