Privacy-Preserving Temporal Record Linkage

Thilina Ranbaduge
Research School of Computer Science
Australian National University
Canberra, ACT 0200 Australia
thilina.ranbaduge@anu.edu.au

Peter Christen
Research School of Computer Science
Australian National University
Canberra, ACT 0200 Australia
peter.christen@anu.edu.au

***Abstract*—*Record linkage* (RL) is the process of identifying matching records from different databases that refer to the same entity. It is common that the attribute values of records that belong to the same entity do evolve over time, for example people can change their surname or address. Therefore, to identify the records that refer to the same entity over time, RL should make use of temporal information such as the time-stamp of when a record was created and/or update last. However, if RL needs to be conducted on information about people, due to privacy and confidentiality concerns organizations are often not willing or allowed to share sensitive data in their databases, such as personal medical records, or location and financial details, with other organizations. This paper is the first to propose a *privacy-preserving temporal record linkage* (PPTRL) protocol that can link records across different databases while ensuring the privacy of the sensitive data in these databases. We propose a novel protocol based on Bloom filter encoding which incorporates the temporal information available in records during the linkage process. Our approach uses homomorphic encryption to securely calculate the probabilities of entities changing attribute values in their records over a period of time. Based on these probabilities we generate a set of masking Bloom filters to adjust the similarities between record pairs. We provide a theoretical analysis of the complexity and privacy of our technique and conduct an empirical study on large real databases containing several millions of records. The experimental results show that our approach can achieve better linkage quality compared to non-temporal PPRL while providing privacy to individuals in the databases that are being linked.**

I. Introduction

In application domains such as banking, health, and national security, it has become an increasingly important aspect in decision making activities to integrate information from multiple sources [1], [2]. Integrated databases can help to identify similar records in different databases that correspond to the same real-world entity which can facilitate efficient and effective data analysis and mining not possible on an individual database. However, since organizations collect vast amount of data in their databases it is becoming increasingly challenging to integrate and combine data from different databases [3]. The process of identifying records that belong to the same real-world entity across different databases is known as *record linkage* (RL), *data matching* or *entity resolution* [2].

RL has been studied extensively over the past two decades [2]. Traditional RL techniques first compute the similarity between each pair of records from different databases. Next, the compared record pairs are grouped into clusters based on the calculated similarities with the aim that all records in the same cluster refer to the same entity while records in different clusters refer to different entities. However, these traditional techniques do not guarantee accurate linkage of data that can change over time [4], [5].

In the real world, RL is challenged because unique identifiers across all databases are not always available. Therefore, the use of personal identifiers (known as *quasi-identifiers* [6]), such as first and last name, address details, and so on, is commonly used in RL for matching pairs of records across different databases [2]. However, the use of personal identifiers raises privacy and confidentiality concerns when the databases to be linked belong to different organizations [7]. Often organizations are not willing or authorized to reveal or share any sensitive information about entities in their databases to any other party which makes the linkage process challenging.

Privacy-preserving record linkage (PPRL) (also known as *private record linkage* or *blind data linkage*) aims to develop linkage techniques that can link databases with sensitive information [7]. PPRL allows the linkage of databases without the need of any private of confidential information to be shared between the participating organizations involved in the linkage process. In PPRL the attribute values of records are usually encoded or encrypted before they are being compared ensuring that approximate similarities between records can still be calculated without the need of sharing the actual attribute values. PPRL is conducted in such a way that only limited information about the record pairs classified as matches is revealed to the participating organizations. The techniques used in PPRL must guarantee no participating party, nor any external party, can compromise the privacy of the entities in the databases that are linked [7].

In both traditional RL and PPRL the databases to be linked are considered as static (records are not changing over time) such that no attribute values of records would change over time and any changes of values over time of records of the same entity are considered as errors or variations. This assumption can potentially leads to incorrect record pair classifications because most linkage techniques consider record pairs that are highly similar as matches, and therefore records with similar attribute values (like a very similar address) that however refer to two different entities will be linked together [8].

However, entities may change or evolve their attribute values over time. For example, people often change their addresses or phone numbers, and some may even change their

TABLE I
AN EXAMPLE TEMPORAL DATABASE THAT SHOWS THE ATTRIBUTE VALUE CHANGES IN RECORDS THAT BELONG TO THE SAME ENTITY.

Record	Entity	First name	Last name	Street Address	Creation date
r_1	e_1	Anne	Miller	161 Main Road Sydney	2002-09-11
r_2	e_2	Anne	Smith	43 Town Place Sydney	2005-05-23
r_3	e_2	Anne	Miller	43 Town Place Sydney	2006-11-05
r_4	e_1	Anne	Smith	23 Town Place Sydney	2007-04-10
r_5	e_2	Anne	Miller	12 Main Road Sydney	2007-12-21
r_6	e_3	Anne	Miller	12 Main Road Sydney	2010-02-11

names after getting married or divorced, as shown in Table I. In both RL and PPRL, temporal information of records, such as the time when a record was created or last modified in a database, are not used in similarity calculations [9]. Incorporating temporal information into similarity calculations between record pairs can help to identify similar records that belong to the same entity over a period of time.

For example, given six records of three entities (records r_1 and r_4 of entity e_1, r_2, r_3, and r_5 of entity e_2, and r_6 of entity e_3) in Table I, a linkage without considering temporal information would potentially classify r_1, r_5, and r_6 as matches because they have high attribute value similarities, and it would potentially classify records r_2 and r_3 to refer to different entities because they have different last names.

Temporal record linkage (TRL) matches records in databases while considering temporal information in the form of attribute values that evolve over time [9]. In TRL the similarities between record pairs are calculated based on temporal information. Such temporal information enables the calculation of temporal similarities between records which can then be used in TRL to adjust the overall similarity of a record pair accordingly. For example, if 30% of entities change their addresses over a period of 5 years then address similarity over this time period should not be given a high weight in the overall similarity calculation of record pairs. In the example in Table I, the aim of TRL is to correctly link r_1 with r_4, and r_2, r_3, and r_5 using the temporal information provided in the creation date attribute. The challenge of how to make use of sensitive information in temporal linkage, while at the same time ensuring the privacy of such information, is a problem that not has been studied so far.

To this end we propose a novel PPRL technique that can be used to link databases using temporal information. The aim of our approach is to identify the records that belong to the same entity across databases owned by two or more organizations while preserving the privacy of the entities in those databases. We adjust the similarities between a pair of records based on their temporal distance since it is more likely to find similar entities that have similar attribute values over a long period of time. For example, it is more likely to have the same patient with different addresses attend the same hospital over the past 10 years instead of at the same time.

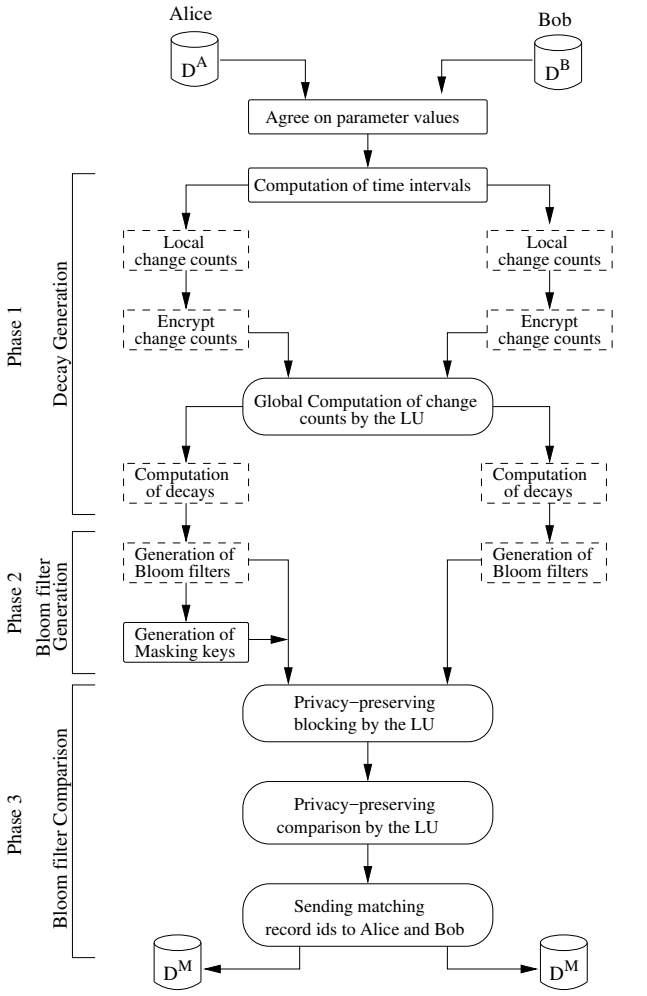

Fig. 1. Overview of our temporal PPRL protocol (for linking two databases) with three main phases. As detailed in Section IV-A, the database owners (DOs) Alice and Bob start the protocol by independently calculating the list of decay intervals and corresponding decay probabilities in phase 1. In phase 2, Alice and Bob then encode the records in their databases ($\mathbf{D}^A$ and $\mathbf{D}^B$, respectively) into Bloom filters (BF). As we explain in Section IV, only one DO needs to generate the BF masking keys for comparison. Masking keys and BFs are sent to a linkage unit (LU) in phase 3 for comparison, and the final set of classified matched record identifiers ($\mathbf{D}^M$) are sent back to the DOs. Each step shown in rectangles with rounded edges is performed by the LU while all other steps are performed by the DOs. Rectangular boxes with dashed lines represent steps that are performed similarly by the DOs.

As shown in Fig.1, our temporal PPRL (PPTRL) protocol accepts two or more databases as input and outputs a list that contains the identifiers of record pairs that belong to the same entity. We propose a secure protocol for each participating organization to individually (1) calculate the probabilities for an attribute value of an entity to change over different time periods and (2) for different entities to share the same attribute value over time. Such probability values are known as *decay values* [9]. Using these decay values we assign a weight to each attribute where the sum of attribute weights are used to calculate the similarity between a pair of records. We use this adjusted similarity value to decide if a pair of records is a match or a non-match using a similarity threshold.

Contribution: We propose a novel protocol that can perform PPRL between different databases while incorporating temporal information. We use a homomorphic encryption [10] based technique to calculate decay values between different databases. We then proposed a Bloom filter (BF) encoding [11] and masking based technique to securely calculate the similarities between record pairs. We analyze the privacy of our protocol which shows no participating party can learn anything about the entities in the other databases. An empirical evaluation with large real databases shows our protocol is scalable to large databases which make this protocol applicable to Big Data applications. As we show in our experiments our proposed protocol can easily be extended to multiple databases which allows the protocol to be used in a multi-party context [6], [7]. To the best of our knowledge, no temporal linkage technique has been proposed for PPRL.

II. Related Work

The concepts of using temporal information for linking records was first introduced by Li et al. [9] who proposed a supervised temporal model that considers the probability of attribute value changes, known as *time decays*, over time. These probabilities are used as weights to adjust the similarities calculated between pairs of records based on their time difference. The experiments conducted on a bibliographic database showed that using temporal information of records can improve linkage quality.

Christen and Gayler [4] proposed an approach to adaptively train a temporal model using a stream of temporal records. The proposed technique calculates the disagreement decay similar to [9], but instead uses the agreement decay based on the frequency distribution of attribute values. This approach is thus similar to the frequency based weight adjustment applied in traditional RL [2]. Further, this approach continuously trains the temporal model using linkage results which leads to improved linkage quality.

Chiang et al. [12] proposed an algorithm to learn a decay value of an attribute value based on its re-occurrences within different time periods. This approach uses a recurrence function to calculate the probability of each attribute value to reoccur after a certain period of time. Compared to [9] this approach uses the entire temporal history (changes of values) of an entity to adjust the similarities of pairs of records. A recent approach by Li et al. [13] adapted [12] by introducing a model that learns the transition probabilities between attribute values based on the statistics of their occurrences in entity profiles. The calculated transition probabilities are then used in the process of linking a record to an entity profile. The experimental results showed that the use of transition probabilities of an attribute value in the similarity calculations between records could potentially increase the overall quality of linking records to generate a complete entity profile.

Recently, Hu et al. [5] proposed a linear regression model added into the decay value calculation that uses a supportive attribute to calculate the decay of an attribute. The idea behind this approach is that the probability for an attribute value to change over time can be affected by other attributes that it depends upon. For example, any changes to the last name of an entity might depend on the gender of the entity such that female entities are more likely to change their last name over time than male entities. Hence, using gender as the support attribute in the decay calculation for last name enables the linear regression model to learn gender specific decay models for last name. Experiments conducted on a voter database showed the use of support attributes for learning decay of a given attribute can improve the linkage quality compared to the original temporal linkage approach by Li et al. [9].

Another recently proposed approach by Christen et al. [14] utilizes the relationships between entities to determine the similarity of groups of entities in different households using a graph-based method. The approach follows an iterative process that first identifies high quality links between records thereby limiting the more error-prone identification of links between less similar records. The approach uses temporal information such as age differences between entities to identify similar households in different historical census databases.

To summarize, all these existing approaches to temporal linkage use temporal information for similarity calculations on pairs of records, and they assume all databases to be linked belong to the same database owner. Therefore the privacy of each individual represented by these records is not considered. However, when databases are to be linked across different organizations none of these temporal linkage techniques could be used because they do neither protect the privacy of individuals nor the sensitive attribute values of these individuals. On the other hand, existing PPRL techniques do not make use of temporal information available in records when calculating similarities which makes them unsuitable for linking temporal databases [7]. To the best of our knowledge, ours is the first work to address the problem of performing temporal RL on different databases while preserving the privacy of individuals in these databases.

III. Problem Statement

Let $\mathbf{D}^i$ represents a database that belongs to database owner DO_i. Each record $r \in \mathbf{D}^i$ represents an entity e from a set of entities $\mathbf{E}$. Each r consists of a list of attribute values $\mathbf{A} = [A_1, A_2, \cdots, A_M]$ and a time-stamp $r.t$. We use $r.A_m$ to denote the value of attribute A_m in r with $1 \leq m \leq M$ where $M = |\mathbf{A}|$ is the number of attributes. We use $r.e$ to denote that record r refers to entity e.

An entity e can have multiple records where each record r_i contains a time-stamp $r_i.t$ to denote its time of creation or its last modification in the database. For a given entity e, its records r_i can contain the same or changed values in the attributes $A \in \mathbf{A}$. Let us assume two records r_i and r_j of entity e (i.e., $r_i.e = r_j.e$). We can say an entity e has changed the value of attribute A between time-stamps $r_i.t$ and $r_j.t$ if $r_i.A \neq r_j.A$, and $r_i.t < r_j.t$. Our aim is to identify which records across different databases $\mathbf{D}$ held by different DOs refer to the same entity $e \in \mathbf{E}$. We formally define the problem of temporal privacy-preserving record linkage as follows.

Definition 1. *Temporal privacy-preserving record linkage*: Assume d database owners DO_i, $1 \leq i \leq d$, with their respective databases $\mathbf{D}^i$. Each record $r^i_j \in \mathbf{D}^i$, $1 \leq j \leq |\mathbf{D}^i|$, in the form of $(r^i_j.A_1, r^i_j.A_2, \cdots, r^i_j.A_m, r^i_j.t)$ where $r^i_j.t$ is the time-stamp associated with record r^i_j, and $r^i_j.A_m$, $m \in [1, M]$, is the value of attribute A_m at the time t represents an entity e in a set of entities $\mathbf{E}$, such that every record r^i_j must belong to exactly one entity $e \in \mathbf{E}$. The linkage across the databases $\mathbf{D}^i$, $i \in [1, d]$, aims to determine which of their records $r^i \in \mathbf{D}^i$ belong to the same entity e even if $\exists A_m \in \mathbf{A} : r^i.A_m \neq r^j.A_m$, where $r^i \in \mathbf{D}^i$, $r^j \in \mathbf{D}^j$, $r^i.e = r^j.e$, $i \neq j$, and $i, j \in [1, d]$. The aim of temporal privacy-preserving record linkage is that at the end of the linkage process the DOs learn only which of their records belong to the same entity $e \in \mathbf{E}$ without revealing the attribute values of the records $r^i \in \mathbf{D}^i$ to any other DO or any party external to the DOs.

We now describe the calculation of decay values which we will use to adjust the similarities between a pair of records. We use the two decay values *disagreement decay* and *agreement decay*, as proposed by Li et al. [9] in our approach. These two decays describe the characteristics of attributes values of entities across a time period. Disagreement decay defines the probability that an entity changes its value for a given attribute over a certain period of time, while agreement decay specifies the likelihood that multiple entities share the same value for an attribute over a period of time. We define the *time period* (Δt), which also can be defined as a *time interval*, as the difference between a start (t_s) and an end time (t_e), denoted as [t_s, t_e], which can be measured in days, months, or years. We measure the *time distance* (t_d) between two records r_i and r_j as the difference between their time-stamps, such that $t_d = |r_i.t - r_j.t|$. From here onward we use the terms time interval and time period interchangeably.

Both disagreement and agreement decays can be learned using training data or specified by domain experts [9]. In this paper we use labeled records available in a training dataset to calculate decays assuming DOs know the attribute value changes that occur in records in their databases that belongs to the same entity. With such labeled data we can identify the number of occurrences where an entity changes its attribute value(s) over a certain period of time. However, the need of training data is a limitation of our approach.

To this end we formally define the disagreement and agreement decays as follows [9]. Consider an attribute A and a time period Δt. Assume each entity $e \in E$ has a list of records $T = [r_1, r_2, \cdots, r_n]$ such that $r_i.t < r_{i+1}t$, $1 \leq i \leq n-1$.

Definition 2. Disagreement decay ($d^{\neq}$) [9] is the probability that an entity e changes its value for attribute A in Δt time period. We calculate $d^{\neq}$ as follows.

$$d^{\neq}(A, \Delta t) = \frac{|\{e : \forall_{e \in E} \forall_{r_i, r_{i+1} \in T} (r_{i+1}.t - r_i.t) \leq \Delta t \wedge r_{i+1}.A \neq r_i.A\}|}{|E_c| + |E_{nc}|},$$

where E_c represent the set of entities that change their values in attribute A, $E_c = \{e : \forall_{e \in E} \forall_{r_i, r_{i+1} \in T} r_{i+1}.A \neq r_i.A \wedge (r_{i+1}.t - r_i.t) \leq \Delta t\}$, and E_{nc} represents the set of entities that do not change their values in attribute A, $E_{nc} = \{e : \forall_{e \in E} \forall_{r_i, r_{i+1} \in T} (r_{i+1}.t - r_i.t) \geq \Delta t \wedge r_{i+1}.A = r_i.A\}$.

Definition 3. Agreement decay ($d^{=}$) [9] for Δt is the probability that two entities e_i and e_j, such that $e_i, e_j \in E$ and $i \neq j$, share the same value for attribute A. This probability depends upon how frequently a certain value a occurs in attribute A. We calculate agreement decay $d^{=}$ as follows.

$$d^{=}(A, \Delta t) = \frac{|\{(e_i, e_j) : \forall_{e_i, e_j \in E} \ |r_i.A - r_j.t| \leq \Delta t \wedge r_i.A = r_j.A\}|}{|E_a|},$$

where E_a represents the union of the set of entity pairs that share the same attribute value of A and the set of entity pairs that do not share the same value for attribute A, $E_a = \{(e_i, e_j) : \forall_{e_i, e_j \in E} \ |r_i.A - r_j.t| \leq \Delta t \wedge r_i.A = r_j.A\} \cup \{(e_i, e_j) : \forall_{e_i, e_j \in E} \ |r_i.A - r_j.t| \leq \Delta t \wedge r_i.A \neq r_j.A\}$ and $r_i \in T_i$ of e_i and $r_j \in T_j$ of e_j.

For example, the probability that two entities share a rare English surname like 'Mirren' depends upon the likelihood that a record contains such a value in the surname attribute.

The aim of our approach is to use these disagreement and agreement decay probabilities to adjust the similarities between a pair of records in a privacy-preserving manner without revealing any attribute values. We next describe our approach in more detail.

IV. Temporal PPRL

Without loss of generality, let us assume two database owners (DOs), Alice (DO_A) and Bob (DO_B), with databases $\mathbf{D}^A$ and $\mathbf{D}^B$, respectively. We assume a *linkage unit* (LU) is employed in the protocol to facilitate the linkage [7], [2]. The LU is a party that may or may not be external to the DOs [3]. In general, the LU does not have any data itself but it conducts the linkage of the data sent to it by DOs. We assume the DOs are honest and do not collude with any other party, and the LU follows a semi-honest adversary model [10].

Our temporal PPRL protocol is designed to allow DOs to compute the decay values globally based on the attribute value changes of entities in both databases. We assume each database is not de-duplicated such that each entity can be represented by one or more records in a database. As shown in Fig. 1, our protocol consists of three main phases:

1) *Decay generation*: As we describe in Section IV-A, the DOs first individually compute a list of time intervals, $\mathbf{T}_I$. These time intervals are used to calculate disagreement and agreement decay probabilities which are then used to adjust the similarity of pairs of records. Each DO computes the attribute value changes of the entities in its database independently. The LU computes the summation of these attribute value change counts which are then send to the DOs to compute the decay probabilities for each attribute in each interval $I \in \mathbf{T}_I$.
2) *Bloom filter generation*: Each DO encodes the records in its database using Bloom filter (BF) encoding [11]. As we describe in Section IV-B, to assist the LU in the

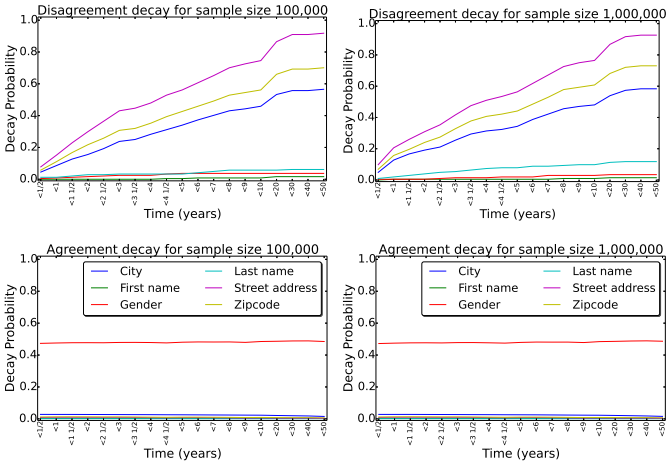

Fig. 2. Decay probabilities for different attributes based on samples of 100,000 and 1,000,000 records from the North Carolina voter registration database (described in Section VI). The top row shows disagreement decay probabilities while the bottom row shows agreement decay probabilities. As can be seen (and as expected [4]), disagreement decays increase monotonically with time while agreement decays do not change over time.

Algorithm 1: Decay generation

Input:
- $\mathbf{L_D}$: List of databases, $\mathbf{L_D} = [\mathbf{D}^i]$, $1 \leq i \leq d$
- $\mathbf{A}$: List of attributes, $\mathbf{A} = [A_1, A_2, \cdots, A_M]$, $1 \leq M \leq |\mathbf{A}|$
- n_s: Sample size
- n_I: Number of required temporal intervals
- n_k: Minimum entity count
- pk, sk: Public and private key for encryption and decryption

1: $\mathbf{T}_I = [\,]$ // Initialize an empty list for time intervals
2: **foreach** $i \in [1, 2, \cdots, n_I]$ **do:**
3: $I = genInterval(\mathbf{L_D}, n_s, n_k)$ // Compute the time interval
4: $\mathbf{T}_I.add(I)$ // Add the interval to list of intervals
5: **foreach** $\mathbf{D}^i \in \mathbf{L_D}$ **do:**
6: $\mathbf{D}^i_c = compChangeCounts(\mathbf{D}^i, n_s, \mathbf{T}_I, \mathbf{A})$ // Get change counts
7: $\overline{\mathbf{D}}^i_c = encrypt(\mathbf{D}^i_c, pk)$ // Encrypt the local change counts
8: DO^i sends $\overline{\mathbf{D}}^i_c$ to the LU
9: LU receives $\overline{\mathbf{D}}^i_c$ // The LU receives encrypted change counts
10: LU computes $\overline{\mathbf{D}}_c = \sum_{i \in [1,d]} \overline{\mathbf{D}}^i_c$ // Homomorphic addition of counts
11: LU sends $\overline{\mathbf{D}}_c$ to DOs // Sends encrypted sum to Alice and Bob
12: **foreach** $i \in [1, d]$ **do**:
13: DO$_i$ receives $\overline{\mathbf{D}}_c$
14: $\mathbf{D}_c = decrypt(\overline{\mathbf{D}}_c, sk)$ // DOs decrypt summed change counts
15: $\mathbf{d}^{\neq}, \mathbf{d}^{=} = computeDecay(\mathbf{D}_c, \mathbf{T}_I, \mathbf{A})$ // Compute decay values

third phase of our protocol one of the DOs generates a list of masking BFs used to adjust the similarities of pairs of BFs compared by the LU based on the temporal distance of the corresponding records. Each DO then sends its encoded database to the LU.

3) *Bloom filter comparison*: As we describe in Section IV-C, the LU first applies a blocking technique upon the received BF databases of each DO and then generates a set of blocks, $\mathbf{B}$. We use a Hamming distance based locality sensitive hashing (HLSH) technique as the blocking technique [15]. The BF pairs in each block $B \in \mathbf{B}$ are compared using a similarity based classification and the record identifiers of the classified matching BF pairs are sent back to the DOs.

Prior to these three phases, as prerequisites, the participating DOs agree upon the list of attributes $\mathbf{A}$ to be used, the parameters required for Bloom filter (BF) generation, including the BF length l, the q-gram length q (in characters), the number of hash functions k, a similarity function $sim()$ to compare the pairs of BFs, and the minimum similarity threshold s_t to decide if a pair of BFs is a match or not. As we describe in Section IV-A, all DOs also agree upon a public key pk and a secret key sk to be used in a homomorphic cryptosystem [16] in Phase 1 of our protocol. In the following three subsections we discuss each phase in more detail.

A. Decay Generation

In the first phase of our protocol each DO computes the disagreement and agreement decay probabilities that are needed in the record comparison phase to adjust similarities between record pairs. Algorithm 1 outlines the steps performed by the DOs and the LU.

First the DOs compute the list of time intervals that need to be considered when computing decay values (lines 2 to 4). The participating DOs need to agree on the number of intervals, n_I, that need to be considered. The DOs follows an iterative approach where in each iteration each DO computes a possible time interval that is added to the list of time intervals $\mathbf{T}_I$ using the function $genInterval()$ (line 3).

In the function $genInterval()$ each DO$_i$ randomly samples n_s records from its database. Each DO$_i$ then iterates through its sampled records to calculate a possible time period that can be used as an interval I_i. As explained in Section V, to improve privacy each DO$_i$ ensures there are records of at least $n_k > 1$ entities available with their pairwise time distances that are within I_i. This is required because if, for example, an interval I contains attribute value changes or a new record creation of an entity e in $\mathbf{D}^A$ of Alice, then Bob can learn in the decay computation step that Alice's database contains a single entity that has changed its attribute values within I. This could potentially lead to the identification of e in the record comparison phase. As shown in Fig. 1, all DOs follow these steps independently.

Once each DO$_i$ has computed its interval I_i in a given iteration, all DOs participate in a secure multi-party computation protocol to calculate the maximum time interval I out of all generated intervals I_is. This computation follows *Yao's Millionaires problem* [17] where we use a homomorphic encryption technique to identify the maximum time interval I from all computed I_is to ensure all DOs have records of at least n_k entities in the time interval I [18]. The interval I is returned from the function $genInterval()$ which is then added into the list of time intervals $\mathbf{T}_I$ in line 4. To avoid such computations, as a prerequisite, all DOs could instead agree on the list of time intervals to be considered in the decay calculations which can be used as input to the algorithm.

In lines 5 to 9, each DO$_i$ computes the list of change counts $\mathbf{D}^i_c$ independently for each time interval $I \in \mathbf{T}_I$. Following

Definitions 2 and 3, in line 6 each DO computes the number of times an entity does change and does not change its value for each attribute $A \in \mathbf{A}$, and the number of entity pairs that share the same value and different values for each attribute $A \in \mathbf{A}$ for each interval $I \in \mathbf{T}_I$. To improve the efficiency each DO first samples a set of records and computes the required change counts. As shown in Fig. 2, given a large enough sample size, adding further records to the sampling does not significantly affect the calculation of decay probabilities.

Each DO$_i$ then encrypts the calculated list of local change counts $\mathbf{D}_c^i$ into $\overline{\mathbf{D}}_c^i$ using the function $encrypt()$ and the public key pk (line 7). Each DO$_i$ then sends its $\overline{\mathbf{D}}_c^i$ to the LU (line 8). Once all $\overline{\mathbf{D}}_c^i$s are received, the LU uses homomorphic addition [16], [18] to add all encrypted $\overline{\mathbf{D}}_c^i$s into $\overline{\mathbf{D}}_c$ (line 10). The LU sends $\overline{\mathbf{D}}_c$ to all DOs.

Once $\overline{\mathbf{D}}_c$ is received by a DO, the DO decrypts $\overline{\mathbf{D}}_c$ into $\mathbf{D}_c$ using the function $decrypt()$ and using the secret key sk (line 14). The function $computeDecay()$ is used to generate the list of disagreement ($\mathbf{d}^{\neq}$) and agreement ($\mathbf{d}^{=}$) decay probabilities (as discussed in Section III) using $\mathbf{D}_c$ for each attribute $A \in \mathbf{A}$ for each interval $I \in \mathbf{T}_I$ (line 15). These decays probabilities are used next in the second phase of our protocol to generate a set of masking BFs.

B. Bloom Filter Generation

As shown in Fig. 1, the second phase of our protocol consists of two main steps. First each DO encodes its database into Bloom filters (BFs). A BF is a bit vector of length l bits initially set to 0. To encode a value into a BF a set of hash functions is used where based on the output of each hash function the corresponding bits are set to 1 [11]. BF based PPRL techniques have the advantage of being efficient and facilitate the linkage of large real-world databases, where approximate matching is crucial due to errors and variations in attribute values [1].

As outlined in Algorithm 2, we use an adapted record level BF (RBF) encoding approach, where RBF has shown to be secure against frequency attacks [15]. In RBF the values of each attribute in a record are hash-mapped into different BFs, which are then concatenated to create a single BF for a record. The aim of using RBF is to allow the LU to adjust the similarities based on the calculated decay values for each attribute separately. Before generating RBFs, the number of bits required for each attribute $A \in \mathbf{A}$ is calculated (lines 2 to 4). Each DO$_i$ first calculates the list L_b^i of average attribute value length for each A (line 2). Once all DOs compute their L_b^is, each DO$_i$ sends its L_b^i to the other DOs to compute the list L_b of average attribute value length across all DOs. For example with Alice and Bob, $\forall_{j \in |L_b|} L_b[j] = \sum_{i \in \{A,B\}} L_b^i[j] / |\{A, B\}|$. Following [15] we then compute the corresponding number of bits for each attribute that needs to be selected from a BF of length l bits, and add these bit lengths to the list L_{bf} (line 4).

Next, each DO independently encodes its records into BFs and sends these BFs to the LU (lines 6 to 15). In line 6, each DO$_i$ loops over each record $r \in \mathbf{D}^i$ and first converts each attribute value $r.A$ into a set $S_q = \{q_1, q_2, \cdots, q_n\}$ of sub-strings of length q (line 9), known as q-grams [2]. Then each $q \in S_q$ is encoded into a BF b_A by using k independent hash functions, $\mathcal{H} = \{h_1, h_2, \cdots, h_k\}$, and all bits having index positions $h_j(s)$ for $j \in [1, k]$ in the BF are set to 1 (line 10). All BFs b_A as generated for the attributes $A \in \mathbf{A}$ are concatenated into the final BF b (line 11). As detailed in Section V, to improve the privacy the bit positions in each BF are permuted according to a random sequence $\mathcal{R}$ (not shared with the LU) which is agreed upon by all DOs (line 12). Each permuted BF b_p is added with its corresponding record time-stamp $r.t$ into an inverted index $\mathbf{B}_i$ using its corresponding record identifier $r.id$ as a key (line 13). Each DO$_i$ sends its $\mathbf{B}_i$ to the LU for comparison in the third phase (line 14).

In the second step of phase 2, a list of masking BFs, L_m, where each masking BF $b_m \in L_m$ is of length l, is generated by one DO to be sent to the LU, as illustrated in Fig. 1. The generation of masking BFs is based on selecting a random subset of BF bits for each attribute. The number of masking bits per attribute is calculated based on their corresponding decay values. The aim of these masking BFs is to allow the LU to adjust the similarity of a pair of BFs (b_i, b_j) according to the corresponding decays of their time distance $t_d = (r_i.t - r_j.t)$. To adjust the similarities separately for each attribute we generate each b_m as a concatenated bit vector. For each attribute $A \in \mathbf{A}$ we generate a bit vector segment b_A using the corresponding bit length l_A for A in L_{bf} and concatenate these bit vectors together to create a masking BF b_m of total length l. First all the bit positions in each b_A are set to 0. We calculate the number of bit that need to be set to 1 in b_A using the decay probabilities of A for a time interval $I \in \mathbf{T}_I$.

As can be seen in Fig. 2, since agreement decay ($d^{=}$) does not change over time we only consider disagreement decay

Algorithm 2: Bloom filter generation

Input:
- $\mathbf{D}^i$: Database of database owner i (DO$_i$)
- $\mathbf{A}$: List of attributes, $\mathbf{A} = [A_1, A_2, \cdots, A_M]$, $1 \leq M \leq |\mathbf{A}|$
- n_s: Sample size
- q, l: Length of a q-gram and a Bloom filter
- k: Number of hash functions
- $\mathcal{H}$: List of hash functions, $\mathcal{H} = [h_1, h_2, \cdots, h_k]$
- $\mathcal{R}$: Random permutation of bit positions, $|\mathcal{R}| = l$

1: $L_{bf} = [\,]$ // Initialize an empty list for attribute bit lengths
2: L_b^i = $getAttrValueLen(\mathbf{D}^i, \mathbf{A}, n_s)$ // Get attribute value lengths
3: L_b = $computeSummation(L_b^i)$ // Compute averages globally
4: L_{bf} = $compAttrBFLen(l, L_b)$ // Get BF length for each $A \in \mathbf{A}$
5: $\mathbf{B}_i = \{\}$ // Initialize an empty Inverted Index
6: **foreach** $r \in \mathbf{D}^i$ **do:** // Loop over all records in database
7: b = [] // Initialize an empty Bloom filter
8: **foreach** $A \in \mathbf{A}$ **do:** // Loop over each attribute
9: S_q = $genQgrams(r.A, q)$ // Generate set of q-grams
10: b_A = $genBF(S_q, L_{bf}, k, \mathcal{H})$ // Encode q-grams into the BF
11: b = $b.concatenate(b_A)$ // Concatenate the generated BF
12: b_p = $permute(b, \mathcal{R})$ // Permute the bit positions randomly
13: $\mathbf{B}_i[r.id]$ = $(b_p, r.t)$ // Add BF to the inverted index
14: DO$_i$ sends $\mathbf{B}_i$ to the LU
15: LU receives $\mathbf{B}_i$

Algorithm 3: Bloom filter comparison

Input:
- $\mathbf{B}$: List of inverted indexes of BFs, $\mathbf{B} = [\mathbf{B}_i], i \in [1, d]$
- L_m: List of masking Bloom filters
- $sim(,)$: Similarity function for comparing BFs
- s_t: Similarity threshold value
- λ, μ: Number of iterations and bit positions sampled for HLSH

1: $\mathbf{D^M} = [\,]$ // Initialize an empty list of matching record identifiers
2: $L_B = compLSHBlocking(\mathbf{B}, \lambda, \mu)$ // Perform LSH based blocking
3: **foreach** $B \in L_B$ **do:** // Loop over all generated blocks
4: **foreach** $(b_i, b_j) \in B : b_i \in \mathbf{B}_i, b_j \in \mathbf{B}_j, i \neq j$ **do**:
5: $t_d = r_i.t - r_j.t$ // Compute time distance between BFs
6: $b_m = getMaskingBF(L_m, t_d)$ // Get the relevant masking BF
7: $b'_i = b_i \wedge b_m$ // Conjunct masking BF with BF b_i
8: $b'_j = b_j \wedge b_m$ // Conjunct masking BF with BF b_j
9: $s = sim(b'_i, b'_j)$ // Compute similarity between masked BFs
10: **if** $s \geq s_t$ **then**: // Check if similarity above the threshold
11: $\mathbf{D^M}.add((r_i.id, r_j.id))$ // Add record identifiers into $\mathbf{D^M}$
12: LU sends the list of matching record identifiers $\mathbf{D^M}$ to DOs

($d^{\neq}$) as weights for the number of bits to be selected. Hence, the number of bits n_1 that need to be set in b_A for each attribute is calculated as $n_1 = (1 - d^{\neq}(A, I)) \cdot l_A$. As a result, a maximum of n_1 bit positions in b_A is set to 1. For example, if $d^{\neq}(A, I) = 1.0$ then n_1 is equal to 0 and $d^{\neq}(A, I) = 0.0$ then $n_1 = l_A$. All b_As of the interval I are concatenated and bit positions are permuted according to $\mathcal{R}$. This ensures the bit positions of each b_A are correctly aligned with bit positions in each permuted BF b_p for each attribute $A \in \mathbf{A}$. Each b_m is added into a list L_m and finally L_m is sent to the LU to be used in the comparison phase. As explained in Section IV-C, each BF b in a pair of BFs is conjuncted (logical AND) with the corresponding masking BF b_m based on t_d. This allows to mask a set of bit positions from each b from the similarity calculation, thereby reducing the similarity component of the different attributes according to their decay values.

C. Bloom Filter Comparison

As shown in Fig. 1, in the third phase the LU conducts the linkage upon the BFs of each DO as generated in phase 2. In order to prevent a full pair-wise comparison of each BF from a database with every BF of another database (which has a quadratic complexity), the LU uses a privacy-preserving blocking technique upon BFs to group them into blocks. We employ Hamming distance based locality sensitive hashing (HLSH) [15] which requires the two parameters λ (the number of iterations) and μ (the number of bit positions selected in an iteration). The LU then compares each BF pair in each block using a similarity based linkage technique to identify the record identifiers (IDs) that refer to the same entities. Algorithm 3 outlines the steps involved in this phase.

Algorithm 3 starts by initializing an empty list of matching record IDs $\mathbf{D^M}$ (line 1), followed by the generation of the list of blocks L_B using the function $compLSHBlocking()$ (line 2). Each block $B \in L_B$ contains one or more BFs from $\mathbf{B}_i$ from different DO$_i$s that share the same bit pattern for randomly selected λ bit positions. In lines 3 and 4 of the algorithm we loop over each block $B \in L_B$ and generate all unique pairs of BFs (b_i, b_j) in each block B where b_i and b_j are from different databases.

Next, the time distance t_d between the BF pair (b_i, b_j) is calculated as the difference between the corresponding time-stamps (line 5). In line 6, based on t_d the function $getMaskingBF()$ selects the appropriate masking BF b_m (line 6) which is then conjuncted with b_i and b_j to generate b'_i and b'_j, respectively (lines 7 and 8). We calculate the similarity s between the conjuncted BFs b'_i and b'_j using a similarity function, such as Hamming distance or Dice similarity [2]. If this similarity s is at least the minimum similarity threshold s_t (line 10) then the corresponding record ID pair $(r_i.id, r_j.id)$ of b_i and b_j is classified as a match and is added to the list $\mathbf{D^M}$. Finally, in line 12, the LU sends the list $\mathbf{D^M}$ to all DOs.

V. Discussion of the Protocol

We now analyze our protocol in terms of complexity and privacy. We assume d database owners (DOs) are participating in the protocol and each DO has a database with n_r records. We assume all the parties in the protocol are directly connected to each other through a secure communication channel.

A. Complexity Analysis

We compute the computation complexities for each step of the three phases of our protocol, which are shown in Fig. 1. We analyze these complexities in terms of a single DO and the LU independently. In phase 1 all the DOs need to participate to compute the list of time intervals $\mathbf{T}_I$. In the function $getInterval()$ in Algorithm 1, each DO samples n_s records from its database to compute a time interval $I \in \mathbf{T}_I$. This requires each DO to loop over each record in the selected sample to check if at least n_k entities can be found with two or more records that have time distances within I. This results in a $O(n_s \cdot n_I)$ complexity for each DO to generate n_I time intervals in the first step of phase 1.

In the next step each DO computes the number of times an entity changes the attribute values in its records and the number of times a pair of entities share the same value for each attribute $A \in \mathbf{A}$. In the function $compChangeCounts()$ in Algorithm 1, each DO loops over the attribute values of each attribute $A \in \mathbf{A}$ of the n_s records sampled from its database to calculate these counts. Hence, the second step of phase 1 is of $O(n_s^2 \cdot n_I \cdot |\mathbf{A}|)$ complexity, where $|\mathbf{A}|$ represents the number of attributes.

Once the change counts are computed, each DO then encrypts its list of these counts before sending the list to the LU. This requires $O(n_I \cdot |\mathbf{A}|)$ complexity in the third step of phase 1. Each DO sends its encrypted list of change counts to the LU in step 4 where the LU performs homomorphic addition on each change count in these lists. This homomorphic addition is of $O(d \cdot n_I \cdot |\mathbf{A}|)$ complexity. As shown in Fig. 1, in the last step of phase 1 each DO computes the decay probabilities using the summation of the lists of encrypted change counts. Hence, the function $computeDecay()$ in Algorithm 1 requires a complexity of $O(n_I \cdot |\mathbf{A}|)$ for computing decay probabilities for each attribute $A \in \mathbf{A}$ under each time interval $I \in \mathbf{T}_I$.

In the first step of the second phase of our protocol, all DOs encode the records in their databases into BFs and send their encoded databases to the LU. We assume each attribute $A \in \mathbf{A}$ has an average of n_q q-grams. Hence, the generation of BFs for a single database is of $O(n_r \cdot n_q \cdot n_k \cdot |\mathbf{A}|)$ complexity. Once the BFs are generated, in the second step of phase 2 one DO needs to generate the list of masking BFs to be used in the similarity calculation step in phase 3. The generation of masking BFs of length l is of $O(n_I \cdot l \cdot |\mathbf{A}|)$ complexity.

In the third phase the LU performs HLSH blocking [15] upon the BFs received from the DOs. This requires the LU to iterate over each BF in each encoded database. Each BF is added into (l/μ) blocks over λ iterations. Hence, the function $compLSHBlocking()$ in Algorithm 3 is of $O(d \cdot n_r \cdot \lambda \cdot l/\mu)$ complexity. By assuming each block $B \in L_B$ contains n_b BFs from different databases, $n_b(n_b-1)/2$ BF pairs need to be compared by the LU for each B. Therefore, the comparison of the list of blocks L_B is of $O(n_b^2 \cdot |L_B|)$ complexity.

B. Privacy

We analyze the privacy by assuming each DO is honest and does not collude with any other party that participates in the protocol. We assume the LU follows the *honest-but-curious* adversary model [1], [7], [10]. This is a common assumption in PPRL protocols [6], [7]. To consider the worst-case scenario, let us assume all the DOs and the LU have access to a publicly available database $\mathbf{G}$ where the private databases held by the DOs are subsets of $\mathbf{G}$, $\forall_{i\in[1,d]}\mathbf{D}^i \subset \mathbf{G}$.

In the time interval computation step in phase 1, each DO ensures that each time interval they compute contains records of at least n_k entities. This provides k-anonymous privacy ($k = n_k$) for the entities in each database as none of the DOs will be able to identify specific information about individual entities in each time interval [7]. For example, for the database of a given DO_i, if a given time interval I contains change counts only about a single entity that changes its last name then another DO could analyze the records in $\mathbf{G}$ to identify potential entities who have changed their last names in I. Hence, when $n_k \geq k$ neither of the DOs would be able to identify an entity in a database of another DO.

In the third phase of our protocol the LU applies a privacy-preserving blocking and comparison step upon the BFs it receives from all DOs. The LU does not know anything about the parameters used in the BF generation process and the random sequence R that has been used to permute the bit positions in these BF. Since the masking BFs are also permuted in the same way, without knowing R the LU cannot identify the sets of bit positions that have been allocated to each attribute $A \in \mathbf{A}$. Hence, even if the LU uses a frequency analysis using $\mathbf{G}$ upon the BFs it cannot learn any information about the attribute values that have been encoded in the BFs.

VI. Experimental Evaluation

We now provide the details of the experimental evaluation of our proposed temporal PPRL protocol. The programs and test datasets are available from the authors.

TABLE II
THE AVERAGE NUMBER OF ENTITIES WITH VALUE CHANGES IN THEIR RECORDS FOR DIFFERENT ATTRIBUTES IN THE DATABASES USED IN OUR EXPERIMENTAL EVALUATION.

Number of DOs	Number of entities with value changes				
	First Name	Last Name	Street Address	City	Zipcode
2	42,922	197,163	1,609,251	820,783	1,114,127
3	41,667	191,068	1,557,560	791,020	1,073,184
5	31,385	140,785	1,228,758	612,135	836,311
7	29,661	136,147	1,203,799	606,838	822,097
10	24,569	106,656	1,011,569	496,717	684,267

A. Experimental Setup

For experiments we used a real voter registration dataset (NCVR) from the US state of North Carolina (NC) (available from: http://dl.ncsbe.gov/). We have downloaded this dataset every second month since October 2011 to April 2018 (in total 25 datasets) and built a compound temporal dataset that contains over 8 million records of voters names and addresses. The records (about the same entity) in these datasets can be grouped into three categories: (1) *exact matching*: those records that are exactly matching with each other, (2) *unique*: those records that are only appearing in one database, and (3) *updated*: those records where at least one attribute value has changed. In these records we only used *first name*, *last name*, *street address*, *city*, and *zipcode* as the set of attributes $\mathbf{A}$, because these are commonly used for record linkage [6], [7].

From these NCVR datasets we extracted records for subsets including 2, 3, 5, 7, and 10 DOs by assigning each dataset to a DO in a round robin fashion. For example, with 2 or 10 DOs in total, each DO is assigned at least 12 and 2 NCVR datasets, respectively. We generated two variations of databases for each DO by extracting records from each database assigned to it. In the first variation (named as NC-20E) we extracted 20% of *exact matching* with all *unique* and *updated* records while in the second variation (named as NC-0E) we only include *unique* and *updated* records. Each database in NC-20E and NC-0E contains an average of 5,185,859 and 3,705,233 records, respectively. Table II provides an overview of attribute value changes in the databases we generated.

The scalability of our protocol is evaluated by using runtime. We measured the runtime required for each phase of our approach with different database sizes and different number of databases. The linkage quality of our protocol is measured by using precision and recall [2]. Precision is calculated as the ratio of the number of true matched BF pairs found against the total number of candidate BF pairs compared across databases, while recall is calculated as the ratio of the number of true matched BF pairs against the total number of true matched BF pairs across all databases.

To evaluate the privacy we used the recently proposed cryptanalysis attack by Christen et al. [19]. This attack aligns frequent BFs and plain-text values in a public database $\mathbf{G}$ to allow re-identification of the most frequent values encoded in these BFs. We conducted this attack assuming the worst-case

TABLE III
THE AVERAGE RUNTIME REQUIRED FOR EACH PHASE OF OUR APPROACH COMPARED WITH THE NON-TEMPORAL PPRL TECHNIQUE. RUNTIMES ARE SHOWN IN SECONDS AND K REPRESENTS 1,000 RECORDS.

Number of records	Our temporal PPRL approach			Non-temporal PPRL
	Decay Generation	Bloom filter Generation	Bloom filter Comparison	
10K	3,997.1	19.0	56.3	54.2
50K	4,005.4	96.0	286.4	282.5
100K	4,018.2	201.4	595.2	559.6
500K	4,053.7	998.7	2,899.7	2,788.3
1,000K	4,105.8	2,156.2	5,819.4	5,761.8
5,000K	4,567.3	9,876.8	25,923.5	28,805.6

scenario of the LU gaining access to a database $\mathbf{D}$ of a DO, where $\mathbf{G} \equiv \mathbf{D}$, and trying to re-identify the values in $\mathbf{D}$ by using the BFs the LU received from the DO. However, note that such an attack is highly unlikely in practice since the DOs do not send their own databases to any other party.

For comparison we used a non-temporal state-of-the-art PPRL technique proposed by Schnell et al. [11] as there are no existing PPRL technique we are aware of that can be used for temporal PPRL. This approach uses *cryptographic long-term key* (CLK) for encoding records into BFs. Each DO first converts the attribute values of each record in its database into a set of q-grams and then each q-gram is encoded using k hash functions into a BF. Each DO sends its BFs to the LU for comparison, and if the similarity of a pair of BFs is at least a given threshold (s_t) then it is considered as a match.

We set the sample size n_s to 50,000, and computed the list of time intervals from 1 to 10 years in one-year gaps and from 10 to 50 years in ten-year gaps in phase 1 by setting $n_k = 10$. We set the public (pk) and private (sk) key lengths to 128 bits. Following earlier work in PPRL [6], [7], [11], in phase 2 and the CLK encoding in non-temporal PPRL we set the BF parameters as l = 1,000 bits, k = 30 hash functions, and q = 2. In phase 3 for HLSH blocking we used parameter settings in a similar range as used by the authors [15], where the number of iterations is set to λ = 20 and the number of bits to be sampled from the BFs at each iteration is μ = 100. Following [11] we set the threshold s_t to 0.8 and used Dice coefficient as the function $sim()$ in Algorithm 3.

We implemented all the approaches using Python programming language (version 2.7.3). All experiments were run on a server with 64-bit Intel Xeon (2:4 GHz) CPUs, 128 GBytes of main memory, and running Ubuntu 14.04.

B. Results and Discussion

Table III show the scalability of our protocol in terms of average runtime required in each phase with different database sizes. As can be seen from this table, the runtime required in phase 1 does not vary with the database size because we use the same sample size n_s (n_s = 50,000 records) in the change count computation step for each database. As we expected the runtime required by a DO to encode its database in the second phase and the runtime required by the LU to compare the BF pairs in the third phase scale linearly with the number of records and the number of databases, respectively. However, we noted that our approach consumes more runtime compared to the non-temporal PPRL technique because each DO has to compute the decay probabilities in phase 1 which requires the comparison of $n_s(n_s - 1)/2$ record pairs.

Fig 3 shows the precision and recall of our approach for the linkage of different number of databases compared with the non-temporal PPRL technique. Figs. 3 (a) to (d) show that our approach achieves better precision compared to non-temporal PPRL (Figs. 3 (e) to (h)). Our approach achieves a precision of 0.913 while non-temporal PPRL achieves a precision of 0.802 for the linkage of 10 databases with five attributes in $\mathbf{A}$. This is because the use of masking BFs helps to adjust the similarities between pairs of BFs based on the time distance between them. We also noted that our proposed approach achieves higher precision even with an increasing number of attributes used in the linkage process. This is because the similarities of each pair of BFs are adjusted separately for each attribute which reduces the number of false positives in the BF comparison phase. Hence, the use of temporal information improves the overall linkage quality of our approach.

However, as shown in Figs (c), (d), (g), and (h), the recall drops in the linkage of the NC-0E databases compared to the NC-20E databases. This is because each record of an entity in NC-0E contains one or more attribute value changes that potentially results in the corresponding BFs of true matching record pairs to be grouped into different blocks in the HLSH step in Algorithm 3. Hence, further investigation is required to incorporate temporal information into blocking of BFs.

Finally, Fig. 4 shows the re-identification results from the cryptanalysis attack applied in the third phase of our approach. We evaluate the re-identification accuracy [19] of the attack by calculating (1) the percentage of correct guesses with 1-to-1 matching, (2) the percentage of correct guesses with 1-to-m (many) matching, (3) the percentage of wrong guesses, and (4) the percentage of no guesses, where these four percentages sum to 100. These four categories are labeled as 1-1 corr, 1-m corr, Wrong, and No in Fig. 4, respectively. We conducted the attack for different attribute (two to five) combinations.

As can be seen in this figure, in both temporal (Fig. 4 (a)) and non-temporal (Fig. 4 (b)) PPRL techniques the attack can exactly or partially re-identify a considerable percentage of encoded plain-text values when two attribute are used in the BF encoding process. This is because the frequencies of q-grams can be correctly identified as a lower number of q-grams is mapped to a certain bit position. Also we note that for two attribute combination more values are re-identified in our approach compared to non-temporal technique. This is because q-grams from different attributes are mapped to independent bit positions in RBF, and therefore the frequencies of these q-grams can potentially be analyzed. However, as we increase the number of attributes an attacker could not re-identify plain-text values as not enough frequency information is available to identify q-grams that are encoded in the BFs. Hence, conducting such cryptanalysis upon BFs that are encoded with q-grams from different attributes will unlikely be successful.

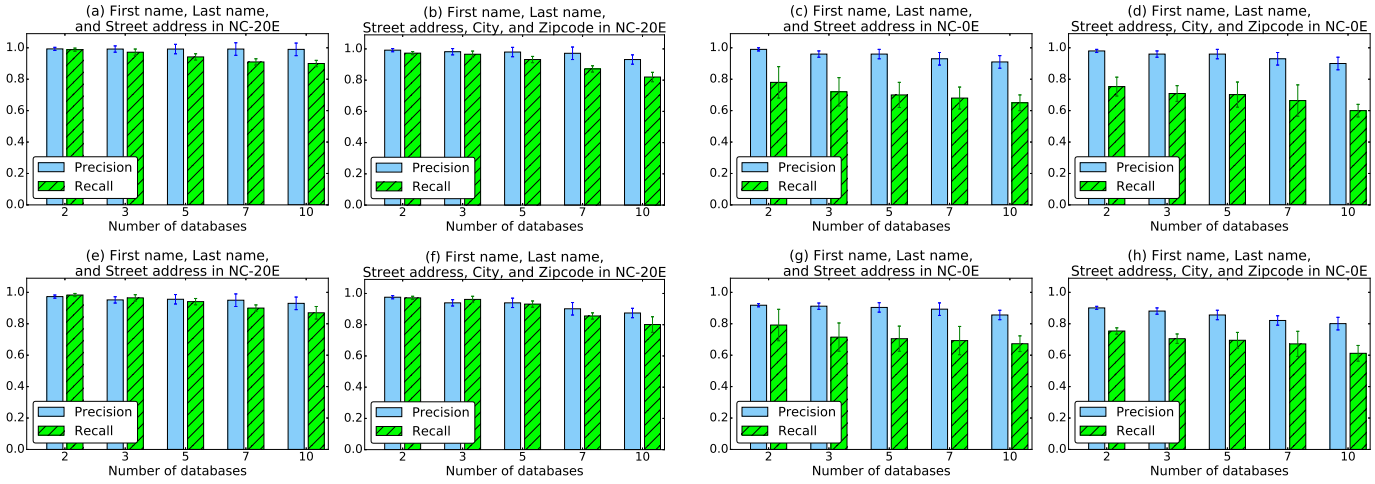

Fig. 3. The linkage quality results with different number of databases, where the left two columns and the right two columns show the results for the NC-20E and NC-0E datasets, respectively. The top row shows the results for our approach and the plots in the bottom row shows results for non-temporal PPRL.

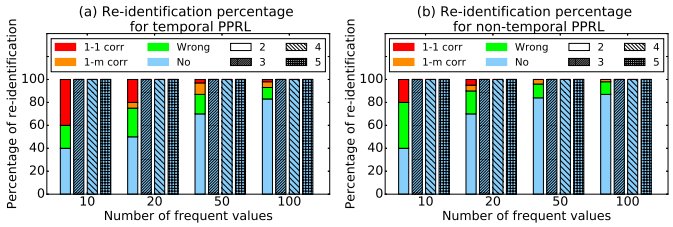

Fig. 4. Results of the cryptanalysis attack performed on the (a) temporal and (b) non-temporal PPRL approaches with different number of attributes.

VII. Conclusion

We have proposed a novel scalable privacy-preserving temporal record linkage protocol. Our approach consists of three phases, where in the first phase all database owners (DOs) securely compute decay probabilities using homomorphic encryption (likelihoods that an entity changes it attribute values and two entities share the same values for attributes over a given period of time). In the second phase all DOs encode their databases using Bloom filters (BFs) and send these BFs to a linkage unit (LU). In the third phase the LU uses a set of masking BFs that are generated based on the decay probabilities to adjust the similarities between pairs of BFs to identify the matching record pairs. Our experimental results showed that the proposed approach can achieve better linkage quality when incorporating temporal information in the linkage process compared to non-temporal PPRL while providing privacy to individuals in the databases that are being linked.

As future work we aim to investigate the use of active learning strategies [8] for learning decay probabilities to overcome the need of training data in phase 1 of our approach. We plan to incorporate other secure BF encoding mechanisms [6] in our approach to improve privacy. We also aim to experiment with different real data sets and PPRL techniques.

Acknowledgments

This work was funded by the Australian Research Council under Discovery Projects DP130101801 and DP160101934.

References

[1] S. M. Randall, A. M. Ferrante, J. H. Boyd *et al.*, "Privacy-preserving record linkage on large real world datasets," *JBI*, vol. 50, no. 0, 2014.
[2] P. Christen, *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.
[3] S. Randall, A. Ferrante, J. Boyd, and J. Semmens, "The effect of data cleaning on record linkage quality," *BMC Med Inform Decis Mak*, 2013.
[4] P. Christen and R. W. Gayler, "Adaptive temporal entity resolution on dynamic databases," in *PAKDD*. Springer, 2013, pp. 558–569.
[5] Y. Hu, Q. Wang, D. Vatsalan, and P. Christen, "Improving temporal record linkage using regression classification," in *PAKDD*, 2017.
[6] D. Vatsalan, Z. Sehili, P. Christen, and E. Rahm, *Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges*. Springer International Publishing, 2017, pp. 851–895.
[7] D. Vatsalan, P. Christen, and V. S. Verykios, "A taxonomy of privacy-preserving record linkage techniques," *JIS*, vol. 38, no. 6, 2013.
[8] P. Christen, D. Vatsalan, and Q. Wang, "Efficient entity resolution with adaptive and interactive training data selection," in *IEEE ICDM*, 2015.
[9] P. Li, X. L. Dong, A. Maurino, and D. Srivastava, "Linking temporal records," *VLDB Endowment*, vol. 4, no. 11, pp. 956–967, 2011.
[10] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *JPC*, vol. 1, no. 1, p. 5, 2009.
[11] R. Schnell, T. Bachteler, and J. Reiher, "Privacy-preserving record linkage using Bloom filters," *BMC Med Inform Decis Mak*, vol. 9, 2009.
[12] Y.-H. Chiang, A. Doan, and J. F. Naughton, "Modeling entity evolution for temporal record matching," in *ACM SIGMOD*, 2014, pp. 1175–1186.
[13] F. Li, M. L. Lee, W. Hsu, and W.-C. Tan, "Linking temporal records for profiling entities," in *ACM SIGMOD*, 2015, pp. 593–605.
[14] V. Christen, A. Groß, J. Fisher *et al.*, "Temporal group linkage and evolution analysis for census data," in *EDBT*, 2017, pp. 620–631.
[15] E. A. Durham, C. Toth, M. Kuzu, M. Kantarcioglu, Y. Xue, and B. Malin, "Composite Bloom filters for secure record linkage," *TKDE*, 2013.
[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*. Springer-Verlag, 1999, pp. 223–238.
[17] A. C. Yao, "Protocols for secure computations," in *IEEE SFCS*, 1982.
[18] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the Millionaires problem based on homomorphic encryption," in *Applied Cryptography and Network Security*. Springer, 2005, pp. 456–466.
[19] P. Christen, R. Schnell, D. Vatsalan, and T. Ranbaduge, "Efficient cryptanalysis of Bloom filters for privacy-preserving record linkage," in *PAKDD*, 2017.