**Wen Z, Dong C.**

**Efficient protocols for private record linkage.**

***In: SAC '14: Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 2014, Gyeongju, Korea: ACM.**

# Efficient Protocols for Private Record Linkage

Zikai Wen[*†]
wjb12186@uni.strath.ac.uk

[†] College of Information Science and Technology
Beijing University of Chemical Technology
Beijing, China

Changyu Dong[*]
changyu.dong@strath.ac.uk

[*] Dept. of Computer and Information Sciences
University of Strathclyde
Glasgow, UK

## ABSTRACT

Record linkage allows data from different sources to be integrated to facilitate data mining tasks. However, in many cases, records have to be linked by personally identifiable information. To prevent privacy breaches, ideally records should be linked in a private way such that no information other than the matching result is leaked in the process. In this paper, we present an exact Private Record Linkage (PRL) protocol and an approximate PRL protocol. The exact PRL protocol is based on Oblivious Bloom Intersection, which is an efficient private set intersection protocol. The approximate PRL protocol extends the exact PRL protocol by incorporating Locality Sensitive Hash functions. Both protocols are secure in the semi-honest model. We also report the evaluation results based on our C implementation of the protocols. The results show that our protocols are efficient and effective.

## 1. INTRODUCTION

Data is invaluable to organizations. Everyday, a large amount of data is generated, collected and stored. By analyzing the data, new knowledge can be discovered that will lead to improvement in public health, productivity for government agencies, and competitive edge for a commercial enterprise. However, often data is possessed by different entities separately, therefore needs to be integrated to facilitate data mining that is not feasible on a single database. To link two databases, record pairs are compared using a variety of fields and record comparison functions. The goal is to classify the record pairs into matches and non-matches.

Linking data may cause privacy concerns. When linking two databases from different sources, the data usually lacks unique entity identifiers. That means linking is often based on some *personally identifiable information*. Sharing personal information across multiple entities may cause a breach of privacy. Legally, it may also be prohibited by laws and regulations. For example, in the USA the Health Insurance Portability and Accountability Act (HIPPA) sets the standard for protecting health data, e.g. what kind of matching and analysis can be conducted with health data, and at what level of detail health data can be published. Similarly, in Europe, Data Protection Directive regulates the processing of personal data. Thus it is vital to ensure that whenever databases are linked across organizations, privacy of individuals is maintained.


*SAC'14* March 24-28, 2014, Gyeongju, Korea.

http://dx.doi.org/10.1145/2554850.2555001 ...$15.00.

Private Record Linkage (PRL) is the process of identifying records from multiple data sources that refer to the same individual, without revealing more information besides the matched records. More formally, assuming $A$ and $B$ are the data owners and each holds a database $D_A, D_B$ respectively, for each record $R_i^A \in D_A$ and $R_j^B \in D_B$, they want to decide whether $R_i^A \cong R_j^B$ where $\cong$ is the match relation defined by certain record comparison functions. In the process data privacy must be retained in the sense that no more information other than the linking result ($R_i^A \cong R_j^B$ or $R_i^A \ncong R_j^B$) should be leaked. PRL has many practical uses. For example, epidemiological research often requires correlated demographics data with diseases to identify possible risk factors or targets for preventive medicine. However in many countries the data is held separately by different registries, and often privacy concerns arise if such data is stored and linked at a central location. As a result, in the past a few years we have seen a lot of research work in this area.

There are three requirements for a PRL protocol:

- Privacy: As one of the goals of PRL is to maintain privacy, it must not allow additional information to be leaked from the linking process. The privacy guarantee must stand rigorous analysis.
- Effective: In case of exact match, the identified matches must be correct. In case of non-exact match, the quality of the identified matches must conform certain evaluation criteria.
- Efficient: The database being linked can be very large, therefore the protocol needs to be efficient so the linking can be done in a reasonable amount of time.

**Our Contributions:** In this paper, we present two novel and efficient PRL protocols. They are based on Oblivious Bloom Intersection, an efficient and scalable Private Set Intersection protocol. The first protocol supports exact match. The second protocol is built on top of the exact match protocol and supports similarity-based approximate match by incorporating Locality Sensitive Hashing. The protocols are secure in the semi-honest model. The protocols are very efficient and have linear complexity. We have built prototypes of the PRL protocols and evaluated the protocols in terms of efficiency and accuracy. We also compared our protocols with other protocols and the comparison shows that our protocols are much more efficient.

## 2. RELATED WORK

Work in PRL can be classified into two categories, exact PRL which only matches records when the matching attributes are exactly identical, and approximate PRL which matches two records if they are very similar. Early work in exact PRL can be traced

back to [8]. The protocol hashes the records and uses the hash values instead of the records for linking two databases. The protocol however is subject to frequency attacks because hash functions are deterministic. Later exact PRL work includes various protocols that rely on private set intersection [1, 9, 14, 11, 6, 12].

Approximate PRL usually depends on some sort of similarity measures. If, according to the chosen similarity measure, the distance between two record values is less than a certain threshold, then the two records are considered a match. Many approximate PRL protocol are three-party protocols, i.e. apart from the two parties who hold the data, there is also a trusted or semi-trusted party involved in the protocol. In [5] the two parties hash their attribute values to be compared, then the hash values are sent to a third party who compares the values blindly and finds values in the two sets that match. In [19] the two parties embed their values using the SparseMap method. Then the embedded strings are sent to a third party who determines the similarity. A third party can make protocol more efficient. However, in practice a trusted or semi-trusted third party may not always be available. In a strict two-party setting, [2] links records by securely computing edit distance of strings, [18] securely computes TF-IDF (term-frequency, inverse document frequency), [22] uses phonetic functions and commutative encryption to link records, and [24] proposed a very efficient protocol that first converts records into vectors and then securely computes the distance between each pair of the records. A good survey of PRL protocols can be found in [23].

Our protocol uses Bloom filters. Previously, a protocol in [20] is also based on Bloom filters. However, our approach is quite different from that in [20]. Firstly, in [20] , each record needs a Bloom filter to store q-grams of the attribute values to be compared, while in our protocol, only one Bloom filter is required for the whole database. Secondly, in [20], a semi-trusted third party is needed, while in our protocol we do not need a third party. The protocol in [20] has also been shown to be insecure [15]. Our protocol is secure in the semi-honest model.

# 3. THE EXACT PRL PROTOCOL

In this section, we introduce the exact PRL protocol. Here exact means the match relation is defined by an equality relation on the attribute values being compared. The protocol follows the private set intersection approach and is an extension of the Oblivious Bloom Intersection (OBI) protocol [7].

## 3.1 Oblivious Bloom Intersection

Oblivious Bloom Intersection (OBI) [7] is an efficient and scalable private set intersection protocol. A private set intersection protocol is a protocol between two parties, $A$ and $B$. Each party has a private set as input. The goal of the protocol is that $A$ learns the intersection of the two input sets, but nothing more about $B$'s set, and $B$ learns nothing. A PRL protocol can be built on top of a private set intersection protocol directly. However, previous private set intersection protocols are not efficient enough to be used in real applications. This situation is changed by the recently proposed OBI protocol. The OBI protocol adapts a very different approach for computing set intersections. It is mainly based on efficient hash operations. Therefore it is significantly faster than previous private set intersection protocols. In addition, the protocol can also be parallelized easily, which means performance can be further improved by parallelization. The protocol is secure in the semi-honest model and an enhanced version is secure in the malicious model. We refer the readers to [7] for more details regarding OBI. In the rest of this section we will show how to modify OBI and build an exact PRL protocol on top of it.

## 3.2 Bloom Filters and Garbled Bloom Filters

The OBI protocol relies on Bloom filters (BF) and garbled Bloom filters (GBF). Both BF and GBF are probabilistic data structures that encode set membership and allow queries without false negative but may have negligible false positive. The two data structures are both size $m$ arrays that can encode a set $S$ of at most $n$ elements, and are associated with a set of $k$ independent uniform hash functions $H = \{h_0, ..., h_{k-1}\}$ such that each $h_i$ maps elements to index numbers over the range $[0, m-1]$ uniformly. The main difference between the two is that a Bloom filter is a bit array while a garbled Bloom filter is an array of $\lambda$-bit strings. We will use the notation in [7]: we use $(m, n, k, H)$-Bloom filter to denote a Bloom filter parameterized by $(m, n, k, H)$, $(m, n, k, H, \lambda)$-Garbled Bloom filter to denote a garbled Bloom filter parameterized by $(m, n, k, H, \lambda)$, use $BF_S$ or $GBF_S$ to denote a Bloom filter or a garbled Bloom filter that encodes the set $S$, and use $BF_S[i]$ or $GBF_S[i]$ to denote the bit or string at index $i$ in the filter's array.

The two data structures work in a similar way. For a Bloom filter, to insert an element $x \in S$ into the filter, the element is hashed using the $k$ hash functions and set $BF_S[h_i(x)] = 1$. To check if an item $y$ is in $S$, $y$ is hashed by the $k$ hash functions, and all locations $y$ hashes to are checked. If any of the bits at the locations is 0 , $y$ is not in $S$, otherwise $y$ is *probably* in $S$. For a garbled Bloom filter, to insert an element $x \in S$ into the filter, $x$ is split into $k$ shares using an XOR-based secret sharing scheme. The secret sharing scheme ensures that the element can only be recovered from the shares if all $k$ shares are available. The element $x$ is hashed using the $k$ hash functions to get $k$ indexes and the shares of $x$ are stored in the $GBF$ array by the indexes, one share at each index. To check if an item $y$ is in $S$, $y$ is hashed by the $k$ hash functions, and all strings at locations $y$ hashes to are retrieved back and XORed together. if the result string is not $y$ , $y$ is not in $S$, otherwise $y$ is *probably* in $S$.

## 3.3 Modified Garbled Bloom Filter

The idea of the OBI protocol is that $A$ encodes its set into a Bloom filter and $B$ encodes its set into a garbled Bloom filter, then they run an oblivious transfer protocol so that $A$ receives a new garbled Bloom filter that encodes the set intersection and $B$ gets nothing. The reason why $A$ uses a Bloom filter and $B$ uses a garbled Bloom filter is because if both parties uses Bloom filters then the protocol is not secure, while if both parties uses garbled Bloom filters then the protocol cannot produce meaning results.

However this approach cannot be directly used in PRL. The original algorithm in [7] for building garbled Bloom filters is not suitable and needs to be modified. Loosely speaking, the original algorithm allows $A$ to find a subset of its records that can be linked to records in $B$'s database, but $A$ cannot find out for each of the records in the subset, which record in $B$'s database is linked to it. Let us elaborate it: for a database that has $n$ records, we assume each record $R_i$ has a unique identifier $id_i$ and there is a function $f$ such that $x_i = f(R_i)$ is computable and $x_i$ can be used for record matching purpose. For example, $x_i$ can be the concatenation of some attribute values of $R_i$, or a hash value of $R_i$ (or part of $R_i$). We say a record $R_i^A \in D_A$ matches another record $R_j^B \in D_B$ if $x_i^A = x_j^B$. For two databases $D_A$ and $D_B$, an exact PRL protocol should return the following: $\{(id_i^A, id_j^B) \mid x_i^A = x_j^B\}$, i.e. all pairs of record IDs of matching records in the two databases. If we use OBI without any modification, then $A$ uses the set of all $x_i^A$ to generate a Bloom filter, and $B$ uses the set of all $x_j^B$ to generate a garbled Bloom filter. Then they run the oblivious transfer protocol so that $A$ obtains a garbled Bloom filter that encodes the intersection. After that, $A$ can query each $x_i^A$ against the intersection garbled Bloom filter to find out which ones are in the inter-

section. For each $x_i^A$ such that the query result is positive, $A$ only knows it is in the intersection and then there must be some $x_j^B$ such that $x_i^A = x_j^B$. In other words, at the end of the protocol, $A$ can only output $\{x_i^A \mid x_i^A = x_j^B\}$, but not $\{(id_i^A, id_j^B) \mid x_i^A = x_j^B\}$. This is because the garbled Bloom filter generated by the original algorithm does not allow us to encode any record ID information. To fix the problem, we modified the algorithm and the new algorithm is shown in Algorithm 1. Accordingly, the query algorithm is modified and is shown in Algorithm 2.

**Algorithm 1:** $BuildGBF(D, n, m, k, H, \lambda)$

**input** : A set $D$, $n$, $m$, $k$, $\lambda$, $H = \{h_0, ...h_{k-1}\}$
**output**: An $(m, n, k, H, \lambda)$-garbled Bloom filter $GBF_D$

1 $GBF_D$= new $m$-element array of bit strings;
2 **for** *i= 0* **to** $m - 1$ **do**
3 $GBF_D$[i]=*NULL*; // NULL is the special symbol that means no value
4 **end**
// encodes $(x_i, id_i)$ rather than just $x_i$ as in the original algorithm
5 **for each** $(x_i, id_i) \in D$ **do**
6 emptySlot = $-1$, finalShare= $id_i$; // $id_i$ is the value to be stored
7 **for** *j=0* **to** *k-1* **do**
8 $z = h_j(x_i)$; // get an index by hashing $x_i$
9 **if** $GBF_D[z]$==*NULL* **then**
10 **if** *emptySlot* ==$-1$ **then**
11 emptySlot=z; // reserve this location for finalShare
12 **else**
13 $GBF_D[z] \xleftarrow{r} \{0,1\}^\lambda$; // generate a new share
14 finalShare=finalShare$\oplus GBF_D[z]$;
15 **end**
16 **else**
17 finalShare=finalShare$\oplus GBF_D[z]$; // reuse a share
18 **end**
19 **end**
20 $GBF_S[emptySlot]$=finalShare; // store the last share
21 **end**
22 **for** *i= 0* **to** $m - 1$ **do**
23 **if** $GBF_D[i]$==*NULL* **then**
24 $GBF_D[i] \xleftarrow{r} \{0,1\}^\lambda$;
25 **end**
26 **end**

Algorithm 1 differs from the original algorithm only in one place: for each record, we split its identifier $id_i$, rather than $x_i$, into $k$ shares on the fly and store the shares in $GBF_D[h_j(x_i)]$ (line 5-21). The intuition is that later the GBF query returns two things: (1) for each $x_i$ whether $x_i$ is encoded in the GBF; (2) for each $x_i$ that is encoded in the GBF, the identifier $id_i$ of the record $R_i$ such that $x_i = f(R_i)$. Thus when $A$ queries the intersection GBF, for each $x_i^A = x_j^B$, it also gets the identifier $id_j^B$ of the record from which $x_j^B$ is extracted. Since $A$ knows the identifier of its record from which $x_i^A$ is extracted, now $A$ can output $(id_i^A, id_j^B)$.

**Algorithm 2:** $QueryGBF(GBF_D, x, k, H)$

**input** : A garbled Bloom filter $GBF_D$, an element $x$, $k$, $H = \{h_0, ...h_{k-1}\}$
**output**: the identifier of $R_i$ for some $R_i \in D$ if $x = f(R_i)$ , NULL otherwise

1 $recovered = \{0\}^\lambda$;
2 **for** *j=0* **to** *k-1* **do**
3 $z = h_j(x)$;
4 $recovered = recovered \oplus GBF_D[z]$;
5 **end**
// We assume that identifiers can be distinguished from random strings.
6 **if** $recovered$ *is an identifier* **then**
7 return $recovered$;
8 **else**
9 return NULL;
10 **end**

## 3.4 PRL by Oblivious Bloom Intersection

The idea of the protocol is to compute and transfer a GBF that contains only the matching record identifiers to $A$ in an oblivious way. The protocol runs as follows:

1. A's private input is $D_A$, $B$'s private input is $D_B$. The auxiliary inputs include the security parameter $\lambda$, the maximum database size $n$, the Bloom filter parameters $m, k$ and $H = \{h_0, ..., h_{k-1}\}$. The parameter $k$ is set to be the same as the security parameter $\lambda$ to make false positive negligible.
2. $A$ generates an $(m, n, k, H)$-BF that encodes its records in $D_A$. To do so, $A$ first produces an empty set $S_A$, then $A$ computes for each $R_i^A \in D_A$, $x_i^A = f(R_i^A)$ and puts $x_i^A$ into $S_A$. Then $A$ encodes $S_A$ into an $(m, n, k, H)$-BF $BF_{S_A}$.
3. $B$ generates an $(m, n, k, H, \lambda)$-GBF that encodes its records in $D_B$. To do so, $B$ first produces an empty set $S_B$, then $B$ computes for each $R_i^B \in D_B$, $x_i^B = f(R_i^B)$ and puts $(x_i^B, id_i^B)$ into $S_B$. Then $B$ encodes $S_B$ into an $(m, n, k, H, \lambda)$-GBF $GBF_{S_B}$ using Algorithm 1.
4. $A$ uses its Bloom filter as the selection string and acts as the receiver in an $OT_\lambda^m$ protocol. $B$ acts as the sender in the OT protocol to send $m$ pairs of $\lambda$-bit strings $(x_{i,0}, x_{i,1})$ where $x_{i,0}$ is a uniformly random string and $x_{i,1}$ is $GBF_{S_B}[i]$. For $0 \le i \le m-1$, if $BF_{S_A}[i]$ is 0, then $A$ receives a random string, if $BF_{S_A}[i]$ is 1 it receives $GBF_{S_B}[i]$. The result is $GBF_{match}$.
5. $A$ finds the matches by querying all elements in $S_A$ against $GBF_{match}$ .

At the end of step 4, $A$ receives a new garbled Bloom filter $GBF_{match}$. This GBF encodes only identifiers of records in $D_B$ that match records in $D_A$. Let's see why this is the case: for any matching record pair $(R_i^A, R_j^B)$, we have $x_i^A = x_j^B$ where $x_i^A = f(R_i^A)$ and $x_j^B = f(R_j^B)$. Note that $A$ and $B$ use the same set of hash functions $H$ when building $BF_{S_A}$ and $GBF_{S_B}$ and the hash functions are deterministic. Therefore, because $x_i^A = x_j^B$, for any $h_l \in H$ we have $z_l = h_l(x_i^A) = h_l(x_j^B)$. In $GBF_{S_B}$, each $GBF_{S_B}[z_l]$ is a share of $id_j^B$ and it will be transferred to $A$ because $BF_{S_A}[z_l] = 1$. So all shares of $id_j^B$ will be preserved in $GBF_{match}$ and by querying $GBF_{match}$, $A$ can recover $id_j^B$. For any non-matching record pair $(R_i^A, R_j^B)$, since $x_i^A \neq x_j^B$, we have with a high probability $h_l(x_i^A) \neq h_l(x_j^B)$, therefore with an overwhelming probability, at least one share of $id_j^B$ will not be transferred to $A$. Instead, a random string will be transferred. Then when $A$ queries $GBF_{match}$, $id_j^B$ cannot be recovered because at least one share is missing.

This protocol is secure in the semi-honest model [10]. We have the following theorem:

THEOREM 1. *Let $f_\equiv$ be the exact record linkage function defined as: $f_\equiv(D_A, D_B) = (\{(id_i^A, id_j^B) \mid x_i^A = x_j^B\}, \Lambda)$, where $\Lambda$ denotes the empty string. If OBI is secure in the semi-honest model, then the exact PRL protocol in Section 3.4 securely computes $f_\equiv$ in the semi-honest model.*

As we can see, the only modification we have made to OBI is the content of the garbled Bloom filter. The modification does not affect the security of OBI, so the proof in [7] still holds. Due to lack of space, the proof of this theorem (and also the proof of Theorem 2) is omitted and will appear in the full version

# 4. THE APPROXIMATE PRL PROTOCOL

In this section, we present the approximate PRL protocol. An approximate PRL protocol matches records based on similarity. This is useful when the data is not "clean", i.e. contains variations and errors such as typos. To design an approximate PRL protocol, the first challenge is how to compare the similarity of two records. Also unlike exact PRL protocols that operate on clean data, for which we

usually can assume that any record matches at most one record in the other database, here we must allow one-to-many matches.

## 4.1 Q-gram Based String Comparison

In many cases, record matching can be done by comparing field values as strings. For example, to compare the values in fields such as name, address, telephone number and date of birth. One widely accepted string comparison mechanism is to split two input strings into short sub-strings of length $q$ characters (called $q$-grams) using a sliding window approach, then calculate a distance using the $q$-gram representation of the strings [21]. The number $q$ is usually small, e.g. 2 or 3. Q-gram based string comparison has been used in a few approximate PRL protocols [5, 20].

Given a string, we can construct a set that contains all q-grams of it. For example, the set for all 2-grams of `smith` is $\{sm, mi, it, th\}$. Given two q-gram sets of two strings, a popular similarity metric is the Jaccard distance [16]. For two set $S_1$ and $S_2$, the Jaccard distance is calculated as:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

The Jaccard distance always lies between 0 and 1.

## 4.2 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [13] is a method to hash data items such that the hash values will collide with high probability if the data items are similar, while dissimilar ones do not. It has been extensively used for similarity search in the information retrieval community. LSH is based on locality sensitive function families. Formally, let $d_1 < d_2$ be two distances according to some metric $d$, and $p_1, p_2$ be two probabilities, A family $G$ of functions is said to be $(d_1, d_2, p_1, p_2)$-sensitive if for every $g$ in $G$ and any $x, y$, the following holds:

- If $d(x, y) \leq d_1$, then the probability that $g(x) = g(y)$ is at least $p_1$.
- If $d(x, y) \geq d_2$, then the probability that $g(x) = g(y)$ is at most $p_2$.

Given a family of $(d_1, d_2, p_1, p_2)$-sensitive functions, we can compose the functions to obtain a new locality sensitive function family that is $(d_1, d_2, p_1', p_2')$-sensitive [17]. Usually $p_1' > p_1$ and $p_2' < p_2$ so false negative and false positive are reduced. For example, if we have a family $G$ of $(d_1, d_2, p_1, p_2)$-sensitive functions, we can obtain a new family $G'$ of $(d_1, d_2, 1 - (1 - p_1^\alpha)^\beta, 1 - (1 - p_2^\alpha)^\beta)$-sensitive functions by forming a composite hash family from an $\alpha$-AND-construction followed by a $\beta$-OR-construction. The $\alpha$-AND-construction results in a composite hash function $\tilde{g} = (g_1, g_2, \ldots, g_\alpha)$ which is the combination of $\alpha$ random functions from $G$. We say $\tilde{g}(x) = \tilde{g}(y)$ if and only if $\forall j, g_j(x) = g_j(y)$ where $1 \leq j \leq \alpha$. The $\beta$-OR-construction results in a composite hash function $g' = (\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_\beta)$. We say $g'(x) = g'(y)$ if and only if $\exists j, \tilde{g}_j(x) = \tilde{g}_j(y)$ where $1 \leq j \leq \beta$. Readers can find more details about the constructions in [17].

There exist effective locality sensitive families for several distance measures, e.g. Jaccard distance, Hamming distance, Euclidean Distance and Cosine distance. The locality sensitive family for Jaccard distance is called MinHash [3]. Let $\Delta$ be the domain of all possible elements of a set, $P$ be a random permutation on $\Delta$, $P[i]$ be the element in $i$th position of $P$ and $min$ be a function that returns the minimum of a set of numbers. Then MinHash function of a set $S$ under $P$ is defined as:

$$h_P(S) = min(\{i \mid 1 \leq i \leq |\Delta| \wedge P[i] \in S\})$$

To compute the MinHash value over a string, we can use the $q$-gram set of the string as the input of the MinHash function. The MinHash family is the set of MinHash functions under different random permutations on $\Delta$. The MinHash family is $(d_1, d_2, 1 - d_1, 1 - d_2)$-sensitive for $0 \leq d_1 < d_2 \leq 1$. In the protocol we will use LSH obtained from the MinHash function family. To control the rate of false positives and false negatives, we always use composite LSH functions that are parameterized by $(\alpha, \beta)$, which means the LSH function is composed by an $\alpha$-AND-construction of MinHash functions, followed by a $\beta$-OR-construction.

## 4.3 LSH Hashtables

As we said at the beginning of this section, there are two problems we need to address in building an approximate PRL protocol: matching by similarity and allowing one-to-many matches. We solve the problems by first building LSH hashtables and then encoding hashtable buckets into a BF or a GBF.

The hashtable structure is quite similar to normal hashtables: it is an array of buckets, each bucket contains a list of values. Given a key-value pair, the key is hashed by a LSH function $g$ and then the value is put into buckets based on the hash result. We want to achieve the following goal by using the hashtable: for any two pairs $(k_1, v_1), (k_2, v_2)$, if $g(k_1) = g(k_2)$, then we can find in the hashtable a bucket that contains both $v_1$ and $v_2$; if $g(k_1) \neq g(k_2)$, then we cannot find in the hashtable a bucket that contains both $v_1$ and $v_2$. Because LSH is different from normal hash functions, we also require each bucket in the hashtable to have a bucket identifier ($bid$). The $bid$ is used when assigning values into buckets. The bucket identifier depends on the LSH function. In our protocol, the LSH functions we use are composite and the final composition is through a $\beta$-OR-construction, i.e. $g = (g_1', g_2', \ldots, g_\beta')$ such that each $g_i'$ is an LSH by $\alpha$-AND-construction. Then the hash value produced by $g$ is a $\beta$-tuple $g(x) = (g_1'(x), g_2'(x), \ldots, g_\beta'(x))$, and the bucket id is the sub-hash value concatenated with the hash index number, i.e. $g_1'(x)||1$, $g_2'(x)||2$, …, $g_\beta'(x)||\beta$. For any $(k, v)$ pair, we can hash $k$ to get $\beta$ bids $g_1'(k)||1$, $g_2'(k)||2$, …, $g_\beta'(k)||\beta$, then $v$ will be put into all $\beta$ buckets that have the bids. This is because the collision semantics of the OR-construction requires *at least one* value in the $\beta$-tuple to be matched. The concatenated index number prevents collisions caused by equal hash values produced by different sub-hash functions, i.e. we do not want $v_1, v_2$ to be put in the same bucket if for example $g_1'(k_1) = g_\beta'(k_2)$ because that does not mean $g(k_1) = g(k_2)$.

## 4.4 Approximate PRL

In the approximate PRL protocol, $A$ and $B$ use a locality sensitive hash function to build hashtables. For each record $R_i$ in their database, $A$ and $B$ use $f(R_i)$ as the key and put $id_i$ in their hashtables. After doing so, $B$ constructs a set $S_B$ such that each element in $S_B$ is a tuple $(a, b)$ where $a$ is the $bid$ of a non-empty bucket in the hashtable, and $b$ is the concatenation of the records identifiers in the bucket. For example if there are three records $R_i^B, R_j^B, R_k^B$ in a bucket, then there is an entry $(x, id_i^B||id_j^B||id_k^B)$ in $D$ such that $x$ is the $bid$ of the bucket. Then $S_B$ is used as input to build the $GBF_{S_B}$ using algorithm 1 [1]. On the other hand, on $A$'s side, $A$ builds $BF_{S_A}$ using $bid$s of a non-empty bucket in its own hashtable. Then they use oblivious transfer so $A$ obtains a GBF that contains the matching record identifiers. $A$ then queries the GBF to find all matching records. The protocol is as follows:

1. A's private input is $D_A$, $B$'s private input is $D_B$. The aux-

[1]Without loss of generality, we assume that the concatenated identifiers can be encoded using a $\lambda$-bit string.

iliary inputs include the security parameter $\lambda$, the maximum database size $n$, the Bloom filter parameters $m, k$ and $H = \{h_0, ..., h_{k-1}\}$. The parameter $k$ is set to be the same as the security parameter $\lambda$. An LSH function $g$ agreed by the two parties.

2. $A$ encodes its database records into a LSH hashtable using the LSH function $g$. To do so, $A$ computes for each $R_i^A \in D_A$, $x_i^A = f(R_i^A)$ and adds $(x_i^A, id_i^A)$ into the hashtable as described in Section 4.3. Similarly, $B$ encodes its database records into a LSH hashtable using the same LSH function.
3. $B$ constructs a set $S_B$ as follows: initially $S_B$ is empty, then for each non-empty bucket in its hashtable, put $(a, b)$ into $S_B$ where $a$ is the $bid$ of this bucket and $b$ is the concatenation of all values in the buckets. After all buckets in the hashtable have been processed, $B$ generates $GBF_{S_B}$ that encodes $S_B$ using algorithm 1.
4. $A$ generates a BF that encodes the $bid$ of every non-empty bucket in its hashtable. To do so, $A$ first produces an empty set $S_A$, then for each bucket in the hashtable, if it is not empty then its $bid$ is put into $S_A$. Then $A$ encodes $S_A$ into $BF_{S_A}$.
5. $A$ uses its Bloom filter as the selection string and acts as the receiver in an $OT_\lambda^m$ protocol. $B$ acts as the sender in the OT protocol to send $m$ pair of $\lambda$-bit strings $(x_{i,0}, x_{i,1})$ where $x_{i,0}$ is a uniformly random string and $x_{i,1}$ is $GBF_{S_B}[i]$. For $0 \leq i \leq m-1$, if $BF_{S_A}[i]$ is 0, then $A$ receives a random string, if $BF_{S_A}[i]$ is 1 it receives $GBF_{S_B}[i]$. The result is $GBF_{match}$.
6. $A$ finds the matches by querying all elements in $S_A$ against $GBF_{match}$ . For each positive query, $A$ can get a list of record identifiers of records in $D_B$ from the query result, and get a list of record identifiers of records in $D_A$ from its hashtable using the $bid$ being queried. Then $A$ computes the Cartesian product of the two lists and put the result into the result set.

The correctness is obvious: in the last step if a query against $GBF_{match}$ is positive, then the buckets in $A$'s hashtable and $B$'s hashtable with the $bid$ being queried are both non-empty. Since $A$ and $B$ use the same LSH function, records sharing the same $bid$ implies their LSH values are equal, thus they are similar. Then the records in the two databases should be linked.

This protocol is also secure in the semi-honest model. The security follows directly the security of the OBI protocol. We have the following theorem:

THEOREM 2. *Let $g$ be a LSH function, $f_\approx$ be the approximate record linkage function defined as: $f_\approx(D_A, D_B) = (\{(id_i^A, id_j^B) \mid g(x_i^A) = g(x_j^B)\}, \Lambda)$. If OBI is secure in the semi-honest model, then the approximate PRL protocol in Section 4.4 securely computes $f_\approx$ in the semi-honest model.*

# 5. EXPERIMENTS AND EVALUATION

In this section, we evaluate our PRL protocols in terms of efficiency and accuracy. The protocols were implemented in C. Our experimental datasets were created by FEBRL [4], a data generator that can generate records with realistic characteristics and can accurately model typographic, phonetic and OCR errors. FEBEL has been used widely in (non-private) record linkage research. All records generated in our experiments contain 12 attributes such as personal identities, full resident information, social security ID and so on. We use all of those attributes for matching records. The attribute values are treated as strings. We use the concatenation of the attribute values in the exact PRL protocol. In the approximate PRL protocol, for each record we creates a 2-gram set from the strings as the input to the LSH function. Our experiments were carried out on two Mac computers: party A ran on a Macbook Pro with a 2.2 GHz quad-core CPU and 16 GB RAM, party B ran on a MacPro with two 2.4 GHz six-core CPUs and 32 GB RAM.

| Record number ($10^3$) | Exact PRL | Approximate PRL | | | | | |
|---|---|---|---|---|---|---|---|
| | | $(\alpha, \beta)$=(10,16) | | $(\alpha, \beta)$=(16,16) | | $(\alpha, \beta)$=(16,32) | |
| | | Offline | Online | Offline | Online | Offline | Online |
| 10 | 0.49 | 0.62 | 3.28 | 0.86 | 3.4 | 1.75 | 6.60 |
| 100 | 4.6 | 6.66 | 34.13 | 9.30 | 33.24 | 18.61 | 66.55 |
| 1000 | 48 | 76.08 | 352.4 | 101.72 | 347.88 | 208.04 | 672.4 |

Table 1: Overall Execution Time (Sec)

## 5.1 Efficiency

In this section we evaluate the efficiency of our protocols. We first measured the overall execution time of the exact PRL protocol. The result is shown in Table 1. We used databases with 10 thousand, 100 thousand and 1 million records in the experiment. The running time is almost linear in the size of the database. The performance is quite close to the result reported in [7].

The approximate PRL protocol splits record matching into two phases: offline and online. In the offline phase, each party independently processes its database to create a LSH hashtable as described in Section 4.3. Then in the online phase, the hashtable is used in oblivious Bloom intersection to generate matching pairs. In the experiment, we varied the size of databases and also parameters $(\alpha, \beta)$ for the LSH function. Then for each combination, we measure the execution time of the offline and online phases. The results are shown in Table 1 . The sizes of the databases are 10,000, 100,000 and 1,000,000. Three pairs of $(\alpha, \beta)$ used are (10,16), (16,16) and (16,32).

We can see from Table 1 that the total execution time increases almost linear in the size of the database. This is as expected because the time complexity of both phases is $O(n)$. From Table 1 we can also see that the choice of $(\alpha, \beta)$ has impact on performance. For example, when $(\alpha, \beta)$ is (10,16), (16,16) and (16,32), the total execution time is 404.08, 429.72 and 864.04 seconds respectively on databases with 1 million records. We can also see from the figure that $\alpha$ affects only offline execution time while $\beta$ affects both offline and online execution time. When $\alpha = 16$, the offline execution time only slightly higher (about 10%) than when $\alpha = 10$. When $\beta = 32$, the offline and online execution time are almost doubled comparing to when $\beta = 16$. This is because increasing $\alpha$ and $\beta$ will increase the cost of LSH, therefore the offline phase takes longer to finish. Increasing $\beta$ also results in a bigger hashtable with more buckets, in turn the sizes of Bloom filter and garbled Bloom filter become bigger and thus the online phase requires more time.

## 5.2 Accuracy

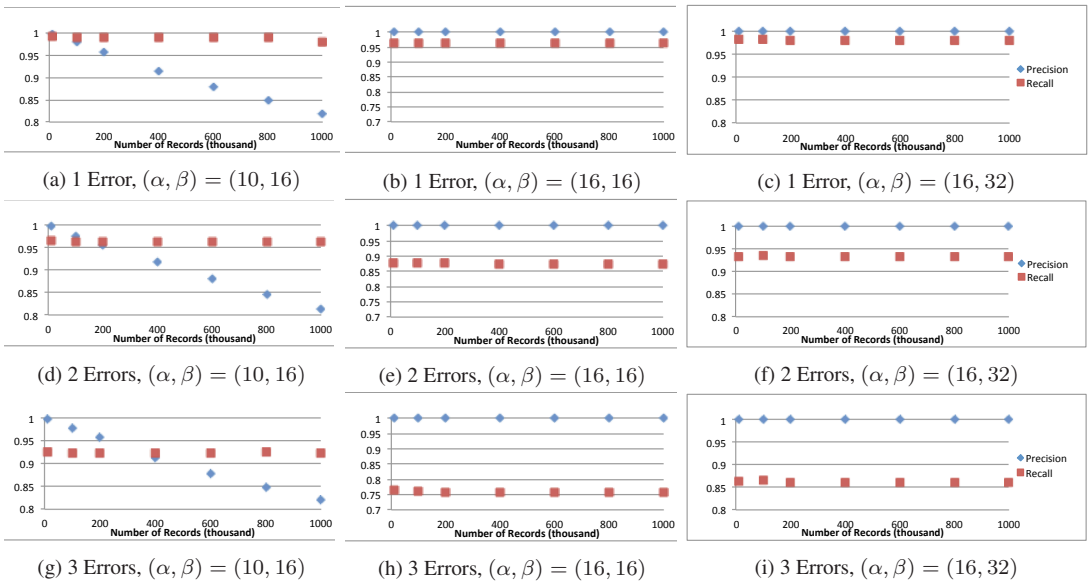
For the approximate PRL protocol, another very important evaluation criterion is accuracy. The accuracy of our approximate PRL protocol is assessed by precision and recall. These metrics are based on the notions of true positive (TP), false positive (FP) and false negative (FN).

$$precision = \frac{\sum TP}{\sum TP + \sum FP}$$

$$recall = \frac{\sum TP}{\sum TP + \sum FN}$$

A correct match is a TP, a wrong match is an FP, and a missed match is an FN.

In Figure 1, we show the accuracy measured with different number of errors and different $(\alpha, \beta)$. In the experiment, we created

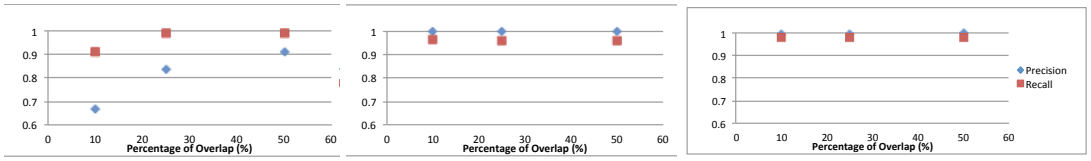



(a) 1 Error, $(\alpha, \beta) = (10, 16)$ (b) 1 Error, $(\alpha, \beta) = (16, 16)$ (c) 1 Error, $(\alpha, \beta) = (16, 32)$

(d) 2 Errors, $(\alpha, \beta) = (10, 16)$ (e) 2 Errors, $(\alpha, \beta) = (16, 16)$ (f) 2 Errors, $(\alpha, \beta) = (16, 32)$

(g) 3 Errors, $(\alpha, \beta) = (10, 16)$ (h) 3 Errors, $(\alpha, \beta) = (16, 16)$ (i) 3 Errors, $(\alpha, \beta) = (16, 32)$

Figure 1: Accuracy with Different numbers of errors and Different $(\alpha, \beta)$




(a) $(\alpha, \beta) = (10, 16)$ (b) $(\alpha, \beta) = (16, 16)$ (c) $(\alpha, \beta) = (16, 32)$

Figure 2: Accuracy with Different Overlap Percentages

several databases with different sizes, then for each database we created a modified duplicate. Each record from the original database is copied to the duplicate database with 1, 2 or 3 modifications. The modifications simulate errors such as typos, insertion, deletion, or switch of positions. We measure precision and recall with different $(\alpha, \beta)$. The results of experiments with 1 error are show in Figure 1a, 1b and 1c. We can see the protocol generally performs well with recall always higher than 0.95 and precision higher than 0.80. We can see from Figure 1a that with a small $\alpha$, we might get large amount of false positive if the database size is big. This is because smaller $\alpha$ means the LSH function is more likely to produce equal hash values from distinct records. Therefore false positive increases. To deal with it, we can increase $\alpha$. In our experiments, set $\alpha = 16$ would be enough to make false positive almost negligible. The other results in Figure 1 show that the number of errors has little effect on precision, but might affect recall. Nevertheless, if we increase $\beta$, then we can get a higher recall. Lower recall means more false negative. Higher $\beta$ means the LSH function is more likely to produce equal hash values from similar records. Therefore by increasing $\beta$ we can get a better recall.

In Figure 2, we show the accuracy measured with different number of overlapping records and different $(\alpha, \beta)$. In the experiments, we set the size of databases to 200 thousand. In each of the experiment, we create the duplicate database as follows: we take a portion of the records in the original database (10%, 25% or 50%), copy them, with 1 modification, into an empty database, and then insert new records that are not in the original database until the new database contains also 200 thousand records. As we can see, when $\alpha$ is small, the precision is affected by the percentage of overlap. This is because in case of less overlapped databases, more records are irrelevant. Therefore the actual number of false positive is more significant comparing to the number of true positive, which takes down the precision. With a higher $\alpha$ value, we can get a better precision.

In summary the accuracy of our approximate protocol may be affected by multiple factors, but by setting $(\alpha, \beta)$ properly, we can ensure the accuracy is within a satisfactory bound.

## 5.3 Comparison

The performance of our exact PRL is better than all known protocols. This is mainly due to the efficiency of the OBI protocol. As reported in [7], the OBI protocol is orders of magnitude faster than all existing private set intersection protocols. Because currently the main stream exact PRL protocols are all based on private set intersection, it is not surprising that our protocol is much faster. However, it is possible that the other protocols can also use OBI, thus achieve similar performance as our protocol.

For the approximate PRL protocol, we compared our protocol performance and accuracy against two most prominent protocols, namely Vatsalan's protocol [22] and Yakout's protocol [24]. In the comparison, we use the results reported by the authors. The hardware used in their experiments is similar to or better than ours, so we believe the performance difference caused by hardware should be small thus can be ignored.

In our experiments, we set security at 80-bit. Both our protocol and Vatsalan's protocol have linear (in the size of database) computational complexity. When processing databases with 1 million records, our protocol needs 429 seconds when $(\alpha, \beta)$ is (16,16) and 864 seconds when $(\alpha, \beta)$ is (16,32). For Vatsalan's protocol, the time reported by the authors is approximately 100 seconds. How-

ever this time is measured without any encryption. Since the most costly part of the protocol is the commutative encryption, which involves public key operations, the actual execution time should be much longer with encryption. In our experience, an RSA-base commutative encryption scheme at 80-bit security normally needs more than an hour to execute 1 million encryption operations. Another problem of Vatsalan's protocol is that the number of the attributes affects performance greatly. The more attributes to be used in matching, the longer it will take to do the record linking. While in our protocol, the impact of the number of attributes is negligible. As for the accuracy, the precision of Vatsalan's protocol is always above 0.99. However, the recall is consistently low (less than 0.7). In our experiments, the lowest value of recall we measured is 0.759, which is still better Vatsalan's best recall value. And as we discussed, the recall value can be improved by adjusting $(\alpha, \beta)$.

The computational complexity of Yakout's protocol is $O(n^2)$ where $n$ is the size of the databases. The security strength of the protocol is hard to estimate because no detailed analysis was provided. When the database size is 100 thousand and with the minimal privacy parameter ($k = 4$), Yakout's protocol takes about 2.5 minutes, which is 3.5 times ($(\alpha, \beta)$ = (16,16)) or 1.75 times ($(\alpha, \beta)$ = (16,32)) of our protocol. When the database size is 1 million, the ratios become 28.5 ($(\alpha, \beta)$ = (16,16)) or 14.1 ($(\alpha, \beta)$ = (16,32)). In their paper, the accuracy tests only used small datasets with 1000 and 5000 records. They converts string records to vectors then embeds vectors into a metric space. The authors mentioned that if there is a large variations of records' length, then this method is not very accurate and would result in bad matching quality. This is not a problem in our protocol.

# 6. CONCLUSION

In this paper, we investigated the PRL problem and proposed an exact PRL protocol and an approximate PRL protocol. The exact PRL protocol is an extension of the OBI protocol. The approximate PRL protocol is built by combining LSH functions with the exact PRL protocol. We implemented the two protocols and evaluated the protocols in terms of efficiency and accuracy. We also compared our protocols against other PRL protocols and the comparison shows our protocols are much more efficient.

In terms of future work, we would like to validate our protocols with real world data. Another direction would be to implement and evaluate LSH functions for other similarity metrics and incorporate them into our approximate PRL protocol.

**Acknowledgement** Zikai Wen is supported by an undergraduate research internship from the University of Strathclyde, Changyu Dong is supported by a science faculty starter grant from the University of Strathclyde. We thank the anonymous reviewers for their helpful comments.

# 7. REFERENCES

[1] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD Conference*, pages 86–97, 2003.

[2] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES*, pages 39–44, 2003.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.

[4] P. Christen and A. Pudjijono. Accurate synthetic generation of realistic personal information. In *PAKDD*, pages 507–514, 2009.

[5] T. Churches and P. Christen. Some methods for blindfolded record linkage. *BMC Med. Inf. & Decision Making*, 4:9, 2004.

[6] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, pages 143–159, 2010.

[7] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *ACM Conference on Computer and Communications Security*, 2013.

[8] L. Dusserre, C. Quantin, and H. Bouzelat. A one way public key cryptosystem for the linkage of nominal files in epidemiological studies. *Medinfo*, 8 Pt 1:644–7, 1995.

[9] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[10] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[11] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[12] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[14] L. Kissner and D. X. Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[15] M. Kuzu, M. Kantarcioglu, E. Durham, and B. Malin. A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In *PETS*, pages 226–245, 2011.

[16] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[17] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge.

[18] P. Ravikumar and S. E. Fienberg. A secure protocol for computing string distance metrics. In *In PSDM held at ICDM*, pages 40–46, 2004.

[19] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In *SIGMOD Conference*, pages 653–664, 2007.

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

[21] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[22] D. Vatsalan, P. Christen, and V. S. Verykios. An efficient two-party protocol for approximate matching in private record linkage. In *AusDM '11*, pages 125–136, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

[23] D. Vatsalan, P. Christen, and V. S. Verykios. A taxonomy of privacy-preserving record linkage techniques. *Inf. Syst.*, 38(6):946–969, 2013.

[24] M. Yakout, M. J. Atallah, and A. K. Elmagarmid. Efficient and practical approach for private record linkage. *J. Data and Information Quality*, 3(3):5, 2012.