Cryptographically Secure Private Record Linkage Using Locality-Sensitive Hashing

Ruidi Wei
University of Waterloo
Waterloo, Ontario, Canada
r33wei@uwaterloo.ca

Florian Kerschbaum
University of Waterloo
Waterloo, Ontario, Canada
florian.kerschbaum@uwaterloo.ca

ABSTRACT

Private record linkage (PRL) is the problem of identifying pairs of records that approximately match across datasets in a secure, privacy-preserving manner. Two-party PRL specifically allows each of the parties to obtain records from the other party, only given that each record matches with one of their own. The privacy goal is that no other information about the datasets should be released than the matching records. A fundamental challenge is not to leak information while at the same time not comparing all pairs of records. In plaintext record linkage this is done using a *blocking* strategy, e.g., locality-sensitive hashing. One recent approach proposed by He et al. (ACM CCS 2017) uses locality-sensitive hashing and then releases a provably differential private representation of the hash bins. However, differential privacy still leaks some, although provable bounded information and does not protect against attacks, such as property inference attacks. Another recent approach by Khurram and Kerschbaum (IEEE ICDE 2020) uses locality-preserving hashing and provides cryptographic security, i.e., it releases no information except the output. However, locality-preserving hash functions are much harder to construct than locality-sensitive hash functions and hence accuracy of this approach is limited, particularly on larger datasets. In this paper, we address the open problem of providing cryptographic security of PRL while using locality-sensitive hash functions. Using recent results in oblivious algorithms, we design a new cryptographically secure PRL with locality-sensitive hash functions. Our prototypical implementation can match 40000 records in the British National Library/Toronto Public Library and the North Carolina Voter Registry datasets with 99.3% and 99.9% accuracy, respectively, in less than an hour which is more than an order of magnitude faster than Khurram and Kerschbaum's work at a higher accuracy.

PVLDB Reference Format:
Ruidi Wei and Florian Kerschbaum. Cryptographically Secure Private Record Linkage Using Locality-Sensitive Hashing. PVLDB, 17(2): 79 - 91, 2023.
doi:10.14778/3626292.3626293

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/rd-wei/Secure-PRL-with-LSH.

Proceedings of the VLDB Endowment, Vol. 17, No. 2 ISSN 2150-8097.
doi:10.14778/3626292.3626293

1 INTRODUCTION

Data relevant to socially beneficial analyses is often distributed across several data sources. Hence, before the data can be used it needs to be linked together. This problem is known as *record linkage* (RL). It is applied even in privacy-sensitive domains, such as financial or health data [26, 30, 33].

A challenge to RL linkage is data errors or schema differences that may lead to different representations of the same entity. Hence, RL needs to apply an approximate matching algorithm. Compared to exact matching, this leads to another challenge, namely selecting which pairs of records to compare. Naively comparing all pairs is too inefficient even for modestly sized datasets. RL hence applies *blocking* strategies to only compare a subset of the pairs [23]. A commonly used blocking strategy is locality-sensitive hashing (LSH) [39]. A LSH puts similar, but not necessarily identical records in the same hash bin. Good LSHs exist even for "difficult" distance functions such as the edit distance [4, 5].

However, this approach is difficult to implement when the records need privacy protection. The problem of providing privacy of the records (that are not matching) while performing RL is called private record linkage (PRL) or privacy-preserving record linkage. Ideally, one wants to compare the records (probabilistically) encrypted, i.e., using either fully homomorphic encryption or (interactive) secure multi-party computation which provide strong, provable security guarantees in an established formal model. A matching algorithm that securely computes a distance between two records (and compares it to a threshold) can be efficiently constructed using multi-party computation [15, 22]. However, how to encrypt the output of the blocking algorithm, i.e., the distribution of elements over the bins of an LSH is not clear, since it is the number of elements that leaks information and not their contents. Trivial constructions are inefficient. An LSH may hash all elements into the same bin and hence padding the bin lengths to maximum possible length is even worse than naively comparing all pairs. He et al. [17] propose to use differential private padding, i.e., adding a random number of elements drawn from a shifted Laplacian distribution. This provides differential privacy for the input datasets, but only in weakened form, since the output of the PRL also leaks information that may violate standard differential privacy. Furthermore, it is well accepted that differential privacy does not protect against certain attacks, such as property inference attacks [2, 13]. A property inference could, e.g., try to determine the size or presence of a group in an ethnic, gender or religious minority in the datasets. While differential privacy provides protection for an individual, group privacy properties – while provably preserved – can be significantly weakened, particularly in large datasets.

It is hence advisable to strive for stronger privacy protection, such as cryptographic security, which leaks no information to a computationally bounded adversary. Khurram and Kerschbaum [25] present the first cryptographic secure, efficient PRL scheme with less than quadratic comparisons. However, they use locality-preserving hash (LPH) [20] functions for blocking. LPH are much harder to construct than LSH which questions their accuracy for large datasets or datasets with complex distance metrics, even edit distance. We provide a detailed comparison of the formal security definitions of He et al.'s, Khurram and Kerschbaum's, and our approach in Appendix A.

This leaves an important problem open: Is it possible to construct a cryptographically secure PRL scheme using locality-sensitive hashing? We address this open problem in this paper and provide a new construction. This construction is non-trivial which is why it was believed for a long time that such a construction could not exist [17]. Our construction uses several recent advances in oblivious algorithms. None of these advances by themselves could achieve the desired outcome, but in their combination they do. First, we use the frequency smoothing technique of Grubbs et al. [16] to provide a uniformly smoothed hash table. This smoothing replaces the padding by He et al. and hence the need for differential privacy. This technique not only pads bins, but may split them and ensures that the total padding at most doubles the number of elements. The splitting of bins deviates from any previous blocking approach in a PRL protocol and it also introduces a problem not present in any previous construction, namely that multiple bins need to be privately linked together. This can be solved by oblivious join algorithms which exist for a long time, but for which the first practical construction was presented by Krastnikov et al. [28]. However, the algorithm cannot be used in a multi-party computation as described by Krastnikov et al. We need to modify it to run in a logarithmic number of rounds to ensure efficiency over a network instead of a trusted execution environment. Hence, the main contribution of this paper is a new PRL protocol that combines these two techniques to achieve the first cryptographically secure PRL using LSH.

We have to recognize that even with cryptographic security, the runtime of our protocol leaks the number of comparisons made. If this number is not leaked, n^2 comparisons are necessary and the protocol is not efficient. The number of comparisons is, however, dataset-dependent when using LSH. Fortunately, there are a number of mitigating factors. First, we only provide an upper bound on the number of comparisons due to some padding. Second, the number of comparisons is correlated with the number of elements in the output which is necessarily revealed. Third, the number of total comparisons is strictly less information than the distribution of elements over the LSH bins.

We implemented a prototype of the resulting algorithm and evaluated its efficiency and accuracy. Compared to Khurram and Kerschbaum [25], the state-of-the-art in cryptographically secure PRL, we compare 40000 records in each dataset in less than an hour which is an order of magnitude faster. Note that, Khurram and Kerschbaum already improve over the approach by He et al. by more than an order of magnitude while also providing stronger security guarantees. Also, our accuracy of 99.3% and 99.9% on the same datasets that Khurram and Kerschbaum used, is higher.

In summary, our paper makes the following contributions

(1) A new PRL protocol combining two oblivious algorithms [16, 28] in a novel way resulting in cryptographically secure blocking with locality-sensitive hashing.
(2) A security, efficiency and accuracy analysis of our new PRL protocol using a prototypical implementation.

The remainder of the paper is structured as follows. We formally define our problem including security in the next section. In Section 3 we review related work. We describe our protocol in detail in Section 4 and a proof of its security, the evaluation of accuracy and efficiency of its prototypical implementation in Section 5. Finally, in Section 6 we summarize our conclusions. In Appendix A, we provide a detailed comparison of the formal security definitions of He et al.'s, Khurram and Kerschbaum's, and our approach .

2 PROBLEM DEFINITION

We consider the problem of private record linkage (PRL). In PRL, there are two parties, Alice and Bob, which each have a table of records $T_A = \{A_1, \ldots, A_{n_A}\}$ and $T_B = \{B_1, \ldots, B_{n_B}\}$, respectively. We represent a record A_i or B_j in the table as a variable-length string, abstracting from its schema representation. The goal of PRL is to identify which records A_i and B_j represent the same entity, e.g., a person, a book or a product, despite some small differences. This is achieved by comparing A_i and B_j using a distance function Δ. There are plenty of examples of such distance functions in the literature, e.g., the Levenshtein (or edit) distance [31]. The basic algorithm of PRL is to compare pairs A_i and B_j and output them as matching records if $\Delta(A_i, B_j) < \tau$ from some threshold τ. The challenge of efficient PRL is not to compare every pair A_i, B_j in the cross-product of the tables T_A and T_B.

A common way to achieve this is blocking by using locality-sensitive hash (LSH) functions [39]. LSHs have the property that similar elements are likely put into the same bin. However, we want *private* record linkage, i.e., Bob should not learn anything about Alice's input that is not in the output and vice versa. This is challenging, since the number of elements per bin reveals significant information about a party's input. He et al. [17] hence proposed to pad the bins, such that their count is differentially private. In this paper we are concerned to reduce this leakage even further, since differential privacy only protects individual records and not properties of the dataset. For example, despite differentially private protection, this padding would still leak the approximate size or presence of a group in an ethnic, gender or religious minority in the datasets. We provide a detailed comparison of the formal security definitions in Appendix A.

However, note that, completely preventing this leakage, when using LSH is very inefficient. An LSH could put all records into the same bin and hence the worst case complexity is $O(n_A \cdot n_B)$. This would be trivially achievable, but also completely impractical. Hence, we allow the total number of comparisons $c \leq n_A \cdot n_B$ in the protocol to be leaked. The complexity of an efficient record linkage protocol is usually $O(c)$ and hence this number is inevitably leaked from the running time of such a protocol.

We formalize security in the semi-honest model. The semi-honest model is required by and sufficient for many practical applications as it secures the data against inspection by the other party. It is also more efficient than the malicious model and hence practically

deployed [21, 44]. In theory it is possible to convert any protocol secure in the semi-honest model into one secure in the malicious model using Goldreich's compiler [14]. However, due to the cost, this is rarely done.

Let O be the set of matching record pairs, i.e., the output of the protocol. Let $\mathsf{View}_{A/B}$ be the view of Alice and Bob, respectively, i.e., the messages received and the random coins chosen during the protocol execution. We say a protocol is secure in the semi-honest model, if there exists two simulators $\mathsf{Sim}_{A/B}$ for the views $\mathsf{View}_{A/B}$, respectively.

$$\begin{aligned} \mathsf{Sim}_A(T_A, n_B, O, c) &= \mathsf{View}_A(T_A, T_B) \\ \mathsf{Sim}_B(T_B, n_A, O, c) &= \mathsf{View}_B(T_A, T_B) \end{aligned}$$

In summary, our problem is to design an (almost) leakage-free PRL protocol that uses LSH.

3 RELATED WORK

RL was identified as a useful tool in medical research by Dunn in 1946 [11]. Due to the sensitive nature of medical records, privacy is a long-standing research problem in RL [40]. Karakasidis and Verykios [23] have identified two algorithms that need to be secured and composed to build a private record linkage system: blocking and matching.

Blocking is commonly based on LSH, since it allows the use of the entire information of a record and is domain-independent [39]. Steorts et al. [39] provide a comparison of some LSHs.

Private blocking techniques can be categorized by the privacy property they ensure. There are cryptographically secure, differentially private, and ad-hoc blocking techniques. The first two aim for a rigorous, commonly used definition of privacy. A cryptographically secure blocking technique requires a simulation-based proof as in Section 5.1. Khurram and Kerschbaum have provided the first such proof for a blocking technique based on locality-preserving hash functions [25]. However, locality-preserving hash functions are much harder to construct than locality-sensitive hash functions. Al-Lawati et al. [1] also aim for cryptographic security. However, they do not provide a simulation-based proof which seems hard to construct given that they leak a permutation of the LSH bins. Furthermore, they require a third party and their techniques do not efficiently translate to a two-party setting.

Differentially private blocking has been introduced by Inan et al. [19]. The idea is to pad the LSH block sizes with differentially private noise using dummy elements. The approach was later refined by Kuzu et al. [29] and Cao et al. [6]. Cao et al. identified a privacy flaw in the previous protocols leading to a successful attack that they presumably fixed. However, as He et al. [17] point out that the protocol by Cao et al. still does not satisfy differential privacy. Instead, they propose a somewhat weakened definition of differentially privacy that takes the output of the private matching into account. Loosely speaking, they only consider databases neighbours for differential privacy if they produce the same output in the matching. This definition is natural for PRL, since the unmatched records remain differentially private. One could have also defined neighbours as the union of any database combined with the output records (assuming there are no duplicates). However, in general the definition in He et al.'s paper is a strong weakening of differential privacy. Furthermore, differential privacy still leaks information. It is well accepted that differential privacy does not protect against property inference attacks [2, 13]. A property inference could, e.g., try to determine the size or presence of a group in an ethnic, gender or religious minority in the datasets. Humphries et al. [18] and Stadler et al. [38] have further shown that much weaker membership inference attacks may still be feasible despite differential private protection when the data is not uniformly sampled.

Prior to these rigorous approaches, several ad-hoc approaches have been proposed. Karapiperis et al. [24] provide a good overview of these methods. Some of them also use a form of locality-sensitive hashing and combine it with anonymization, e.g., k-anonymization of the data prior to hashing. k-anonymity protects the quasi-identifier of a record, but not its sensitive attributes which may, however, require the most protection during PRL. Other methods use a public reference dataset and bin records according to their proximity to the public records. Note that, although the reference dataset is public, this does not necessarily provide any protection to the blocking of the private records. One has to take into account that the adversary also knows the public dataset and can compute the proximity function.

There are alternatives to blocking in PRL which can also provide cryptographic security. One can compare all pairs, as Yakout et al. [43], but that does not scale to large datasets (even none privately). One can expand the records to (all) possible variations and then use exact matching instead of approximate matching, as Wen and Dong [42], but He et al. [17] already point out that this approach is as inefficient as comparing all pairs. Exact matching can be very efficiently performed using private set intersection (PSI) [12, 37, 44]. In PSI records can be deterministically encrypted using a joint key and then compared in plain which makes blocking obsolete. However, this approach only works for exact matching, since encryption destroys any similarity between records.

Private matching protocols are much easier to implement with cryptographic security. Even the edit distance can be efficiently implemented using two-party computation [22]. However, in the interest of even higher efficiency ad-hoc approaches have been also proposed. Schnell et al. [36] propose to match on Bloom filters computed on keyed hash functions over ngrams. However, there exists a plethora of attacks on this method [41] and retrofitting security is a tedious task.

In this paper, we propose the first cryptographically secure blocking method for private record linkage using LSH. We provide a simulation-based proof (Section 5.1). Such additional security is always a benefit if it does not come at the expense of accuracy or efficiency. Since we use LSH – compared to LPH by Khurram and Kerschbaum – we provide state-of-the-art accuracy. Furthermore, our approach is an order of magnitude faster than the LSH-based PRL by He et al. which only provides differential private protection.

4 PROTOCOL OVERVIEW

In this section, we describe our protocol for private record linkage. We summarize our notation in Table 1. A subscript or superscript of A or B indicates the data structure to be at Alice's or Bob's site, respectively. We first provide an overview and then describe the technical details. Our protocol proceeds in three steps:

Table 1: Common notation

T	Data set
n	Size of data set
R	Alice', Bob's or either record
O	Output
c	Number of matches
t	Length of LSH
ℓ	Number of LSH bins
d	Number of LSH functions
B, B_i	LSH bins, LSH bin numbered i
S, S_i	Smoothed bin, smoothed bin numbered i

(1) Preprocessing: A local step where each party hashes their records to bins, and then smoothes the frequency of the bins.
(2) Secret bin join. A private computation step that does an oblivious join operation on the bins.
(3) Record-to-record comparisons. A private computation step that compares each pair of records inside a joined set of bins.

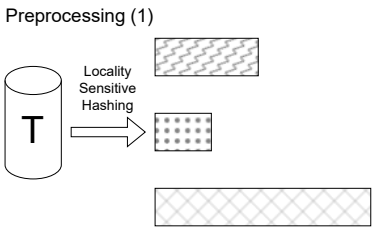


Figure 1: Blocking using an LSH

We can illustrate the steps with a small example. Assume Alice has the data set $T_A = \{Cat, Dog, Airplane\}$ and Bob has the data set $T_B = \{Aeroplane, Bird, Car\}$. We assume the only matching records are "Airplane" and "Aeroplane". In the preprocessing step, Alice hashes her elements into two bins $B_1 = \{Airplane, Cat\}$ and $B_2 = \{Dog\}$ using an LSH. We show a figurative example in Figure 1. We detail the smoothing algorithm in the next section, but after smoothing Alice has four bins: $S^A_{1,tag=1} = \{Airplane\}$, $S^A_{2,tag=1} = \{Cat\}$, $S^A_{3,tag=2} = \{Dog\}$, and $S^A_{4,tag=3} = \{Dummy\}$. Bob locally performs similar steps and ends up with: $S^B_{1,tag=1} = \{Bird\}$, $S^B_{2,tag=1} = \{Aeroplane\}$, $S^B_{3,tag=2} = \{Car\}$, and $S^B_{4,tag=4} = \{Dummy\}$. The smoothing algorithm results in a uniform number of elements in a fixed number of bins and hence leaks no information to the other party. We show a figurative example in Figure 2.

Some bins have been split into several with the same tag. Hence, we can no longer perform a 1:1 bin matching, but need to perform a secret bin join, the second step. In this step all bins $S^{A/B}_{i,tag}$ with the same tag are joined, but without revealing any information about the bins or their tags. Hence, we end up with five joined pairs: $S^A_{1,tag=1} \bowtie S^B_{1,tag=1}$, $S^A_{1,tag=1} \bowtie S^B_{2,tag=1}$, $S^A_{2,tag=1} \bowtie S^B_{1,tag=1}$, $S^A_{2,tag=1} \bowtie S^B_{2,tag=1}$, and $S^A_{3,tag=2} \bowtie S^B_{3,tag=2}$.

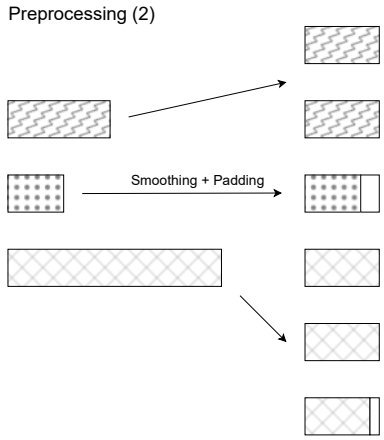


Figure 2: Smoothing of the bins

In the last step, record-to-record comparison, we securely compare all elements in a bin to all elements in a paired bin. One pair of bins ($S^A_{1,tag=1} \bowtie S^B_{2,tag=1}$), contains the elements "Airplane" and "Aeroplane" and the match is found. We show a figurative example of the two last steps in Figure 3.

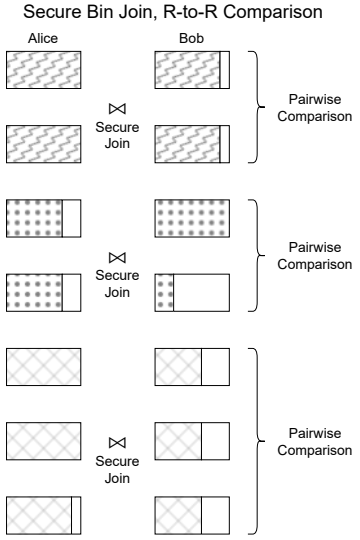


Figure 3: Completing the protocol

4.1 Preprocessing

In this step, each party, Alice or Bob, locally hashes their records in T_A or T_B, respectively, into (local) bins. The frequency of records in each bin is then smoothed to hide the distribution of records before proceeding to private computation. Since this step is symmetric between Alice and Bob and without involvement of the other party, we use T as a notation to mean either T_A or T_B and n to mean n_A or n_B, respectively.

To hash the elements we use a locality sensitive hash (LSH) function. Compared to locality-preserving hash (LPH) functions as used

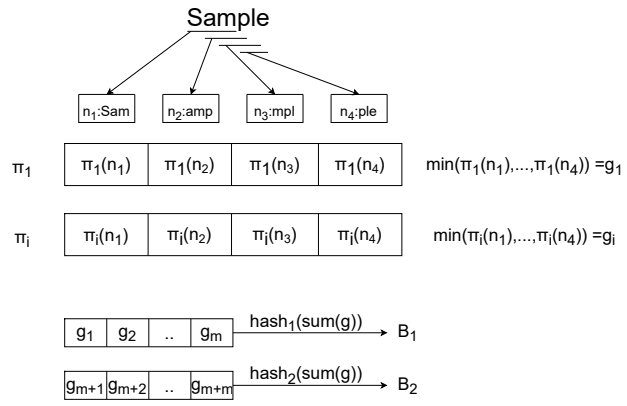


Figure 4: Illustration of the locality sensitive hashing function

by Khurram and Kerschbaum in the first efficient, cryptographically secure PRL protocol, LSH have the advantage that they are available for a wider domain of input ranges and often have lower error rates. There exist several LSH in the literature for different notions of locality and our protocol can use any of them with no further modifications. In the next section, we describe the LSH we use to evaluate accuracy in our experiments.

Locality Sensitive Hash Function. To better compare strings in the presence of errors, we preprocess string values using minhash [4, 5]. Minhash first splits an input string into η overlapping n-grams (substrings of length n). We denote the resulting n-grams α_j, for $j \in [1, \eta]$. Then minhash hashes each n-gram to an integer β_j: $\beta_j = \text{hash}(\alpha_j)$. These integers are randomly permuted to another integer $\gamma_{j,k}$ by t uniformly chosen permutation functions π_k ($k \in [1, t]$), that is $\gamma_{j,k} = \pi_k(\beta_j)$. Minhash then computes the minimum $\theta_k = \min_{j \leq \eta}(\alpha_{j,k})$. The resulting vector $\vec{\theta}$ is an indicator of the similarity of the strings. The shorter the minhash, the more dissimilar the strings can be to result in the same minhash $\vec{\theta}$.

A parameter of the LSH is its number ℓ of bins. Given ℓ LSH bins, $[B_1, \ldots, B_\ell]$, a locality sensitive hash function hashes a record R to an integer i in $[1, \ell]$. Then, we write $R \in B_i$.

We map the minhash vector $\vec{\theta}$ to a uniformly chosen bin B_i ($i \in [1, \ell]$). We compute $\rho = \sum_{k=1}^{t} m_k$. We uniformly choose two 32-bit primes p_1 and p_2. We compute i for B_i as $i = (\rho \cdot p_2 \mod p_1) \mod \ell$.

We summarize this step in Algorithm 1.

Algorithm 1 Blocking(T)

$B \leftarrow \{\varnothing\}^\ell$
for R in T **do**
  $i \leftarrow LSH(R)$
  $B_i \leftarrow B_i \cup \{R\}$
end for
return B

This process is repeated d times, so that we end up with d sets of bins, with ℓ bins in each set. Every string is hashed into one bin for each set, creating d duplicates of it across all the bins.

Frequency Smoothing. After binning elements using the LSH, we independently smooth the frequency of the bins of each of the d replicated sets. Two non-identical sets of records may result in different numbers of elements in a bin and hence may be distinguishable by the other party when used as input to the private computation. Padding each bin to a maximum number results in an inefficient algorithm, since there exist datasets where all records are in the same bin. Hence, each bin would need to be padded to the entire dataset size. He et al. [17] use differentially private padding, but this still leaks information about the set, since differential privacy does not hide properties, e.g., whether the set contains (more) Canadians or Americans, or whether several subjects suffer from a common illness. Instead, we use the frequency smoothing algorithm by Grubbs et al. [16]. This algorithm not only uses padding, but also splits bins with many elements into several bins. It results in a deterministic number of bins with equal and deterministic frequency. We review the algorithm here for completeness.

For a total number of records n, a party first prepares 2ℓ smoothed bins $S_1, ..., S_{2\ell}$, each with size $\sigma = \lceil \frac{n}{\ell} \rceil$. For a bin B_i output by the LSH of size $|B_i| = \alpha\sigma + \beta$, let α be the largest integer such that β is non-negative. A party then puts the α times σ randomly chosen records of B_i into one of α different bins $\{S_j, \ldots, S_{j+\alpha-1}\}$, and the remaining β records of B_i into another bin S_{j+k}. Each bin $\{S_j, \ldots, S_{j+\alpha}\}$ has an associated tag i to show that they are from the same bin B_i used by the LSH. After distributing all bins in B, there are at most ℓ bins in S which are completely empty, since each such bin must be a remainder of a bin in B, and there at most ℓ completely filled bins, because at least n records are required to fill ℓ bins in S. Consequently, all records in B always fit into S (2ℓ bins are always sufficient).

Then a party pads each bin in S to size s using dummy records that deterministically do not match any record in the other party's dataset. A party needs to add $2\ell\lceil \frac{n}{\ell} \rceil - n = O(n)$ such dummy records to pad all bins to size s. We summarize the pseudo-code of this step in Algorithm 2. The function $Noise(Party)$ generates a dummy record that is not in the domain of regular records and not in the domain of dummy records of the other party. The function $Shuffle$ randomly permutes the elements in a set. Applying it to each smoothed bin S_i, ensures that each record R of T (and each dummy record) is at a uniformly random location within its smoothed bin S_i.

4.2 Private Bin Join

After frequency smoothing, we can no longer match bins one-to-one, as e.g., done by He et al. [17], since there can be multiple bins from the same LSH bin. However, these bins have the same tag and we can perform a regular (oblivious) equi-join to select all pairs with the same tag. To do so, we simply view each party's set S as a table and join them on the *tag* attribute. The join needs to be oblivious to not reveal information during the private computation protocol. We use the oblivious join protocol by Krastnikov et al. [28]. However, this protocol has been designed with trusted execution environments in mind and we intend to use it in a multi-party

Algorithm 2 Smoothing(B)

```
S ← 2ℓ empty sets
i ← 1
j ← 1
size ← ⌈n/ℓ⌉
while i ≤ 2ℓ do
    S_j.tag ← i
    m ← min(size, |B_i|)
    k ← 0
    while k < m do
        R ←$ B_i
        B_i ← B_i \ {R}
        S_j ← S_j ∪ {R}
        k ← k + 1
    end while
    j ← j + 1
    if size == |B[i]| then
        i ← i + 1
    end if
end while
for S_i ∈ S do
    while |S_i| < size do
        R ← Noise(Party)
        S_i ← S_i ∪ {R}
    end while
    Shuffle(S_i)
end for
return S
```

computation protocol. Hence, we need to make a few adjustments. In particular, we need to reduce the round complexity of several traversals in the protocol. We do this by replacing linear scans (2ℓ rounds) with parallel scans of depth $\log(\ell) + 1$. We first describe the original algorithm, and then show the changes we made to it to make it suitable for a multi-party computation protocol.

4.2.1 Krastnikov et al.'s Algorithm. Krastnikov et al.'s algorithm proceeds in three steps: Obtaining group dimensions, oblivious distribution, and oblivious expansion.

Obtaining Group Dimensions. In this step, the parties privately compute the number of matching bins in the other party's set. This number is required to later expand each table (the "tablized" set S), such that it can be one-to-one aligned (joined) with the other table. The algorithm starts by putting bins of both parties' tables into a single, joint table. In addition, each row is assigned an attribute tid of 1 or 2, respectively, to indicate their table of origin. Since we join on the tag attribute, the algorithm first sorts the rows by their tag attribute and then their tid attribute. After sorting, Krastnikov et al.'s algorithm does a forward linear pass on the combined table to count the number of bins with the same tid and tag, and another backward linear pass on the table to populate the results. This is a step that we need to modify in our algorithm to enhance the performance, as discussed later in this section. The count of rows with the same tag in Alice's table is stored in the attribute α_1 and the count of elements with this tag in Bob's table is stored in α_2. We write α_1^i and α_2^i for the counts of tag $i \in [1, \ldots, \ell]$.

Oblivious Distribution. After obtaining group dimensions, the rows are separated into two tables again, based on their tid attribute. The challenge is now to position (or distribute) each row from the input table, such that it one-to-one aligns with a row to be joined by the other table in its output table. Krastnikov et al.'s algorithm distributes a row (bin) B_i to position calculated based on B_{i-1}'s last position, the item count, and α_1^i, α_2^i.

Krastnikov et al. use an oblivious distribution algorithm. Specifically, the rows are distributed to the target positions by hopping intervals of powers of 2. That is, in each iteration $j \in [1, \ldots, \gamma]$, the rows are moved forward by $2^{\gamma-j}$ positions, where $\gamma = \lceil \log \beta \rceil$ and β is the total number of matching pairs, i.e., the size of the joint table, previously computed by the algorithm. After γ iterations, the bins are at their target positions.

Oblivious Expansion. After distributing the rows, the empty rows in the resulting table need to be filled with values copied from their adjacent rows. Krastnikov et al. also use a linear pass for this step, which we need to modify as described in the next section.

4.2.2 Modification: Round Complexity of Traversal. There are three traversals in Krastnikov et al.'s join protocol. For a table of size n, the most straightforward implementation – a linear scan – has $O(n)$ rounds in a multi-party computation. This is prohibitively slow even for small dataset sizes. We can convert these linear scans into parallel scans if we can formulate the function computed on each row as an associative function [9]. Then, by recursively separating the array into two halves, and applying the associative function over the two halves, we obtain the traversal result in $O(\log n)$ rounds, and each round executing $O(n)$ functions. Since the latency of multi-party computation is usually dominated by the communication and in particular the number of communication rounds, this reduction in the number of communication rounds greatly improves the overall efficiency.

We describe the forward pass in the step 'Obtaining Group Dimensions' as an example and the other linear passes are transformed correspondingly. Let $\alpha_1[j]$ denote the value of the attribute α_1 in row j, and similarly for other attributes α_2, tag, and tid. Our example linear scan then initializes the α_1 and α_2 attributes for tag 1 as follows.

tag	tid	α_1	α_2
1	1	1	0
1	2	0	1

In subsequent rows j, the example linear scan uses the following truth table to set the α_1 and α_2 attributes.

We modify the linear scan to a parallel scan with an associative function [9]. The parallel scan starts with blocks with one row each and then recursively merges adjacent blocks into larger blocks until ending up with one block. For blocks of size 1 the associative function uses the following truth table.

tid	α_1	α_2
1	1	0
2	0	1

Table 2: Truth table for linear scan

tag[j] = tag[j-1]	tid[j] = tid[j-1]	tid[j] = 1	$\alpha_1[j]$	$\alpha_2[j]$
true	true	true	$\alpha_1[j-1]+1$	$\alpha_2[j-1]$
true	true	false	$\alpha_1[j-1]$	$\alpha_2[j-1]+1$
true	false	any	$\alpha_1[j-1]$	1
false	any	true	1	0
false	any	false	0	1

In subsequent steps two adjacent blocks are merged into one. We consider the two blocks to have γ and β rows, respectively. The first block (block with smaller row numbers) has γ rows, and we denote its j^{th} row as $P1[j]$; the second block has β rows, and we denote its j^{th} row as $P2[j]$. We consider the j^{th} row of the resulting block to be $Q[j]$. We first copy the rows of the first block $P1$ into the resulting block Q: $Q[j] = P1[j]$ for $0 < j \leq \gamma$. Then, we copy the rows of the second block $P2$: $Q[j] = P2[j-\gamma]$ for $\gamma < j \leq \gamma + \beta$. Afterwards, we compute an associative function over the rows $j > \gamma$ of the resulting block Q using the following truth table:

Table 3: Truth table for parallel scan

tag[j] = tag[γ]	tid[j] = tid[γ]	tid[j] = 1	$\alpha_1[j]$	$\alpha_2[j]$
true	true	true	$\alpha_1[\gamma]+\alpha_1[j]$	$\alpha_2[j]$
true	true	false	$\alpha_1[\gamma]$	$\alpha_2[\gamma]+\alpha_2[j]$
true	false	any	$\alpha_1[\gamma]+\alpha_1[j]$	$\alpha_2[j]$
false	any	true	$\alpha_1[j]$	$\alpha_2[j]$
false	any	false	$\alpha_1[j]$	$\alpha_2[j]$

Analysis. We show that the functions using Table 2 in a linear scan and Table 3 in a parallel scan compute the same output.

We call the function to compute α_1, α_2 of the updated j^{th} row r'_j from the previous two rows r_{j-1}, r_j in the linear pass $f_1(r_{j-1}, r_j)$, and the function in the parallel scan to compute α_1, α_2 of the updated j^{th} row r'_j from the previous two rows r_γ, r_j $f_2(r_\gamma, r_j)$.

We first prove that, if we substitute f_1 with f_2. That is, if we use f_2 in the linear pass, we end up with the same result. For f_1, since the values of α_1 and α_2 of r_j are not used, these values can be arbitrary, and do not affect the resulting values of r'_j. Thus, we define the values the same way as we initialize a_1, a_2 for f_2 in the table:

tid	α_1	α_2
1	1	0
2	0	1

Since we use f_2 in place of f_1, the value of r_j when we calculate $f_2(r_{j-1}, r_j)$ is the value that we initialize r_j with, so we can substitute these by the constant values we use for initialization. Then, if we look at the truth table of f_2, with values of $\alpha_1[j]$ and $\alpha_2[j]$ substituted with that from the initialization table, we see that it is exactly the same as that of f_1. So, we have proven that these two functions are equivalent, if used in the linear pass.

We now prove that f_2 is associative, i.e., it can be used in parallel scan.

We consider two orders of applying f_2 on 3 rows of records: r_1, r_2, r_3, where $r_1 < r_2 < r_3$ in the order that the records were sorted by before this step, that is, increasing order of tag, then increasing order of tid. An extended truth table for three rows is shown in Table 4. We compare the result of $f_2(r_1, f_2(r_2, r_3))$ with $f_2(f_2(r_1, r_2), r_3)$. In order to compose the functions, we need the result of applying f_2 of two rows to be a row, then we extend the result of f_2 from just (α_1, α_2) to $(tag, tid, \alpha_1, \alpha_2)$, with tag, tid equal to the second argument to f_2. We call the tag and tid of r_1, r_2, r_3 to be $tag[1], tag[1], tag[3]$ and $tid[1], tid[2], tid[3]$, then we can write a table of results based on these values, which is the same with both application orders. So, we have proven that the truth Table 3 can be used in a parallel scan.

4.3 Record-to-Record Comparison

After sets of smoothed bins have been joined, our PRL protocol simply compares each pair of records in the joint bins. Any comparison function eq can be used. However, those functions that are efficiently implementable in multi-party computation lead to a higher efficiency of the entire protocol. We use the same assumptions of the related work by Khurram and Kerschbaum [25] in order to enable a fair comparison of the protocols. In our efficiency evaluation, we implemented exact equality as a baseline for performance measurements. In our accuracy evaluation, we assume the existence of the perfect comparison function that returns the ground truth. Furthermore, it separates the influence of the blocking algorithm (LPH or LSH) from the influence of the matching algorithm in accuracy evaluation. Any improvement of the matching algorithm can be applied to either protocol, but they necessarily differ in the blocking algorithm which is what we evaluate. We summarize the pseudo-code of this step in Algorithm 3. Let S_A and S_B be smoothed bins from Alice or Bob, respectively, which have the same tag, i.e., they have been joined in the previous step. We then compare each record in bin S_A to each other record in bin S_B.

Algorithm 3 Record to Record Compare(S_A, S_B)

```
for R_A ∈ S_A do
    for R_B ∈ S_B do
        if eq(R_A, R_B) = true then
            Output R_A to Alice
            Output R_B to Bob
        end if
    end for
end for
```

5 EVALUATION

We built a C++ prototype of the protocol for efficiency measurement, and a Python plain text computation version of the algorithm for

Table 4: Extended truth for parallel scan over three rows.

tag[1] = tag[2]	tag[2] = tag[3]	tid[1]	tid[2]	tid[3]	α_1	α_2
true	true	1	1	any	$\alpha_1[1] + \alpha_1[2] + \alpha_1[3]$	$\alpha_2[3]$
true	true	1	2	2	$\alpha_1[1] + \alpha_1[2]$	$\alpha_2[2] + \alpha_2[3]$
true	true	2	2	2	$\alpha_1[1]$	$\alpha_2[1] + \alpha_2[2] + \alpha_2[3]$
false	true	any	1	any	$\alpha_1[2] + \alpha_1[3]$	$\alpha_2[3]$
false	true	any	2	2	$\alpha_1[3]$	$\alpha_2[2] + \alpha_2[3]$
any	false	any	any	any	$\alpha_1[3]$	$\alpha_2[3]$

accuracy measurement. The C++ prototype was based on the ABY framework [7, 10].

5.1 Security Proof

We prove that our protocol is secure in the semi-honest (or passive or honest-but-curious) security model. In this model, an adversary passively observes the computation and tries to infer additional information, but follows the protocol specification. Such an attack is almost impossible to detect, since the output and steps of the protocol are unaffected. To the contrary, many active attacks can be detected by inspection of log files which can be deployed alongside semi-honest security. The semi-honest model is practically deployed [21, 44]. Any protocol secure in the semi-honest model can be transferred into one secure in the malicious model using Goldreich's compiler [14]. The malicious model provides protection against active attacks during the protocol. However, it is commonly too inefficient to deploy and does not provide protection against attacks before the protocol, such as input substitution which can exfiltrate the entire dataset. Indicators of input substitution can be detected from log files.

We say a protocol is secure in the semi-honest model, if there exist two simulators $\mathsf{Sim}_{A/B}$ for the views $\mathsf{View}_{A/B}$, respectively.

$$\begin{aligned} \mathsf{Sim}_A(T_A, n_B, O, c) &= \mathsf{View}_A(T_A, T_B) \\ \mathsf{Sim}_B(T_B, n_A, O, c) &= \mathsf{View}_B(T_A, T_B) \end{aligned}$$

Theorem 5.1. *There exist two simulators* $\mathsf{Sim}_A(T_A, n_B, O, c)$ *and* $\mathsf{Sim}_B(T_B, n_A, O, c)$ *whose output is computationally indistinguishable from the view of the respective party during the execution of our protocol.*

Proof. Our proof is by construction. Wlog. we construct the simulator for Alice's view. The simulator for Bob's view is analogous, since our protocol is symmetric.

We construct the simulator using the three steps of our protocol: preprocessing, bin join, and record-to-record comparison.

Preprocessing. This step is mostly performed locally on each party's dataset. Only the output of the preprocessing after frequency smoothing is shared as an input to a secure two-party computation. Hence, we only need to prove that this input is simulatable given the information to the simulator. According to Grubbs et al.'s analysis of their frequency smoothing [16], Alice receives $2d\ell$ bins with $\lceil n_B/\ell \rceil$ elements in each bin. The content of each element is secret-shared in the ABY framework and simulatable by a random number.

Secure Bin Join. Goldreich's composition theorem [14] in the semi-honest model allows us to prove security of the composed protocol by proving security of each step. Since we predominantly use greater-than or equality comparisons, our (entire) protocol is implemented over Boolean shares in ABY, i.e., the protocol by Goldreich et al. (GMW) [15]. Krastnikov et al. [28] prove that their algorithm implements a circuit and hence translation to GMW is straightforward. Our modifications change the round complexity of the computation, but are still circuits as can be easily verified from their description. Furthermore, a specification of the protocol in ABY ensures that the compiled protocol is secure and we do not release intermediate values. Hence, our simulator executes an arbitrary oblivious join over two smoothed LSH hash tables with $2d\ell$ bins each and a number of elements corresponding to the input tables sizes that produces a joint table of size $c/(\lceil n_B/\ell \rceil \cdot \lceil n_A/\ell \rceil)$.

Record-to-Record Comparison. Each comparison is implemented as a secure computation in GMW. We can apply the composition theorem again and compose c individual comparisons to the full record-to-record comparison. Note that we randomly perturb the order of records in the bins and the order of joint bins is random. The simulator hence performs c arbitrary comparisons and randomly releases the elements in the output O of the protocol given to the simulator as input.

This completes the construction of Alice's simulator and Bob's is analogous, since our protocol is symmetric. Both simulators together complete the proof. □

5.2 Efficiency Measurement

For different sizes of input, and different numbers of bins, we need different numbers of comparisons, which leads to different amounts of computation time required. We measure the relationship between the number of comparisons and the computation time needed. The number of comparisons is the number of comparisons performed in Section 4.3, and the computation time is measured in CPU time of both parties combined. The experiments were performed on a Ubuntu virtual machine on VirtualBox with 6 virtual cores (using 6 of 8 cores of 2 Intel i7-9750H @ 2.60GHz CPU), and 8 GB memory.

For both parties, we set our input numbers to be in the range of 0 to $2^{16} - 1$ as outputs of their LSHs. Then, for a given input size n, we randomly generate n numbers with uniform probability in the range. We put them into bins based on their values, where the bins are uniform intervals between 0 and $2^{16} - 1$. The bins are prepared as in Section 4.1. Then, the bins are privately joined and their contents are securely compared as in Section 5.

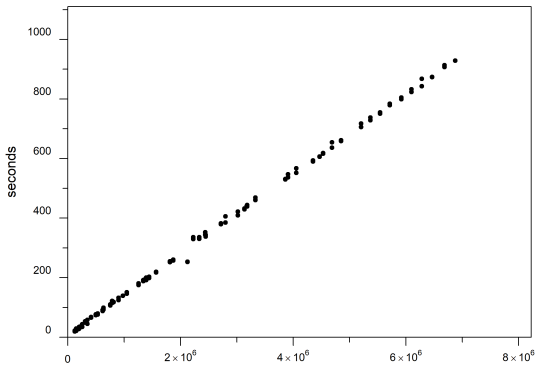


Figure 5: Efficiency measurement, CPU time (in seconds) over number of comparisons

The result using fixed parameters for the LSH function is depicted in Figure 5 and shows the expected linear correlation between number of comparisons and CPU time. From the results, we derive a correlation equation of $t = 0.000135c + 7.84$. Using this equation, we can estimate the amount of computation time t (in seconds) required given the number c of comparison. For example, in one hour of computation time, we can approximately perform 2.65×10^7 comparisons.

5.3 Accuracy Measurement

For comparison with the related work [25], we use the same two datasets for our accuracy measurement: BNB/TPL, and NCVR. We measure the accuracy of our algorithm, with respect to various parameters of the locality sensitive hash function. We estimate efficiency by measuring the number of comparisons needed and using our correlation formula to approximate the computing time.

As done in related work [17, 25], we assume that our match function that compares two elements is 100% accurate, i.e., returns the ground truth. Hence, any inaccuracy in our experiments must stem from the LSH and the corresponding efficiency gains of PRL protocol. Since multi-party computation allows implementing any function, one can always implement a match function that provides them with the best result. Note that these functions are efficient, since they take a small input of a pair of elements. Thus, for each pair of records that is similar in the ground truth, if one of their duplicates is hashed into the same bin, we observe a true positive; if none were hashed into the same bin, we observe a false negative; the accuracy is measured by $\frac{true\ positives}{true\ positives + false\ negatives}$.

We vary the following parameters of the LSH. One parameter is the length of the vector of min-hash values, t. This affects how similar two strings have to be, for them to be in the same bin. A second parameter is the number of duplicates of each record, d. The more duplicates we have for a record, the more likely a matching pair ends up in the same bin, but also the more comparisons are needed. A third parameter is the number of bins per duplicate, ℓ. The more bins we have per duplicate, the less likely a matching pair ends up in the same bin, but less comparisons are needed. Our experiments explore the performance-efficiency trade-off of these parameters t, d and ℓ.

5.3.1 BNB/TPL. The British National Bibliography (BNB) [3] contains records of the publishing activity of the United Kingdom and the Republic of Ireland since 1950. We only included the records with ISBN numbers. The Toronto Public Library (TPL) Open Data [8] contains a catalogue of books. We only included those with ISBN numbers.

The BNB/TPL dataset contains records of information about books. Each record includes the following 4 fields: ISBN number, book name, author's first name, and author's last name. We set the ground truth to be the records with exact match on the field 'ISBN number'. Then, we perform PRL based on the remaining fields, that is book name, author's first name, and author's last name. From these two datasets, we selected 100000 records respectively, with the least value of ISBN. Thus, we obtained two datasets with 100000 records each, and 19.267 matching records across the two datasets.

The dataset was prepared in a similar manner to related work [25], with the difference that we have selected the 100000 records in both datasets with the least ISBN number to compare, which is about 1/20 the size of their dataset.

We first varied the length of the vector of min-hash values, t. We fixed $d = 32$ and $\ell = 1024$, while varying the value of t, selecting from 1, 2, 3, 4. We obtained the results depicted in Figure 6a, and concluded that $t = 2$ offers a good trade-off between efficiency and accuracy.

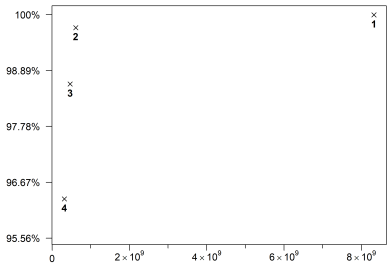
Then, we varied the number of duplicates of each record, d. We fixed $t = 2$ and $\ell = 1024$, while varying the value of d, selecting from 4, 8, 16, 32, 64. We obtained the results in Figure 6b, and concluded that $d = 32$ offers a good trade-off between efficiency and accuracy.

Finally, we varied the number of bins per duplicate, ℓ. We fixed $t = 2$ and $d = 32$, while varying the value of ℓ, selecting from 512, 1024, 2048, 4096, 8192. We obtained Figure 6c, and concluded that $\ell = 1024$ offers a good trade-off between efficiency and accuracy.

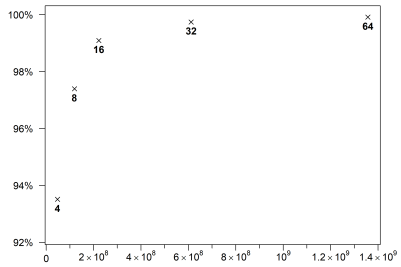
5.3.2 NCVR. The state of North Carolina keeps track of active and inactive voters since 2005, and provides a point-in-time snapshot of information of those voters in the North Carolina Voter Register (NCVR) [35]

We chose the snapshots taken at the beginning of the years 2014 and 2017. The records contain fields including ncid, first name, middle name, last name, age, and birth place. Of these 6 fields, we used the first field (ncid) to determine the ground truth of a matching record, and the remaining 5 for our algorithm to find a match. From the two datasets (ncvr at year 2014 and ncvr at year 2017), we selected 100000 records respectively, with the least value of ncid. Thus, we obtained two datasets with 100000 records each, and 84.391 matching records across the two datasets.

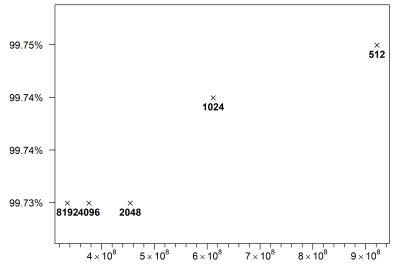
The dataset was prepared in a similar manner to the work by Khurram and Kerschbaum [25], with the difference that we have selected the 100000 records in both datasets with the least ncid number to compare, which is about 1/100 the size of their dataset. In addition, we have selected the data from the beginning of 2014 and 2017, while they selected from the data from November 2014 and 2017.



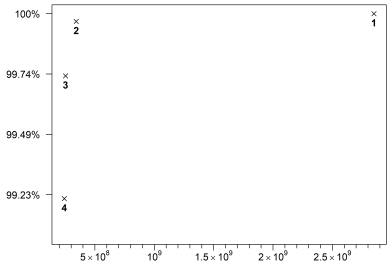
(a) BNB_TPL Result for varying vector length, accuracy vs. number of comparisons



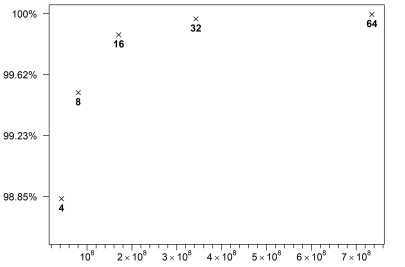
(b) BNB_TPL Result for varying number of duplicates, accuracy vs. number of comparisons



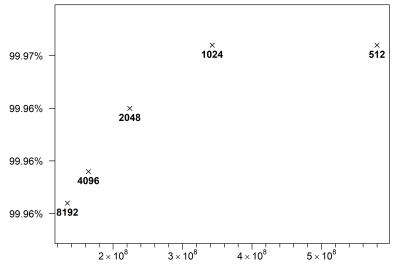
(c) BNB_TPL Result for varying number of bins per duplicate, accuracy vs. number of comparisons



(d) NCVR Result for varying vector length, accuracy vs. number of comparisons

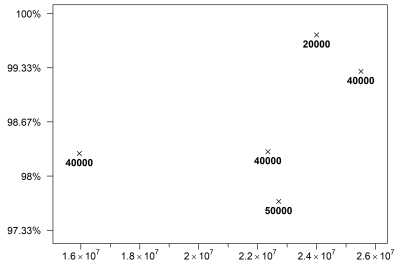


(e) NCVR Result for varying number of duplicates, accuracy vs. number of comparisons

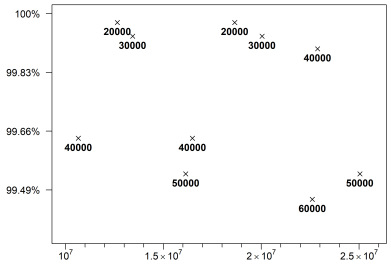


(f) NCVR Result for varying number of bins per duplicate, accuracy vs. number of comparisons

Figure 6: Accuracy results for BNB_TPL (left) and NCVR (right)



(a) BNB_TPL result within one hour for different sized inputs, accuracy vs. number of comparisons



(b) NCVR results within one hour for different sized inputs, accuracy vs. number of comparisons

Figure 7: dataset sizes over number of comparisons within an hour for BNB_TPL (left) and NCVR (right)

We first varied the length of the vector of min-hash values, t. We fixed $d = 32$ and $\ell = 1024$, while varying the value of t, selecting from $1, 2, 3, 4$. We obtained the results depicted in Figure 6d, and concluded that $t = 2$ offers a good trade-off between efficiency and accuracy.

Then, we varied the number of duplicates of each record, d. We fixed $t = 2$ and $\ell = 1024$, while varying the value of d, selecting from $4, 8, 16, 32, 64$. We obtained the results depicted in Figure 6e, and concluded that $d = 32$ offers a good trade-off between efficiency and accuracy.

Finally, we varied the number of bins per duplicate, ℓ. We fixed $t = 2$ and $d = 32$, while varying the value of ℓ, selecting from $512, 1024, 2048, 4096, 8192$. We obtained the results depicted in Figure 6f, and concluded that $\ell = 1024$ offers a good trade-off between efficiency and accuracy.

5.4 Comparison to Related Work

We compare efficiency to both, He et al.'s and Khurram and Kerschbaum's work [17, 25] and accuracy to Khurram et al.'s work. We do not compare accuracy to He et al., since we can always implement their algorithm in our protocol and it is possible to

implement our algorithm in their protocol, i.e., the record linkage algorithms are interchangeable between the two protocols. However, we achieve better security without additional differential private leakage, and we do so at a comparable cost. The code for neither of the two related works is available, such that we cannot reliably replicate their results. Therefore we compare based on the results reported in their papers. The parameters of the experiments reported by He et al. and Khurram and Kerschbaum can be summarized as follows. He et al. [17] uses two data sets: the Taxi data set [34] and the ABT-Buy data set [27]. Khurram and Kerschbaum use four data sets: the Taxi data set, the OpenStreet Map/Yelp, the British National Bibliography [3]/Toronto Public Library [8] and the North Carolina Voter Register [35]. Accuracy on the Taxi data set is trivially 100% for both Khurram and Kerschbaum and He et al. (and hence us). Therefore, we do not use that data set for comparison. The OpenStreetMap/Yelp data set is not publically available, so we compare using the other two data sets. However, Khurram and Kerschbaum post-processed the data sets and the resulting data sets are not available. We summarize our known changes to data set post-processing in Section 5.3 and the comparison in Section 5.4.1.

For efficiency, we compare the number of element comparisons and the wall-clock time. The number of comparisons is due to the blocking algorithm and padding. The wall-clock time is further influenced by the cryptographic protocol and the hardware. He et al. use two threads on a 3.1 GHz Intel Core i7 CPU with 16 GB RAM. Khurram and Kerschbaum use two 2.2 GHz Intel Xeon E7 CPU with 108 GB of RAM. We use two 2.6 GHz Intel Core i7 CPUs with 8 GB memory. All protocol software runs single-threaded (where other cores are available for system software). In comparison, He et al. use a faster CPU and Khurram and Kerschbaum a slower one. He et al. do not implement the cryptographic protocol and hence their memory resources may not be representative. We have much lower memory requirements than Khurram and Kerschbaum, since we do not need to fit the entire circuit of the protocol into memory. We summarize the comparison of wall-clock time in Section 5.4.2. Khurram and Kerschbaum did not document the number of comparisons or how to calculate them and therefore in Section 5.4.3 we compare the number of element comparisons only to He et al.

5.4.1 Accuracy. We compare our accuracy results from Section 5.3 for the common parameter set of $t = 2$, $\ell = 1024$, and $d = 32$ to Khurram and Kerschbaum [25]. Khurram and Kerschbaum use different parameters for their window size and number of iterations. We used both the default parameters of a $\log(n)$ window size and $i = 1$ iterations and the best accuracy reported. Table 5 shows that our protocol's accuracy exceeds that of Khurram and Kerschbaum's protocol by up to 10%.

Table 5: Accuracy on the same data sets

	Ours	[25] with $\log(n)$, $i = 1$	[25] best
BNB/TPL	99.74%	85.4%	89.7%
NCVR	99.97%	97.9%	98.6%

5.4.2 Wall-Clock Time. To compare wall-clock time to related work [17, 25], we attempt to estimate how many records can be compared within one hour (approximately 2.65×10^7 comparisons). We varied the number of records, as well as d and ℓ, while fixing $t = 2$. The results are depicted in Figures 7a and 7b. We found that with both datasets having 40000 records, if we set $d = 16$ and $\ell = 2048$, we can achieve 99.3% and 99.9% accuracy within one hour.

He et al. [17] report 80 hours for two datasets with 5000 records each. Khurram and Kerschbaum [25] report 1 hour for two datasets with 4000 records each. These numbers are influenced by the cryptographic protocols and different hardware. Table 6 shows the summary.

Table 6: Wall-clock time throughput

	Ours	[17]	[25]
Records per hour	40000	560[1]	4000

5.4.3 Number of Comparisons. We compare the number of element comparisons, which is independent of the cryptographic protocol and hardware. Khurram and Kerschbaum [25] require comparisons proportional to the windows size (e.g. $\log(n)$) times data set size n. However, this number is scaled by a constant which is not documented as well as any total number of comparisons. Hence, we do not include Khurram and Kerschbaum in this comparison.

The exact number of element comparisons in He et al.'s and our protocol is parameter and data set-dependent. However, the expected number of parameters is data set-independent. We can compare the expected number of comparisons using the same LSH parameters.

Table 7: Number of dummy elements per bin

n	10^4	10^5	10^6	10^7	10^8
Ours	40	50	60	70	80
[17] $\epsilon = 0.1$	3834	4571	5308	6045	6781
[17] $\epsilon = 0.5$	766	914	1061	1209	1356
[17] $\epsilon = 1$	383	457	530	604	678
[17] $\epsilon = 2$	191	228	265	302	339

We have n data elements and ℓ bins repeated d times. Hence the expected load of a bin is $\frac{n}{\ell}$. He et al. add dummy elements to each bin depending on the differential privacy parameters ϵ and δ. They have

$$\eta_0 = -\frac{d \log((e^{\epsilon/d} + 1)(1 - (1 - \delta)^{1/d}))}{\epsilon}$$

expected dummy elements per bin. Due to the splitting of bins, we add an expected half bin of dummy elements. We have

$$\eta_1 = \frac{n}{2\ell}$$

expected dummy elements per bin. The total expected number of comparisons is in both protocols

$$d\ell \left(\frac{n}{\ell} + \eta_{0/1}\right)^2$$

[1] Assuming running time is proportional to n^2.

It suffices to compare the dummy elements η_0 and η_1 per bin. We set $d = 32$, $\delta = 1/n$, and $\ell = \frac{n}{6\log(n)}$[2]. We summarize our comparison in Table 7. With these parameters He et al. always requires a higher number of comparisons than us.

6 CONCLUSIONS

We present a new PRL protocol that uses LSH blocking, but is cryptographically secure. This improves over previous work that use LSH with differential privacy in security, since it leaks less information, and performance as demonstrated by our prototype. It also improves over equally secure previous work in performance and accuracy, since it replaces the use of an LPH with an LSH. Further research is needed to improve the performance of PRL.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) for grants RGPIN-05849, IRC-537591, and the Royal Bank of Canada. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

A FORMAL SECURITY COMPARISON

A.1 Cryptographic Security

Khurram and Kerschbaum were the first to present a PRL protocol secure in the cryptographic setting of multi-party computations [25]. They use the same definition adapted from Goldreich [14] in the semi-honest model. A protocol is secure if a party's view $\mathsf{View}_{A/B}$ can be (computationally indistinguishably) simulated from only input and output of that party. Formally, let O be the set of matching record pairs. Let $\mathsf{View}_{A/B}$ be the view of Alice and Bob, respectively, i.e., the messages received and the random coins chosen during the protocol execution. We say a protocol is secure in the semi-honest model, if there exists two simulators $\mathsf{Sim}_{A/B}$ for the views $\mathsf{View}_{A/B}$, respectively.

$$\begin{aligned} \mathsf{Sim}_A(T_A, n_B, O) &= \mathsf{View}_A(T_A, T_B) \\ \mathsf{Sim}_B(T_B, n_A, O) &= \mathsf{View}_B(T_A, T_B) \end{aligned}$$

If such a simulator exists, then the following statement holds for any polynomial-time adversary $\mathcal{A}$:

$$Pr[\mathcal{A}(\mathsf{Sim}_{A/B}) = 1] = Pr[\mathcal{A}(\mathsf{View}_{A/B}) = 1]$$

We use the same definition for our protocol, but add the leakage of the total number of comparisons $c \leq n_A \cdot n_B$. The total number of comparisons is necessarily part of any view, but in case of Khurram and Kerschbaum, it is a deterministic function of the input length. We analyze this leakage in Section A.3.

A.2 Computationally Differentially Private Security

He et al. already used LSH but also used a strictly weaker definition of security [17]. They use the concept of differential privacy. In differential privacy, we consider neighbouring inputs $T_{A/B}$ and $T'_{A/B}$ which differ in at most one record. The adversary that computational differential privacy considers is no longer restricted to strict indistinguishability (as in Appendix A.1) but bounded indistinguishability [32]. Loosely speaking, the view of a party may differ between neighbouring datasets, but is restricted by the differential privacy parameter. Formally, let ϵ, δ be the privacy parameters. We say a protocol is secure in the computational differential privacy model, if for all neighbouring datasets $T_{A/B}$ and $T'_{A/B}$ and all polynomial-time adversaries $\mathcal{A}$ it holds that

$$Pr[\mathcal{A}(\mathsf{View}_A(T_A, T_B) = 1] \leq e^{\epsilon} Pr[\mathcal{A}(\mathsf{View}_A(T_A, T'_B)) = 1] + \delta$$
$$Pr[\mathcal{A}(\mathsf{View}_B(T_A, T_B) = 1] \leq e^{\epsilon} Pr[\mathcal{A}(\mathsf{View}_B(T'_A, T_B)) = 1] + \delta$$

He et al. observed that this strict definition is incompatible with PRL and hence further weakened it to consider the output O of the protocol. $T_{A/B}$ and $T'_{A/B}$ are only neighbours if they produce the same output O. This allows the protocol to output O without violating the security definition. Note that this is a strictly weaker definition than cryptographic security. It leaks the same information, i.e., output and what can be inferred from one party's input and output, as the cryptographic definition, but it also additionally leaks bounded information about the datasets in the view. In the protocol of He et al. this is the padded length of the bins which result in the total number of comparisons. Since the bound is with respect to a single change in the datasets, multiple changes, so called properties, e.g., if an ethnic, religious or marginalized group is part of the input, are leaked with much higher probability and can be inferred with high accuracy. We remove the differentially private padding of the bins and instead smooth the bins to remove this leakage and attack vector.

A.3 Leakage

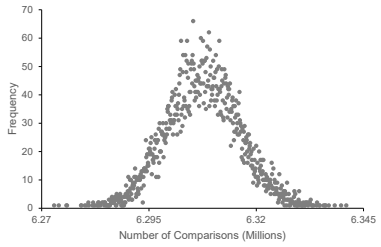


Figure 8: Distribution of number of comparisons for $n = 10000$, $l = 1024$, $d = 32$.

Our protocol leaks the number of comparisons, whereas the protocol by Khurram and Kerschbaum [25] does not. We empirically measure this leakage. We run the following experiment. We fix a database with $n = 10000$ entries for Alice and parameters $l = 1024$ and $d = 32$ as used in our accuracy evaluation. We then choose a random database for Bob ($n = 10000$) and count the number of comparisons. We repeat this experiment 10000 times and graph the results as an X-Y-plot in Figure 8. We see the expected normal distribution and can compute an empirical information entropy. We conclude that when comparing $n = 10000$ elements, we leak less than 9 bits of information about the entire data set.

[2]This approximates our parameters in the evaluation and our protocol then scales in $O(n\log(n))$.

REFERENCES

[1] Ali Al-Lawati, Dongwon Lee, and Patrick McDaniel. 2005. Blocking-aware private record linkage. In *International Workshop on Information Quality in Information Systems (IQIS)*.

[2] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. 2015. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks* 10, 3 (2015), 137–150.

[3] British National Bibliography. 2023. Available at https://www.bl.uk/collection-metadata/metadata-services.

[4] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*.

[5] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 1998. Min-wise independent permutations. In *ACM Symposium on Theory of Computing (STOC)*.

[6] Jianneng Cao, Fang-Yu Rao, Elisa Bertino, and Murat Kantarcioglu. 2015. A hybrid private record linkage scheme: Separating differentially private synopses from matching records. In *IEEE International Conference on Data Engineering (ICDE)*.

[7] ABY Source Code. 2023. Available at https://github.com/encryptogroup/ABY.

[8] Toronto Public Library Open Data. 2023. Available at https://opendata.tpl.ca/.

[9] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[10] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In *Network and Distributed Systems Security Symposium (NDSS)*.

[11] Halbert L Dunn. 1946. Record Linkage. *American Journal of Public Health and the Nations Health* 36, 12 (1946), 1412–1416.

[12] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*.

[13] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. 2018. Property inference attacks on fully connected neural networks using permutation invariant representations. In *ACM Conference on Computer and Communications Security (CCS)*.

[14] Oded Goldreich. 1998. Secure multi-party computation. Manuscript. Available at https://www.wisdom.weizmann.ac.il/~oded/pp.html.

[15] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game, or a completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC)*.

[16] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency smoothing for encrypted data stores. In *USENIX Security Symposium (USENIX Security)*.

[17] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. 2017. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *ACM Conference on Computer and Communications Security (CCS)*.

[18] Thomas Humphries, Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum. 2023. Investigating membership inference attacks under data dependencies. In *IEEE Computer Security Foundations Symposium (CSF)*.

[19] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita, and Elisa Bertino. 2010. Private record matching using differential privacy. In *International Conference on Extending Database Technology (EDBT)*.

[20] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. 1997. Locality-preserving hashing in multidimensional spaces. In *ACM Symposium on Theory of Computing (STOC)*.

[21] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[22] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. 2008. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy (S&P)*.

[23] Alexandros Karakasidis and Vassilios S Verykios. 2011. Secure blocking+ secure matching= secure record linkage. *Journal of Computing Science and Engineering* 5, 3 (2011), 223–235.

[24] Dimitrios Karapiperis, Vassilios S Verykios, Eleftheria Katsiri, and Alex Delis. 2016. A tutorial on blocking methods for privacy-preserving record linkage. In *International Workshop on Algorithmic Aspects of Cloud Computing (ALGO-CLOUD)*.

[25] Basit Khurram and Florian Kerschbaum. 2020. SFour: a protocol for cryptographically secure record linkage at scale. In *IEEE International Conference on Data Engineering (ICDE)*.

[26] Kunho Kim and C Lee Giles. 2016. Financial entity record linkage with random forests. In *International Workshop on Data Science for Macro-Modeling (DSMM)*.

[27] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment* 3, 1 (2010). https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

[28] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. 2020. Efficient oblivious database joins. *Proceedings of the VLDB Endowment* 13, 11 (2020), 2132–2145.

[29] Mehmet Kuzu, Murat Kantarcioglu, Ali Inan, Elisa Bertino, Elizabeth Durham, and Bradley Malin. 2013. Efficient privacy-aware record integration. In *International Conference on Extending Database Technology (EDBT)*.

[30] Glenda Lawrence, Isa Dinh, and Lee Taylor. 2008. The Centre for Health Record Linkage: a new resource for health services research and evaluation. *Health Information Management Journal* 37, 2 (2008), 60–62.

[31] Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, Vol. 10. 707–710.

[32] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. 2009. Computational differential privacy. In *International Cryptology Conference (CRYPTO)*.

[33] Christopher L Morgan, Craig J Currie, and John R Peters. 2000. Relationship between diabetes and mortality: a population study using record linkage. *Diabetes Care* 23, 8 (2000), 1103–1107.

[34] NYC Taxi and Limousine Commission. [n.d.]. TLC Trip Record Data. Available at https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[35] North Carolina Voter Registry. 2023. Available at https://www.ncsbe.gov/results-data/voter-registration-data#historical-data.

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

[37] Adi Shamir. 1980. On the power of commutativity in cryptography. In *International Colloquium on Automata, Languages and Programming (ICALP)*.

[38] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. 2022. Synthetic data–anonymisation groundhog day. In *USENIX Security Symposium (USENIX Security)*.

[39] Rebecca C Steorts, Samuel L Ventura, Mauricio Sadinle, and Stephen E Fienberg. 2014. A comparison of blocking methods for record linkage. In *International Conference on Privacy in Statistical Databases (PSD)*.

[40] Dinusha Vatsalan, Dimitrios Karapiperis, and Vassilios S Verykios. 2022. Privacy-preserving record linkage. *arXiv preprint arXiv:2212.05682* (2022).

[41] Anushka Vidanage, Thilina Ranbaduge, Peter Christen, and Rainer Schnell. 2022. A taxonomy of attacks on privacy-preserving record linkage. *Journal of Privacy and Confidentiality* 12, 1 (2022).

[42] Zikai Wen and Changyu Dong. 2014. Efficient protocols for private record linkage. In *ACM Symposium on Applied Computing (SAC)*.

[43] Mohamed Yakout, Mikhail J Atallah, and Ahmed Elmagarmid. 2009. Efficient private record linkage. In *IEEE International Conference on Data Engineering (ICDE)*.

[44] Moti Yung. 2015. From mental poker to core business: Why and how to deploy secure computation protocols?. In *ACM Conference on Computer and Communications Security (CCS)*.