Tree Based Scalable Indexing for Multi-Party Privacy-Preserving Record Linkage

Thilina Ranbaduge, Peter Christen, and Dinusha Vatsalan

Research School of Computer Science, College of Engineering and Computer Science,
The Australian National University,
Canberra ACT 0200, Australia,
Email: {thilina.ranbaduge, peter.christen, dinusha.vatsalan}@anu.edu.au

Abstract

Recently, the linking of multiple databases to identify common sets of records has gained increasing recognition in application areas such as banking, health, insurance, etc. Often the databases to be linked contain sensitive information, where the owners of the databases do not want to share any details with any other party due to privacy concerns. The linkage of records in different databases without revealing their actual values is an emerging research discipline known as privacy-preserving record linkage. Comparison of records in multiple databases requires significant time and computational resources to produce the resulting matching sets of records. At the same time, preserving the privacy of the data is becoming more problematic with the increase of database sizes.

We propose a novel indexing (blocking) approach for privacy-preserving record linkage between multiple (more than two) parties. Our approach is based on Bloom filters to encode attribute values into bit vectors. The Bloom filters are used to construct a single-bit tree, where the encoded records are arranged into different blocks. The approach requires the parties to only participate in a secure summation protocol to find the best bits to construct the trees in a balanced manner. Leaf nodes of the trees will contain the blocks with encoded records. These blocks can finally be compared using private comparison and classification techniques to determine the similar record sets in different databases. Experiments conducted with datasets of sizes up-to one million records show that our protocol is scalable with both the size of the datasets and the number of parties, while providing better blocking quality and privacy than a phonetic based indexing approach.

Keywords: Multi-party protocol, privacy technologies, scalability, Bloom filter, single-bit tree, secure summation protocol.

Funded by the Australian Research Council under Discovery Project DP130101801.

1 Introduction

Many organizations, including businesses, government agencies and research organizations, are collecting vast amounts of data that are stored, processed and analyzed for the improvement of their works. These data often contain millions of records. As the size of data is continuously increasing, developing techniques for efficient processing, analyzing and mining has gained much recognition in both academia and industry. Many application domains require information from multiple sources to be integrated and combined in order to improve data quality and to facilitate further analysis of the data. The process of matching records that relate to the same entities from different data sources is known as 'record linkage' (Fellegi & Sunter 1969).

In most occasions the linkage of records from multiple sources requires much computational resources for processing. The process becomes even more challenging when the records contain personal information. Organizations commonly do not want their sensitive information to be linked with other data sources due to growing privacy and confidentiality concerns. The research paradigm of finding records in multiple data sources that relate to the same entity without revealing personal information is known as 'privacy-preserving record linkage' (PPRL), 'blind data linkage' or 'private record linkage' (Al-Lawati et al. 2005, Churches & Christen 2004, Karakasidis & Verykios 2011, Yakout et al. 2012).

On occasions where unique identifiers for entities are available across all the databases to be linked, a simple database join would be trivial for the purpose of identifying the matching pairs of records. However, in most cases finding such a common identifier in all databases would not be possible. A possibility to overcome this issue is to use quasi-identifiers (QID) such as first name, last name, address details, age, etc. (Hawashin et al. 2011). This will allow to accurately link records, but it will reveal personal information to other parties involved in the linking process. In order to cope with privacy issues the values in those identifiers need to be somehow encoded.

Bloom filters, which were proposed by Bloom (1970), have widely been used for encoding of records in record linkage approaches. A Bloom filter is a bit array which can hold 1's and 0's according to a bit pattern where initially all bit positions are set to 0's. Records can be hashed by using hash encoding algorithms to generate bit patterns for the records. These bit pattens can then be included in a Bloom filter by setting the relevant bit positions to 1's. Several approaches (Lai et al. 2006, Schnell et al. 2009, de Vries et al. 2011, Vatsalan & Christen 2012) have used Bloom filters for matching record sets.

While preserving the privacy of the QID values, which are used for the linking process, one major challenge is to cope with scalability. Many different indexing or blocking approaches have been introduced to compare databases (Christen 2012*b*), because the naive approach of comparing all pairs of records is not feasible when the databases are large. An indexing mechanism reduces the large number of potential comparisons by removing as many record pairs as possible that correspond to non-matches. This decreases the amount of computational efforts required for the comparison of larger databases.

The aim of this paper is to propose a new indexing mechanism for multi-party PPRL that can provide better scalability, blocking quality, and privacy, which are important factors for any practical PPRL application. We introduce a tree based approach which uses a secure summation protocol to generate blocks. The paper also presents an empirical evaluation of the proposed approach with regard to scalability, blocking quality, and privacy.

The remainder of the paper is structured as follows. In the following section, we provide an overview of related work in PPRL. In Section 3 we describe the current problem of indexing with multiple parties. In Section 4 we provide a detailed description of our protocol, and in Section 5 we analyze our protocol with regard to complexity, quality, and privacy. In Section 6 we validate these analyses through an experimental study. Finally we summarize our findings and discuss future research directions in Section 7.

2 Related Work

Recently, a variety of indexing or blocking approaches have been developed for reducing candidate record pairs that need to be compared in record linkage. In the survey by Christen (2012*b*) a comprehensive review about existing indexing mechanisms is provided. Some of the developed approaches have been adapted for PPRL based on existing indexing techniques, such as standard blocking, mapping based blocking, clustering, sampling, and locality sensitive hash functions. In our research we mainly focus on developing a blocking mechanism that is scalable to large databases as well as with the number of parties while providing blocking quality and privacy, which are the trade-offs of any indexing step in PPRL.

A protocol proposed by Song et al. (2000) tried to address the problem of approximate matching by calculating enciphered permutations for private approximate record matching. Their suggested approach becomes impractical since predicting all possible permutations is impossible in real-world applications. A two-party protocol, which does not require a trusted third party to perform the linking as in three-party protocols, is suggested by Atallah et al. (2003). This approach allows the parties to compute the edit distance between strings without exchanging these strings, but this protocol is impractical due to the large amount of necessary communication required to compute the distances. Ravikumar et al. (2004) used a secure set intersection protocol for PPRL that requires extensive computations, which makes the protocol impractical for large datasets.

A blindfolded multi-party approach was suggested by Churches & Christen (2004), which uses q-gram hash digests to achieve approximate private linkage. Their approach is computationally costly, because of the generation of power sets of the q-grams of record values. Al-Lawati et al. (2005) introduced three blocking mechanisms for three-party protocols which require a trusted third party to perform the linkage. Their work used hash signatures for comparison of records. Their methods provide a trade-off between privacy and computational and communication cost.

Another three-party protocol to provide privacy for both data and schema matching without revealing any information was presented by Scannapieco et al. (2007). It uses a greedy heuristic re-sampling method for arranging records into blocks. However, their experimental results indicate that the linkage quality is affected by this greedy heuristic re-sampling method. Inan et al. (2010) suggested an approach based on anonymization using a cryptography technique to solve the PPRL problem. Their blocking step used value generalization hierarchies and secure multi-party computation (SMC) based matching (Yao 1986) by using a cryptographic technique, which is computationally expensive to perform.

Bloom filters are used commonly in the PPRL context, due to their capability for computing similarities. Various approaches have been suggested for similarity calculation in PPRL by using Bloom filters (Lai et al. 2006, Schnell et al. 2009, Durham 2012, Vatsalan & Christen 2012, Bachteler et al. 2013).

A multi-party approach was proposed by Lai et al. (2006) that uses Bloom filters to securely transfer data between multiple parties for private set intersection. All the records are encoded into Bloom filters and segmented according to the number of parties involved in the protocol. These segments are shared among the parties, where each party will perform a logical conjunction (and) on segments. Each party compares its own full Bloom filter with the segments of the other parties for matches. Their evaluated results showed that the false positive rate increases with the number of parties involved in the communication, where none of the parties will be able to guess the existence of a given record correctly.

A three-party approach was introduced by Schnell et al. (2009), which performs approximate matching of records by using Bloom filters. The string values in the attributes are converted into sets of q-grams which are then mapped into a Bloom filter using a double hashing mechanism. Durham (2012) proposed a three-party framework for PPRL using Bloom filters. She suggested record-level Bloom filters for encoding all attribute values of a record into a single Bloom filter. Locality sensitive hashing (LSH) functions are used to reduce the computational complexity of the private blocking.

An iterative two-party protocol was proposed by Vatsalan & Christen (2012), which reveals selected bits in the Bloom filters between two database owners. The approach classifies record pairs into matches and non-matches in an iterative way to reduce the number of pairs with unknown match status at each iteration without compromising privacy.

Kristensen et al. (2010) used Bloom filters to efficiently find similar chemical fingerprints in a database based on a user defined similarity threshold value. Each chemical fingerprint was represented by a Bloom filter. For rapid screening of fingerprints among the set of Bloom filters, they introduced a novel tree data structure known as multi-bit tree. All fingerprints are arranged in a binary tree data structure according to the value in selected bit positions. These bit positions are selected to keep the tree structure as balanced as possible. According to the experiments conducted by the authors, it was noted that the performance of the queries increased with the use of this tree data structure. The tree reduced the amount of comparison calculations and computationally scaled linearly when the size of the datasets was increasing.

Table 1: Number of candidate record sets generated with multiple parties for different sizes of datasets and blocks.

	Number of parties			
Data set / Block size	3	5	7	10
10,000 / 10	10^6	10^8	10^{10}	10^{13}
10,000 / 100	10^8	10^{12}	10^{16}	10^{22}
10,000 / 1,000	10^{10}	10^{16}	10^{22}	10^{31}
100,000 / 10	10^7	10^9	10^{11}	10^{14}
100,000 / 100	10^9	10^{13}	10^{17}	10^{23}
100,000 / 1,000	10^{11}	10^{17}	10^{23}	10^{32}
1,000,000 / 10	10^8	10^{10}	10^{12}	10^{15}
1,000,000 / 100	10^{10}	10^{14}	10^{18}	10^{24}
1,000,000 / 1,000	10^{12}	10^{18}	10^{24}	10^{33}

The concept of multi-bit trees was further extended by Bachteler et al. (2013) as a new blocking method for record linkage. In their approach they used Bloom filters to hold the data. The string values in the attributes are first converted into sets of q-grams, which are then mapped into a Bloom filter using a double hashing mechanism. The generated Bloom filters are then partitioned into separate bins according to the number of bits set to 1. All the Bloom filters in each bin are stored in a multi-bit data structure. A given Bloom filter will be queried against all Bloom filters that are stored in the multi-bit tree and at each node the similarity is calculated to find matches. According to the experiments conducted it was noted that the proposed blocking approach is performing better than existing blocking methods such as standard blocking, canopy clustering and sorted neighborhood approaches, while scaling linearly in terms of total running time for construction and querying of a multi-bit tree.

3 Problem Statement

As mentioned in Section 1, generating the candidate record sets for multiple databases becomes computationally expensive when the number of parties is increasing. In such situations, methods or techniques to reduce the comparison space are needed. In the record linkage process these methods are referred to as blocking or indexing methods (Christen 2012*b*). Such methods identify reduced sets of candidate records for comparison and classification, by keeping true matching records in sets of candidate records while removing as many of the true non-matching record sets as possible.

When the number of parties is increasing, more sophisticated indexing mechanisms are required. A larger number of blocks with a small number of records, or a smaller set of blocks with a large number of records, will still require more comparisons. This problem is highlighted in Table 1 to illustrate the number of candidate record sets generated for different number of parties with various datasets and block sizes (by assuming all blocks have the same size).

Table 1 shows how the number of generated candidate record sets grows exponentially with the number of parties involved in a multi-party protocol, and how even with very small sized blocks (e.g. 10 records per block per party) the number of candidate record sets becomes prohibitively large to be practically feasible.

One major problem in currently available indexing solutions is that not enough control is available over the block sizes. Generating different sizes of large number of blocks makes the comparison step even more problematic and requires more computational time. To overcome this problem, in our protocol we provide a parameterized solution, where the user can control the size of the blocks that will be generated. The protocol is based on an indexing mechanism that can generate blocks of records in a balanced tree data structure. Each participating party will have a similar tree data structure constructed holding blocks of records on leaf nodes. The construction of the tree is done in a secure manner without exchanging any information about the records that are held by each party.

4 Tree Based Scalable Indexing for PPRL

In this section we provide details on how the proposed indexing mechanism works. We highlight how attribute values are encoded into Bloom filters and how they are used to construct the tree data structure to hold the blocks. First we will explain the details of the building blocks that are needed for the construction of the index.

4.1 Building Blocks

4.1.1 Bloom Filters

Bloom filters are data structures proposed by Bloom (1970) for checking element membership in any given set (Broder & Mitzenmacher 2004). A Bloom filter is a bit vector of length m, where initially all the bits are set to 0. In order to map an element into the domain between 0 and $m-1$ of the Bloom filter, k independent hash functions $h_1, h_2, \ldots, h_k$ are used. Furthermore, to store n elements of the set $S = \{s_1, s_2, \ldots, s_n\}$ into the Bloom filter, each element $s_i \in S$ is encoded using the k hash functions and all bits having index positions $h_j(s_i)$ for $1 \leq j \leq k$ are set to 1.

4.1.2 Q-grams

A q-gram (also known as n-gram) is a character substring of length q in a string (Christen 2012*a*). Often string values are prefixed and suffixed, which is also known as padding, with a special character of length $q-1$ before they are converted into q-grams (Bachteler et al. 2013). This padding character helps to emphasize the first and last characters of a string. A q-gram of length 2 is known as a *bigram* or *digram* and a q-gram of length 3 is known as a *trigram*. A string s of length c contains $l = c - q + 1$ q-grams (Christen 2012*a*). For an example, the string "THILINA" can be padded with character '_' and the resulting *bigram* ($q = 2$) set is $\{\_T, TH, HI, IL, LI, IN, NA, A\_\}$.

In our approach we used q-gram sets of the QIDs to convert these QIDs into Bloom filters. First, the selected QID values of a given record are converted into a q-gram set. Then each q-gram set is stored in a Bloom filter by using k hash functions. This is repeated for all records in the dataset. Figure 1 illustrates the transformation of a record's QID value into a Bloom filter.

4.1.3 Optimal Parameter Settings for Bloom Filters

A crucial aspect that affects all three challenges (quality, scalability and privacy) of PPRL approaches that are based on Bloom filters is the parameter settings used to generate the Bloom filters. In this section we describe our choice of parameters, which follows

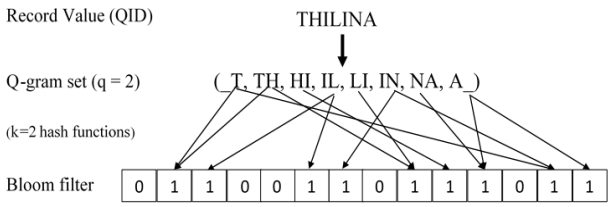

Figure 1: Mapping of a string value into a Bloom filter of $m = 14$ bits by using $k = 2$ hash functions.

earlier Bloom filter based PPRL approaches (Schnell et al. 2009, Durham 2012, Durham et al. 2013, Vatsalan & Christen 2012).

For a given Bloom filter length, m, and number of elements (e.g. q-grams) to be inserted into the Bloom filter, n, the optimal number of hash functions, k, that minimizes the false positive rate, r, can be calculated as (Mitzenmacher & Upfal 2005)

$$k = \frac{m}{n} ln(2), \tag{1}$$

leading to a false positive rate of

$$r = \left(\frac{1}{2^{ln(2)}} \right)^{m/n}. \tag{2}$$

We can calculate the value for n by analyzing the content of a dataset to be used for a PPRL project by calculating the average number of q-grams that are generated from a record (i.e. we convert attribute values into q-grams as described above and count how many q-grams are generated on average for a record).

The value of m determines how much memory and communication will be required in our PPRL protocol. For a given m, we can calculate k based on n as calculated from the datasets. For a certain dataset and n, the larger m the larger k will be, with larger values of k requiring more computation as more hash values need to be calculated and mapped into a Bloom filter for each record.

While k and m determine the computational aspects of our approach, quality and privacy will be determined by the false positive rate r. A higher value for r will mean a larger number of false matches (i.e. a set of non-matching records classified to correspond to the same entity), and thus lower quality. At the same time, a higher false positive rate r will also mean improved privacy, as false positives mean an adversary cannot be absolutely sure that a certain Bloom filter corresponds to a certain record (Schnell et al. 2009, Durham 2012, Vatsalan & Christen 2012).

It was proven (Mitzenmacher & Upfal 2005) that a Bloom filter should ideally have half its bits set to 1 (i.e. 50% filled) to achieve the lowest possible false positive probability for given values of n, m and k. Equations 1 and 2 in fact lead to a probability that a bit in a Bloom filter is set to 1 as $p = e^{-kn/m} = 0.5$ (Mitzenmacher & Upfal 2005). For PPRL this is important, because the bit patterns and their frequencies in a set of Bloom filters can be exploited by a cryptanalysis attack (Kuzu et al. 2013). Such an attack exploits the fact that Bloom filters that are almost empty can provide information about rare q-grams and thus rare attribute values.

In our experimental evaluation we will set the Bloom filter parameters for our approach according to the discussion presented here and following earlier Bloom filter work in PPRL (Schnell et al. 2009).

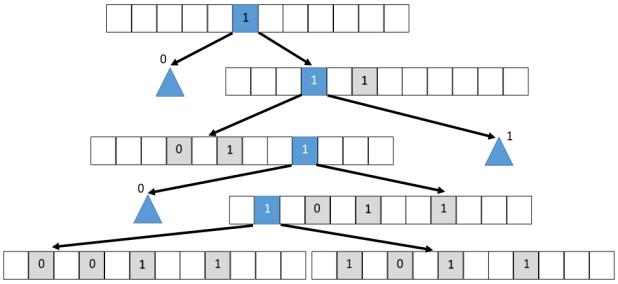

Figure 2: Single-bit tree : The blue squares represent the bits chosen for the given node, while the light gray squares mark bits chosen at an ancestor. The blue triangles represent sub-trees that are not shown.

4.1.4 Single-bit Tree Data Structure

A single-bit tree is a binary tree data structure that can be used to store information of a set of bit vectors (Kristensen et al. 2010). The construction of the tree starts from the root node, where all bit vectors are assigned to this node. At each node in the tree a position in the bit vectors is chosen to best split all the children of the node into two parts of equal size, which in turn will keep the tree as balanced as possible. All bit vectors with a 0 at that position are stored in the left subtree while all bit vectors with a 1 are stored in the right subtree. This division is continued recursively until all the bit vectors in a given node are the same, or all the bit positions have been used for the construction. Figure 2 shows an example of a single-bit tree.

It is not directly apparent how best to choose which bit position to split the data on at a given node when building the tree data structure. The selection of the best splitting bit position requires information about all the bit vectors held by a given node. This requires to view all the child bit vectors in a given node and select a bit position which contains 0 in half of the children and 1 in the other half.

The continuation of the recursive division is based on the bit vectors available in a given node and bit positions used in each parent node. Other than these two factors, in our protocol we provide a parametric solution to control this recursive division. The construction of the single bit tree data structure is described in more detail in Section 4.2.2.

4.1.5 Secure Summation Protocol

The secure summation protocol is a method used in secure multi-party computation, which has been used in several record linkage approaches (Clifton et al. 2002, Rashid et al. 2009). Secure multi-party computation was first introduced by Yao (1986) with the idea of performing computations securely such that at the end of the computation no party knows anything except its own input and the final results of the computed function (O'Keefe et al. 2004, Lindell & Pinkas 2009, Cheng et al. 2010). The secure summation protocol allows multiple cooperating parties to compute a sum over their individual data without revealing their data to the other parties.

The idea behind the secure summation protocol can be described as follows (Clifton et al. 2002, Karr et al. 2004). Protocol 1 shows the steps involved. Assume there are P parties with each one having a secret input a_i, where $1 \leq i \leq P$. The parties want to compute the summation of these inputs. Initially party P_1 chooses a large random number r and then

Protocol 1: Basic secure summation protocol

Input:
- **P** : Number of parties
- $\mathbf{a_i}$: A secret input, $1 \leq i \leq P$

Output:
- s: Final sum , where $s = \sum_1^n a_i$

1: Party P_1 generates a random number r
2: Party P_1 computes partial sum $s_1 = a_1 + r$
3: Party P_1 sends s_1 to P_2
4: **for** ($i = 2$ to P) **do**
5: Party P_i computes partial sum $s_i = s_{i-1} + a_i$
6: **if** $i = P$ **then**
7: Party P_i sends s_i to party P_1
8: **else**
9: Party P_i sends s_i to party P_{i+1}
10: Party P_1 computes final sum $s = s_P - r$
11: Party P_1 sends final sum s to other parties.

adds this random number to his input a_1. Then party P_1 sends this value to party P_2. Since R is random, party P_2 learns effectively nothing about a_1.

Party P_2 adds his value a_2 to $r + a_1$, and sends the result to party P_3. This process repeats until all the parties have added their values and the partial sum $s = r + a_1 + \ldots + a_P$ is received by the first party. Then the first party subtracts r from s and the resulting sum is distributed to all the other parties. Once the computation is finished each party only knows the total sum, from which they are unable to derive the other parties' information.

4.2 Blocking Approach

The previous sections provided details about the building blocks, which are needed for the construction of the blocking mechanism, and here we will elaborate in detail how the single-bit trees can be used as a blocking mechanism in the multi-party PPRL context. The construction of the index for a dataset of an individual party contains two main phases.

1. Generate Bloom filters for the records in the dataset.
2. Construct the single-bit tree by using the generated Bloom filters. This phase can be further extended into three sub phases, which are:
   (a) Perform secure summation to find the best splitting bit position.
   (b) Split the set of Bloom filters.
   (c) Generate the child nodes of the tree.

Each party needs to follow these phases to construct the single-bit tree for their own dataset. The overall indexing protocol, which includes all these phases mentioned above, is outlined in Protocol 2.

4.2.1 Generation of the Bloom Filters

Before the construction of the trees, the set of records needs to be encoded into Bloom filters as given in line 1 in Protocol 2. All parties need to agree upon a bit array length m (length of the Bloom filter); the length (in characters) of grams q, the k hash functions, and a set of attributes (blocking key attributes) that are used to link the records. As per step 1 in Protocol 2, each party needs to iterate over its dataset and each record needs to be encoded. Based on the optimal parameter settings calculated as described in Section 4.1.3, the blocking key values in a record are encoded

Protocol 2: Single-bit tree indexing protocol

Input:
- $\mathbf{D_i}$: Dataset belonging to party P_i
- A: Set of selected attributes
- s_{min}: Minimum bucket size
- s_{max}: Maximum bucket size

Output:
- T_i : Single-bit tree (the tree structure for dataset D_i)

Phase 1 :
1: B = generateBloomfilters(D_i, A)

Phase 2 :
2: $T_i.root$ = makeNode(B)
3: $q = [T_i.root]$ // Initialization of queue
4: **while** $q \neq \emptyset$ **do**
5: $n = q$.pop() // Get the current node

Phase 2.a:
6: R = generateRatios(n) // Generate local bit ratios
7: R_g = secureSummation(R) // Get ratios globally
8: b = getBestBit(R_g)

Phase 2.b:
9: n_0, n_1 = splitNode(n, b)

Phase 2.c:
10: **if** ($|n_0.B| \geq s_{min}$) AND ($|n_1.B| \geq s_{min}$) **then**
11: $n.left = n_0$ // Add left child
12: $n.right = n_1$ // Add right child
13: **if** ($|n_0.B| \geq s_{max}$) **then** // If blocks too large
14: q.push(n_0) // add to queue
15: **if** ($|n_1.B| \geq s_{max}$) **then** // If blocks too large
16: q.push(n_1) // add to queue

17: *return* T_i

into the Bloom filter using k hash functions. This encoding will be performed for all records in the dataset. Each party needs to generate the set of Bloom filters from their own dataset before proceeding to construct their tree.

4.2.2 Construction of the Trees

In the second phase of the protocol, each party can construct their single-bit tree by using the generated Bloom filters from their dataset. The construction of a tree for an individual party is described based on the lines of Protocol 2.

Before starting the construction process all the parties need to agree upon the two parameters of *minimum bucket size* (s_{min}) and *maximum bucket size* (s_{max}). These parameters specify the minimum and maximum number of records that need to be included in a block (bucket), respectively. The use of these parameters is elaborated in the relevant phases of the protocol below. Figures 3 to 7 illustrate the steps of Protocol 2 with an example for three parties.

- **Phase 2 : Initialization**
  As the initial step of the construction, a root node is created and the list of Bloom filters is assigned to the respective root node as the node data. A queue will be created to hold the nodes, where nodes are created at each iteration in the tree construction. Initially the root node is assigned into the queue. The iterations will continue until the queue becomes empty as per lines 2 to 5 in Protocol 2.

Party A

Bloom filter set A

A1	0	1	1	0	0
A2	1	0	1	0	1
A3	1	0	0	0	0
A4	0	1	1	1	0
A5	0	1	0	0	1
A6	1	1	0	1	0
A7	0	1	1	1	0
A8	0	1	0	0	0

Abs Diff from 0.5

1/8	1/4	0	1/8	1/4

Party B

Bloom filter set B

B1	1	0	0	1	1
B2	1	0	1	0	1
B3	0	0	1	1	0
B4	1	1	1	1	1
B5	1	1	0	0	0
B6	0	0	1	0	1
B7	0	0	0	1	1
B8	1	0	1	1	1

Abs Diff from 0.5

1/8	1/4	1/8	1/8	1/4

Party C

Bloom filter set C

C1	0	1	1	1	0
C2	0	1	0	1	0
C3	1	0	1	1	0
C4	1	1	0	0	1
C5	0	0	1	0	0
C6	1	0	0	1	0
C7	0	1	1	0	1
C8	0	1	0	1	0

Abs Diff from 0.5

1/8	1/8	0	1/8	1/4

Figure 3: Bloom filter generation and calculation of 0/1 bit ratios and absolute differences from 50% filled (Phase 2 in Protocol 2).

Random vector (R) =

10	5	12	13	6

10.125	5.25	12.0	13.125	6.25

→

10.25	5.5	12.125	13.25	6.5

→

10.375	5.625	12.125	13.375	6.75

Secure sums:

0.375	0.625	0.125	0.375	0.75

Final absolute differences from 50% filled:

0.125	0.208	**0.042**	0.125	0.25

Figure 4: Secure summation of absolute differences and selecting best bit for splitting (Phase 2.*a* in Protocol 2).

Party A

Bloom filter set A '0'

A3	1	0	0	0	0
A5	0	1	0	0	1
A6	1	1	0	1	0
A8	0	1	0	0	0

Abs Diff from 0.5

0	1/4	–	1/4	1/4

Party B

Bloom filter set B '0'

B1	1	0	0	1	1
B5	1	1	0	0	0
B7	0	0	0	1	1

Abs Diff from 0.5

1/6	1/6	–	1/6	1/6

Party C

Bloom filter set C '0'

C2	0	1	0	1	0
C4	1	1	0	0	1
C6	1	0	0	1	0
C8	0	1	0	1	0

Abs Diff from 0.5

0	1/4	–	1/4	1/4

Random vector (R) =

12	4	–	15	11

12.0	4.25	–	15.25	11.25

→

12.167	4.417	–	15.417	11.417

→

12.167	4.667	–	15.667	11.667

Secure sums:

0.167	0.667	–	0.667	0.667

Final differences from 50% filled:

0.056	0.222	–	0.222	0.222

Figure 5: Calculation of absolute differences and secure summation on sub-sets where bit 3 is 0 (Next iteration of Phase 2.*a* in Protocol 2).

Party A

Bloom filter set A '1'

A1	0	1	1	0	0
A2	1	0	1	0	1
A4	0	1	1	1	0
A7	0	1	1	1	0

Abs Diff from 0.5

1/4	1/4	–	0	1/4

Party B

Bloom filter set B '1'

B2	1	0	1	0	1
B3	0	0	1	1	0
B4	1	1	1	1	1
B6	0	0	1	0	1
B8	1	0	1	1	1

Abs Diff from 0.5

1/10	3/10	–	1/10	3/10

Party C

Bloom filter set C '1'

C1	0	1	1	1	0
C3	1	0	1	1	0
C5	0	0	1	0	0
C7	0	1	1	0	1

Abs Diff from 0.5

1/4	0	–	0	1/4

Random vector (R) =

16	8	–	7	14

16.25	8.25	–	7.0	14.25

→

16.35	8.55	–	7.1	14.55

→

16.6	8.55	–	7.1	14.8

Secure sums:

0.6	0.55	–	0.1	0.8

Final differences from 50% filled:

0.2	0.183	–	**0.033**	0.267

Figure 6: Calculation of absolute differences and secure summation on sub-sets where bit 3 is 1 (Next iteration of Phase 2.*a* in Protocol 2).

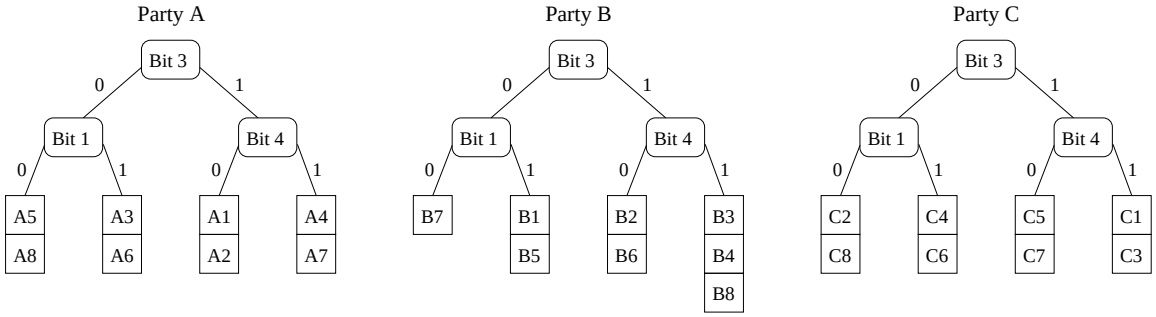

Figure 7: The resulting single-bit trees as generated by the example in Figures 3 to 6 with blocks across the three parties.

- **Phase 2.a : Find best bit position for splitting**
  At each iteration the node that is available at the front of the queue is processed. By processing the data available in this node, each party needs to generate a vector of length m that contains the values of ratios between the number of 0's and 1's for each bit position in the Bloom filters, as is calculated using Equation 3:

$$f_{ij} = abs(0.5 - \frac{o_{ij}}{l}), \tag{3}$$

  where f_{ij} is the ratio value of bit position i of party P_j, o_{ij} is the number of 1's in position i, and l is the number of Bloom filters processed in a given node.

  The bit positions that are having a value of 1 in half of the Bloom filters are given the lowest ratio value of 0, and the bit positions that are having 1's or 0's in all the Bloom filters are given the highest ratio value of 0.5. This processing is shown in lines 6 and 7 in Protocol 2.

  Once all parties have computed the ratio vectors of the bit positions locally based on their individual node data, a common bit position needs to be selected as the best bit for splitting the set of Bloom filters for the child nodes in the next level. For computing this global bit position, we extend Protocol 1 to securely compute the summation of these vectors of ratios, where each party P_j has as private input a vector f_j of length m, and the random value r in Protocol 1 is extended to a random number vector of the same length as the ratio vector. Once the secure summation step (line 7 in Protocol 2) is finished the globally summed ratio vector is used to find the best splitting bit position:

$$i_{best} = argmin\{i : (\frac{P}{2} - \sum_{j=1}^{P} f_{ij})\} \tag{4}$$

  Once each party receives the globally summed ratio vector, the best splitting bit position (i_{best}) is selected to divide the data which are available in the current node, as shown in line 8 of Protocol 2. This best splitting bit position is selected by ranking the globally summed ratio vector according to the summed values, where the bit position with the lowest sum gets the highest rank as shown in Equation 4.

- **Phase 2.b: Split the set of Bloom filters**
  The selected global best bit position is used to split all the Bloom filters of the current node into two portions. All the Bloom filters that contain a 0 in the best bit position are assigned to the left portion of the list and all others are assigned to the right portion. These portions are assigned to two new tree nodes, which are to be processed in the next iterations.

- **Phase 2.c : Generate the child nodes**
  According to the selected best bit position, these two portions may contain an uneven number of Bloom filters, i.e. one portion contains a smaller number of records. The disadvantage of such a division is that sensitive information can be revealed about the blocks that are having a smaller number of records, where an adversary can potentially re-identify individual records (Sweeney 2002). As a solution we propose the *minimum block size* (s_{min}) to guarantee that every block in the tree structure contains at least s_{min} records. After splitting, if any of the portions in the resulting lists contain less than s_{min} records, then these portions will not be assigned to any block in the given node. Instead the two portions are merged and the resulting list is included as a relevant block in the parent of the current node (line 10 in Protocol 2). All parties need to merge the two portions for the current iteration, which is informed by using the secure summation protocol as explained in Phase 2.*a*.

  If these portions contain a number of records greater than s_{min}, then these newly created nodes are assigned as child nodes to the current node, i.e. the node with the left portion becomes the left child of the current node and the other becomes the right child respectively (lines 11 and 12 in Protocol 2).

  One important consideration in the tree construction phase is to have control over the number of blocks created in the tree. We provide a parameter *maximum block size* (s_{max}) for this purpose. This allows the user to control the maximum number of records that can be contained in a block, which indirectly controls the number of iterations that occur when creating subtrees. After splitting the current node data, each portion is checked against the value of s_{max}. If any of the two portions contains less than s_{max} records, then a new block (bucket) is created to hold the relevant portion and is assigned to the current node. If the number of records is greater than s_{max}, then these two child nodes are added to the queue for future splitting (lines 13 to 16 in Protocol 2). Therefore the continuation of iterations is decided based on the number of blocks that are generated.

5 Analysis of the Protocol

In this section we analyze our protocol in terms of complexity, privacy, and quality of blocking.

5.1 Complexity

In this section we analyze the computational and communication complexities of our blocking protocol in terms of a single party. Let us assume there are N records in the dataset with each having an average of n q-grams. In the Bloom filter generation all the records are encoded using k hash functions. This encoding process is applied to all the records in a linear manner. Therefore the Bloom filter generation for a single party is of $O(k \cdot n \cdot N)$.

In the second phase of the protocol the single-bit tree construction starts once the records are encoded into N Bloom filters. In our protocol the parameter s_{max} is used to control the number of blocks, which indirectly controls the number of levels generated in the tree data structure. If s_{max} is equal to N, then the number of levels in the tree becomes 1 (all the records are assigned to the root node), while if s_{max} is equal to $N/2$ then the tree will have two levels (root node with two child nodes), and so on. When s_{max} is equal to 1 the single-bit tree is constructed with $log_2(N)$ levels. Therefore the number of levels in a single-bit tree can be calculated as $log_2(N/s_{max})$. At each level of the tree a total of N records are processed, where all child nodes in a given level hold a total of N records. Therefore the insertion of N Bloom filters into a single-bit tree requires a computational complexity of $O(N \cdot log_2(N/s_{max}))$.

In our protocol, the parties only need to communicate with each other in order to perform the secure summation protocol to find the best bit for splitting the data of the nodes, and to check if block sizes are less than s_{min}. This requires communication in the creation of each node. By assuming each party directly connects to other parties, the distribution of the final sum to P parties requires P messages for each node in the tree data structure, each of size m where m is the length of a Bloom filter. Therefore the entire protocol has a communication complexity of $O(m \cdot P \cdot 2^{N/s_{max}})$ for P parties. The computation of the secure summation protocol requires each party to process the data of the node in a given iteration, which needs a set of Bloom filters to be scanned for each bit position to get a count of the number of 1's. Therefore line 8 of our protocol has a computational complexity of $O(m \cdot N)$ for each level in the tree.

5.2 Privacy

In our protocol we assume each party follows the honest-but-curious (semi-honest) adversary model (Al-Lawati et al. 2005, Scannapieco et al. 2007), where each party follows the steps of the protocol while trying to find as much as possible about the data from the other parties. Privacy is a main factor that needs to be considered to evaluate the amount of information a party can learn from the data from the other parties when they communicate during the protocol. In our protocol, the parties communicate with each other to compute the global best bit position for splitting. For the exchange of ratio values of the Bloom filters our protocol uses a secure summation protocol.

During the secure summation, each party sums their ratio vector with the partial resulting vector sent by the previous party, but will not be able to learn any information about the ratio values of the previous party since the random vector is only known to the party that initiated the protocol. Once the initiated party received the final partial sum vector, he subtracts the random vector from the summed values, but is not capable of deducing anything about the other parties' ratio vector values. At the end of our blocking protocol, the set of blocks are generated and private linkage can be conducted on each respective block by using a private matching and classification technique (Atallah et al. 2003, Ravikumar et al. 2004, Vatsalan & Christen 2012, Durham et al. 2013), which should not reveal any information regarding the sensitive attributes and non-matches.

Our protocol performs a generalization strategy on the blocks that makes the re-identification from the perturbed data not possible (Sweeney 2002). The parameter *minimum block size* (s_{min}) is used to guarantee that every block in the tree structure contains at least s_{min} records. This ensures all blocks that are generated have the same minimum number of records, which makes a dictionary attack, where an adversary hash-encodes values from a large publicly available dataset using existing hash encoding functions, or a frequency attack, much more difficult (Vatsalan et al. 2013*b*).

5.3 Quality

The quality of our protocol is analyzed in terms of effectiveness, which requires all similar records to be grouped into the same block, and efficiency, which requires the number of candidate record sets generated to be as small as possible while including all true matching record sets (Vatsalan et al. 2013*b*). By assuming each block contains s_{max} records and there are b blocks in each tree for P parties, the number of candidate record sets generated by our approach is $((s^P_{max})b)$. The parameter s_{max} decides the number of child nodes that are created in the single-bit tree, which in turn controls the number of blocks generated. If the value of s_{max} is large, the number of blocks generated is reduced by the protocol, while a smaller s_{max} value provides a single-bit tree with more blocks. An optimal value for s_{max} needs to be set by considering factors such as the dataset size and the number of parties, such that both effectiveness and efficiency are achieved while guaranteeing sufficient privacy as well.

6 Experimental Evaluation and Discussion

We evaluated our protocol by performing experiments using a large real world database. In the following sub-sections we provide details on the datasets that we used for our experiments, implementation details of the proposed indexing protocol, and the evaluation measures used in the experiments.

6.1 Datasets

To provide a realistic evaluation of our approach, we based all our experiments on a large real-world database, the North Carolina Voter Registration (NCVR) database as available from `ftp://alt.ncsbe.gov/data`. This database has been used for the evaluation of various other PPRL approaches (Vatsalan et al. 2013*a*, Durham et al. 2013). We have downloaded this database every second month since October 2011 and built a combined temporal dataset that contains over 8 million records of voter's names and addresses.

We are not aware of any available real-world dataset that contains records from more than two parties that would allow to us evaluate our multi-party approach. We therefore created, based on the real NCVR database, a series of sub-sets as described next.

To allow the evaluation of our approach on different number of parties, with different dataset sizes, and with data of different quality, we used and modified a recently proposed data corruptor (Christen & Vatsalan 2013) to generate various datasets with different characteristics based on randomly selecting records from the NCVR database. During the corruption process we keep the identifiers of the selected and modified records, which allows us to identify true and false matches and therefore calculate various blocking quality and complexity measures as will be discussed in Section 6.3.

Specifically, we selected sub-sets from the full NCVR database for 3, 5, 7 and 10 parties, that contain $5,000$, $10,000$, $50,000$, $100,000$, $500,000$ and $1,000,000$ records, respectively. In each of these sub-sets, 50% of records were matches, i.e. half of all records occur in the sub-sets of all parties. We then applied various corruption functions on different numbers (ranging from 1 to 3) of randomly selected attribute values which allows us to investigate how our approach can handle 'dirty' data. We applied various corruption functions, including character edit operations (insertions, deletions, substitutions, and transpositions), and optical character recognition and phonetic modifications based on look-up tables and corruption rules (Christen & Vatsalan 2013).

We also created groups of datasets where we included a varying number of corrupted records into the sets of overlapping records (ranging from 0% to 100% corruption, in 20% steps). This means that a certain percentage of records in the overlap were modified for randomly selected parties. Therefore, some of these records are exact duplicates across some parties in a group, but only approximately matching duplicates across the other parties in the group. This simulates for example the situation where three out of five hospitals have the correct and complete contact details (like name and address) of a certain patient, while in the fourth and fifth hospitals some of the details of the same patient are different.

From the created datasets we extracted four attributes commonly used for record linkage: Given name, Surname, Suburb (town) name, and Postcode. These four attributes were used for generating the Bloom filters for the experiments. In our experiments we set the Bloom filter parameters as $m = 1000$ bits, $k = 30$, and $q = 2$ by following earlier Bloom filter based work in PPRL (Schnell et al. 2009).

6.2 Implementation

We implemented a prototype to evaluate the performance of our protocol using the Python programming language (version 2.7.3). We implemented prototypes for single-bit tree construction with a recursive approach and a loop iteration approach. The run time was measured for both of these approaches on different datasets of various sizes. The results showed that the tree construction using the iterative approach runs faster than the recursive approach. So we based all the experiments using the single-bit tree construction on the iterative approach. All the experiments were run on a server with 64-bit Intel Xeon (2.4 GHz) CPU, with 128 GBytes of main memory and running Ubuntu 12.04. The programs and test datasets are available from the authors.

Table 2: Average memory (MBytes) requirement per party.

Data set	BF Construction	Tree construction
5,000	20	23
10,000	33	47
50,000	112	117
100,000	340	370
500,000	970	987
1,000,000	$1,920$	$1,958$

We performed a comparison with a phonetic based blocking approach as a baseline to measure the level of privacy provided by our indexing approach. In the phonetic blocking we used Soundex (Christen 2012*b*) as encoding function for the Given name, Surname and Suburb attributes, while for the Postcode attribute the first three digits of a Postcode value are used as the blocking key.

6.3 Evaluation Measures

We evaluated our protocol in terms of complexity, blocking quality, and privacy for different sizes of the dataset, different number of parties, and different block sizes. We measured the runtime for generating Bloom filters and for the single-bit tree construction to evaluate the time complexity of blocking. The reduction ratio (RR) and pairs completeness (PC) are used to evaluate the blocking quality, which are standard measures to assess the efficiency and the effectiveness of blocking (Christen 2012*a*), respectively. The RR and PC are calculated as Equations 5 and 6:

$$RR = 1 - \frac{BL_{CS}}{T_{RS}} \tag{5}$$

$$PC = \frac{BL_{TM}}{T_{TM}} \tag{6}$$

where BL_{CS} is the number of candidate record sets generated by blocking, BL_{TM} is the number of true matching candidate record sets generated by blocking, T_{TM} is the number of total true matching record sets in the datasets, and T_{RS} is the total number of record sets.

According to Vatsalan et al. (2014), a standard and normalized measure to quantify privacy based on simulated attacks (Kuzu et al. 2013) has so far not been studied. Vatsalan et al. (2013*a*) introduced a set of disclosure risk (DR) measures that can be used to evaluate and compare different private blocking and PPRL solutions. To evaluate the privacy of our protocol we use the measure *probability of suspicion* (P_s), which is defined for a value in an encoded dataset as $1/n_g$, where n_g is the number of values in a global dataset (G) that match with the corresponding value in an encoded (protected) dataset D.

This measure of P_s is normalized between 0 and 1, where 1 indicates that the value in D can be exactly re-identified with a value in G based on one-to-one matching, and 0 means a value in D could correspond to any value in G and therefore it cannot be re-identified. Based on the normalized P_s values for each value in D, the *maximum disclosure risk* and the *mean disclosure risk* (Mean) of D can be calculated as the maximum value of P_s ($max(Ps)$) of any value in D, and as the average risk ($\sum(Ps)/|D|$) by considering the distribution of P_s of all values in D, respectively.

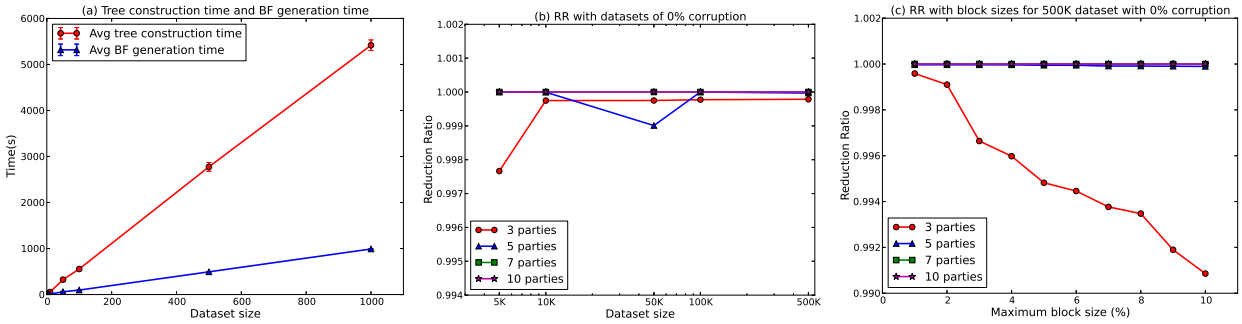

Figure 8: (a) Average time for tree construction and Bloom filter generation, (b) Reduction ratio (RR) with dataset size, and (c) RR with different block sizes.

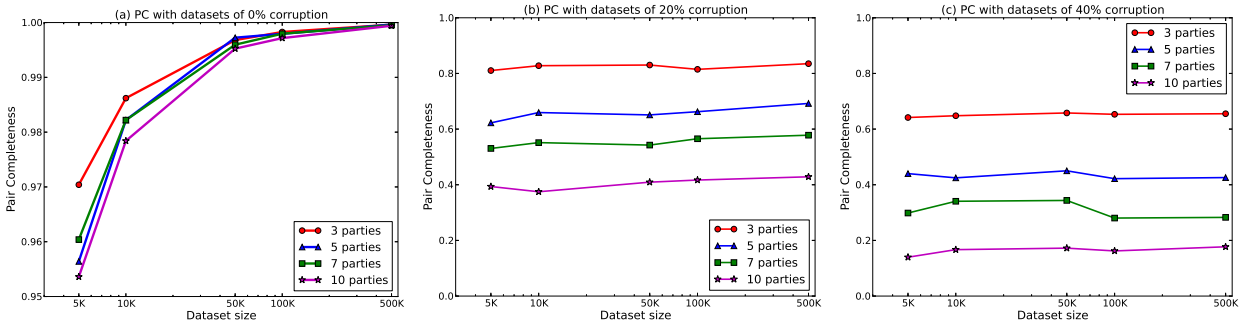

Figure 9: Pair completeness (PC) with dataset sizes for (a) 0% corruption, (b) 20% corruption, and (c) 40% corruption.

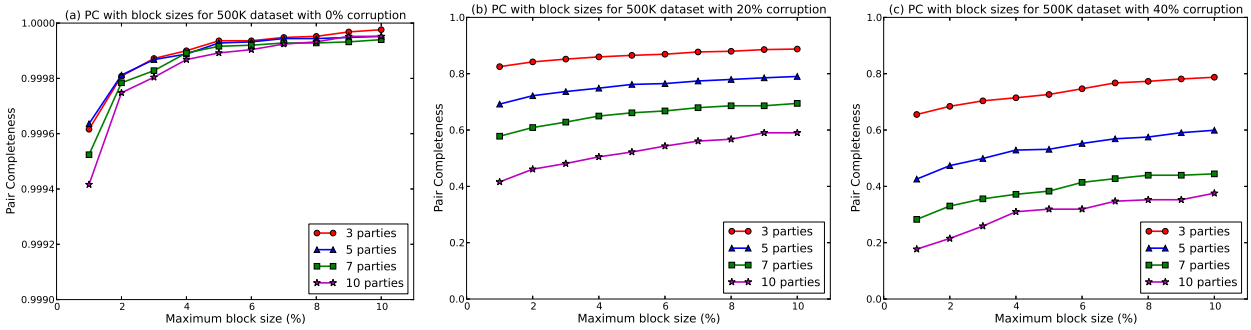

Figure 10: Pair completeness (PC) with different block sizes for (a) 0% corruption, (b) 20% corruption, and (c) 40% corruption.

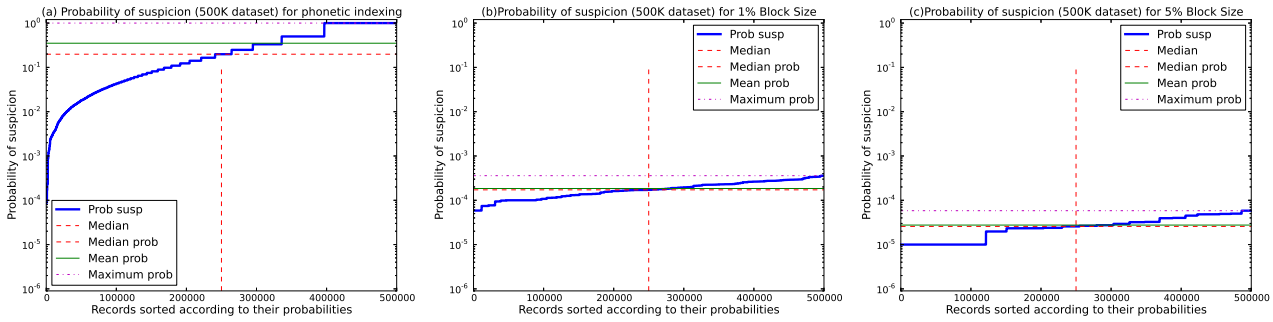

Figure 11: Probability of suspicion (P_s) in the 500K dataset with (a) phonetic indexing, (b) single-bit tree with 1% block size, and (c) single-bit tree with 5% block size.

6.4 Discussion

Figure 8 shows the scalability of our approach in terms of the average time required for generating Bloom filters and the single-bit tree construction time for a single party, and the reduction ratio with different sizes of the dataset and blocks. As expected the tree construction time increases linearly with the dataset sizes. Reduction ratio remains nearly 1 for different dataset sizes and for different number of parties, which illustrates our approach is reducing the

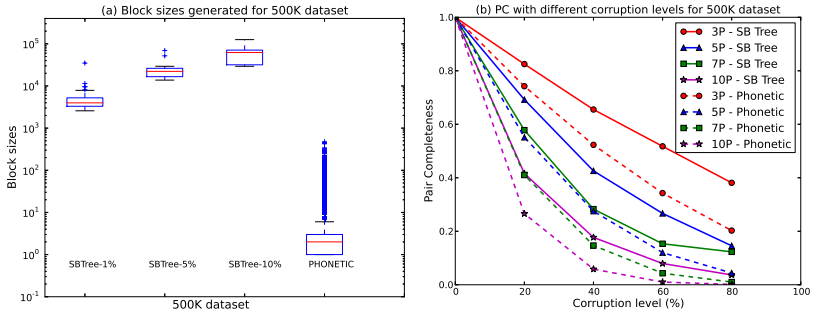

Figure 12: (a) Block sizes generated by phonetic indexing, and single-bit tree with 1%, 5%, and 10% block sizes and (b) Pair Completeness (PC) with different corruption levels for phonetic indexing and single-bit tree.

number of total candidate record sets that need to be compared dramatically. Table 2 shows that our approach can be run with less than 2 GBytes of memory to construct a tree even for one million records.

Figure 9 illustrates the pair completeness (PC) of our approach with different dataset sizes. It shows that PC slightly increases with dataset size (Figure 9(a)), which indicates that the blocks of the single-bit tree contain more true matching records. It is noted that the blocking quality of our approach is affected by the quality of the data and PC is decreasing rapidly with the number of parties with low quality data, as illustrated in Figures 9(b) and 9(c).

As shown in Figure 10, PC increases when the size of the blocks is increased. The block sizes are controlled with the s_{max} parameter, where high PC values are achieved with high s_{max} values. Figures 10(b) and 10(c) illustrate the increment of the PC values with different quality levels of the data for different block sizes.

Privacy is a main aspect of any indexing mechanism in PPRL. We compute P_s values for a single party by assuming all trees contain similar block structures. As shown in Figure 11(b) our approach is having a maximum P_s value less than 0.0001 for each individual, which indicates a record in a block can be matched to more than $10,000$ values in a global dataset (under the worst case assumption of global dataset G is being equal to the linkage dataset). It can be noted that our approach is providing significantly better privacy compared to the phonetic indexing approach that is having a maximum P_s of 1 as shown in Figure 11(a). Figure 11(c) illustrates that better privacy can be achieved with larger block sizes.

We compare our approach with the phonetic indexing in terms of the blocks sizes generated and how the PC is changing with the quality of the data as shown in Figure 12. It shows that the phonetic indexing approach creates a large number of blocks of size 1 (Figure 12(a)). This makes the phonetic based approach not suitable for PPRL, because these records can be exactly re-identified by using a value in G based on one-to-one matching. Figure 12(b) shows that our approach achieves higher PC values than the phonetic indexing even with low quality data.

7 Conclusion

In this paper, we presented a novel indexing protocol for multi-party privacy-preserving record linkage based on Bloom filters and single-bit tree data structures. Each party constructs the single-bit tree index based on the Bloom filters generated on their dataset, and the parties communicate with each other to compute the best bit positions to be used for construction by using a secure summation protocol. The proposed approach was validated by an experimental evaluation, where we performed the experiments on different datasets with a size of up-to one million records. The evaluation results indicated that our approach is scalable with both the size of the databases to be linked and the number of parties. Our approach also outperforms a phonetic indexing approach in terms of privacy and quality of blocking. The blocks which are generated can finally be compared using private comparison and classification techniques to determine the similar record sets in different databases.

We plan to extend our protocol with different tree data structures, which can reduce the number of levels of the trees and by using more bits for the splitting of the tree nodes. We will also investigate the parallelization of the algorithm, which can further improve the performance of our protocol. A limitation in our approach is the assumption of the semi-honest adversary model which is not applicable for some real-world applications. Privacy can be compromised when some parties are malicious which requires more secure communication techniques than the adopted secure summation protocol. We aim to extend this protocol for different adversary models for allowing it to be employed in real-world PPRL applications.

References

Al-Lawati, A., Lee, D. & McDaniel, P. (2005), Blocking-aware private record linkage, *in* 'ACM IQIS', Baltimore, pp. 59–68.

Atallah, M., Kerschbaum, F. & Du, W. (2003), Secure and private sequence comparisons, *in* 'ACM WPES', pp. 39–44.

Bachteler, T., Reiher, J. & Schnell, R. (2013), Similarity filtering with multibit trees for record linkage, Technical report, Working Paper WP-GRLC-2013-02, German Record Linkage Center, Nüremberg.

Bloom, B. (1970), 'Space/time trade-offs in hash coding with allowable errors', *Communications of the ACM* **13**(7), 422–426.

Broder, A. & Mitzenmacher, M. (2004), 'Network applications of Bloom filters: A survey', *Internet mathematics* **1**(4), 485–509.

Cheng, C., Luo, Y.-L., Chen, C.-X. & Zhao, X.-K. (2010), 'Research on secure multi-party ranking problem and secure selection problem'.

Christen, P. (2012*a*), *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Data-Centric Systems and Applications, Springer.

Christen, P. (2012*b*), 'A survey of indexing techniques for scalable record linkage and deduplication', *IEEE TKDE* **24**(9), 1537–1555.

Christen, P. & Vatsalan, D. (2013), Flexible and extensible generation and corruption of personal data, *in* 'ACM CIKM', San Francisco, pp. 1165–1168.

Churches, T. & Christen, P. (2004), Blind data linkage using n-gram similarity comparisons, *in* 'PAKDD', Sydney, pp. 121–126.

Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X. & Zhu, M. (2002), 'Tools for privacy preserving distributed data mining', *SIGKDD Explorations* **4**(2), 28–34.

de Vries, T., Ke, H., Chawla, S. & Christen, P. (2011), 'Robust record linkage blocking using suffix arrays and Bloom filters', *ACM TKDD* **5**(2).

Durham, E. (2012), A framework for accurate, efficient private record linkage, PhD thesis, Faculty of the Graduate School of Vanderbilt University, Nashville, TN.

Durham, E. A., Toth, C., Kuzu, M., Kantarcioglu, M., Xue, Y. & Malin, B. (2013), 'Composite bloom filters for secure record linkage', *IEEE TKDE* .

Fellegi, I. P. & Sunter, A. B. (1969), 'A theory for record linkage', *Journal of the American Statistical Society* **64**, 1183–1210.

Hawashin, B., Fotouhi, F. & Truta, T. (2011), A privacy preserving efficient protocol for semantic similarity join using long string attributes, *in* 'ACM PAIS', Uppsala, Sweden.

Inan, A., Kantarcioglu, M., Ghinita, G. & Bertino, E. (2010), Private record matching using differential privacy, *in* 'EDBT', Lausanne.

Karakasidis, A. & Verykios, V. (2011), 'Secure blocking+secure matching = secure record linkage', *Journal of Computing Science and Engineering* **5**, 223–235.

Karr, A. F., Lin, X., Sanil, A. P. & Reiter, J. P. (2004), Analysis of integrated data without data integration, Vol. 17, Chance, pp. 26–29.

Kristensen, T. G., Nielsen, J. & Pedersen, C. N. (2010), 'A tree-based method for the rapid screening of chemical fingerprints', *Algorithms for Molecular Biology* **5**(1), 9.

Kuzu, M., Kantarcioglu, M., Durham, E. A., Toth, C. & Malin, B. (2013), 'A practical approach to achieve private medical record linkage in light of public resources', *Journal of the American Medical Informatics Association* **20**(2), 285–292.

Lai, P., Yiu, S., Chow, K., Chong, C. & Hui, L. (2006), An efficient Bloom filter based solution for multiparty private matching, *in* 'International Conference on Security and Management'.

Lindell, Y. & Pinkas, B. (2009), 'Secure multiparty computation for privacy-preserving data mining', *Journal of Privacy and Confidentiality* **1**(1), 5.

Mitzenmacher, M. & Upfal, E. (2005), *Probability and computing: Randomized algorithms and probabilistic analysis*, Cambridge University Press.

O'Keefe, C. M., Yung, M., Gu, L. & Baxter, R. (2004), Privacy-preserving data linkage protocols, *in* 'ACM WPES', Washington DC, pp. 94–102.

Rashid, S., Brijesh, K. & Mishra, D. K. (2009), Privacy preserving k-secure sum protocols, *in* 'Computer Science and Information Security', Vol. 6, pp. 40–46.

Ravikumar, P., Cohen, W. & Fienberg, S. (2004), A secure protocol for computing string distance metrics, *in* 'PSDM held at IEEE ICDM', Brighton, UK, pp. 40–46.

Scannapieco, M., Figotin, I., Bertino, E. & Elmagarmid, A. (2007), Privacy preserving schema and data matching, *in* 'ACM SIGMOD', pp. 653–664.

Schnell, R., Bachteler, T. & Reiher, J. (2009), 'Privacy-preserving record linkage using Bloom filters', *BMC Medical Informatics and Decision Making* **9**(1).

Song, D., Wagner, D. & Perrig, A. (2000), Practical techniques for searches on encrypted data, *in* 'IEEE Symposium of Security and Privacy', Oakland, pp. 44–55.

Sweeney, L. (2002), 'K-anonymity: A model for protecting privacy', *International Journal of Uncertainty Fuzziness and Knowledge Based Systems* **10**(5), 557–570.

Vatsalan, D. & Christen, P. (2012), An iterative two-party protocol for scalable privacy-preserving record linkage, *in* 'AusDM, CRPIT 134', Sydney, Australia.

Vatsalan, D., Christen, P., O'Keefe, C. M. & Verykios, V. S. (2014), 'An evaluation framework for privacy-preserving record linkage', *Journal of Privacy and Confidentiality* **6**(1), 35–75.

Vatsalan, D., Christen, P. & Verykios, V. S. (2013*a*), Efficient two-party private blocking based on sorted nearest neighborhood clustering, *in* 'ACM CIKM', San Francisco, pp. 1949–1958.

Vatsalan, D., Christen, P. & Verykios, V. S. (2013*b*), 'A taxonomy of privacy-preserving record linkage techniques', *Elsevier Journal of Information Systems* **38**(6), 946–969.

Yakout, M., Atallah, M. & Elmagarmid, A. (2012), 'Efficient and practical approach for private record linkage', *ACM JDIQ* **3**(3), 5.

Yao, A. (1986), How to generate and exchange secrets, *in* 'Foundations of Computer Science', IEEE, pp. 162–167.