Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

A Scalable Privacy-Preserving Framework for Temporal Record Linkage

Thilina Ranbaduge · Peter Christen

the date of receipt and acceptance should be inserted later

Abstract *Record linkage* (RL) is the process of identifying matching records from different databases that refer to the same entity. In many applications it is common that the attribute values of records that belong to the same entity 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 organisations 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 organisations. This paper proposes a *scalable framework for privacy-preserving temporal record linkage* that can link different databases while ensuring the privacy of sensitive data in these databases. We propose two protocols that can be used in different linkage scenarios with and without a third party. Our protocols use Bloom filter encoding which incorporates the temporal information available in records during the linkage process. Our approaches first securely calculate the probabilities of entities changing attribute values in their records over a period of time. Based on these probabilities we then 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 techniques and conduct an empirical study on large real databases containing several millions of records. The experimental results show that our approaches can achieve better linkage quality compared to non-temporal PPRL while providing privacy to individuals in the databases that are being linked.

Keywords Secure multiparty computation · Encryption · Temporal records

This paper is an extended version of the full paper published in the proceedings of the IEEE International Conference on Data Mining (ICDM) 2018 [20].

Thilina Ranbaduge
Research School of Computer Science, Australian National University, Canberra, Australia
E-mail: thilina.ranbaduge@anu.edu.au

Peter Christen
Research School of Computer Science, Australian National University, Canberra, Australia
E-mail: peter.christen@anu.edu.au

1 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 [2, 24]. 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 organisations collect vast amounts of data in their databases it is becoming increasingly challenging to integrate and combine data from different sources [23]. 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 [3, 10].

In the real world, RL is challenged because unique identifiers across the databases to be linked are not always available. Therefore, the use of personal identifiers (known as *quasi-identifiers* [33]), 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 organisations [32]. Often organisations are not willing or authorised 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 [32]. PPRL allows the linkage of databases without the need of any private of confidential information to be shared between the participating organisations 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 for 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 organisations. 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 [32].

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 lead 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 [4].

Table 1 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	Mary	Miller	161 Main Road Sydney	2002-09-11
r_2	e_2	Anne	Smith	43 Town Place Canberra	2005-05-23
r_3	e_2	Ann	Miller	43 Town Place Canberra	2007-04-10
r_4	e_1	Mary	Smith	23 Town Place Sydney	2007-11-05
r_5	e_2	Anne	Miller	12 Queen Street Melbourne	2010-12-21
r_6	e_3	Maryan	Miller	12 Main Road Sydney	2012-02-11

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 names after getting married or divorced, as shown in Table 1. In both RL and PPRL, temporal information of records, such as the time when a record was created or last modified in a database, is not used for similarity calculations [14]. 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 [14].

For example, given the 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 1, a linkage without considering temporal information would potentially classify r_1 and r_6 as matches because they have high attribute value similarities, and it would potentially classify records r_2, r_3, and r_5 to refer to different entities because they have different last names and street address values.

Temporal record linkage (TRL) matches records in databases while considering temporal information in the form of attribute values that evolve over time [14]. 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 between record pairs. In the example in Table 1, the aim of TRL is to correctly link record 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 record linkage, while at the same time ensuring the privacy of such information, is a problem that has not been studied so far.

To this end we propose a novel privacy-preserving framework for temporal record linkage that can be used to link databases using temporal information. The aim of our framework is to allow secure linkage of databases in any linkage scenarios. The linkage protocols proposed in our framework identify the records that belong to the same entity across databases owned by two or more organisations 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.

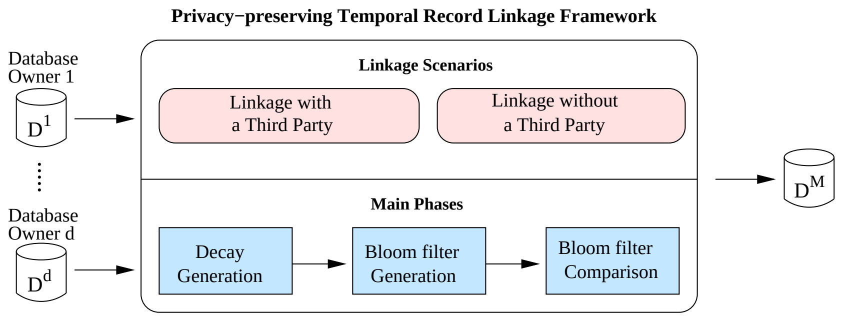


Fig. 1 Overview of our temporal PPRL framework. The framework accepts databases ($\mathbf{D}$) from d database owners ($d \geq 2$) and outputs a set of record pair identifiers $\mathbf{D}^M$ that are classified as matches. Our framework can be used in linkage scenarios with or without a third party to conduct the linkage. As illustrated our framework consists of three main phases. In the decay generation phase we calculate the probabilities that entities can change their attribute values. In the second phase, called Bloom filter generation, we encode attribute values in records into Bloom filters (BFs) and used the previously calculated probabilities to mask certain bit positions in these BFs based on their corresponding temporal distance. In the last phase, called Bloom filter comparison, we compare the masking BFs to identify matches and send the identifiers of matching records back to the database owners.

As shown in Figure 1, our framework supports two linkage scenarios which are (1) the linkage of databases with a third party and (2) the linkage of databases without a third party. In the first scenario a (trusted or semi-trusted) third party (which we call the *linkage unit*) is involved in conducting the linkage, while in the second scenario only the database owners participate in the linkage process. As we will explain in Section 4, the advantages of the second scenario over the linkage protocols with a third party is that it is more secure because there is no possibility of collusion between one of the database owners and the linkage unit, while the first scenario often has lower communication costs. However, the linkage scenario without a third party generally requires more complex techniques to ensure that the database owners cannot infer any sensitive information from each other during the linkage process [32].

Our framework accepts two or more databases as input and outputs a list that contains the identifiers of record pairs that are classified to belong to the same entity. As illustrated in Figure 1, the protocols proposed in our framework consists of three main phases. In the first phase, named *decay generation*, we calculate the probabilities (1) 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. These probability values are known as *decay values* [14]. Using these decay values we assign a weight to each attribute where the attribute weights are used to adjust the similarity between a pair of records.

As we will explain in Section 4, to compare the records of different databases securely we encode each record using Bloom filter (BF) encoding in the second phase of our framework, called *Bloom filter generation*. In the final phase, called *Bloom filter comparison*, we compare BFs from different databases and adjust their similarities according to the temporal distance of their corresponding records. We use these adjusted similarity values to decide if a pair of records is a match or a non-match using a similarity threshold based classifier.

Contribution: We propose a novel scalable linkage framework that can perform temporal record linkage between different databases while preserving the privacy of individuals represented by the records in these databases. We propose two linkage protocols that can be used in different linkage scenarios that are applicable in real-world applications. We use a secure multi-party computation technique to calculate decay values between different databases. We then propose an adapted Bloom filter (BF) encoding [25] and masking based technique to securely calculate the similarities between record pairs. We analyse the privacy of our protocols which shows no participating party can learn anything about the entities in the other databases. An empirical evaluation shows our protocols are scalable to large real databases which make our protocols applicable to Big Data applications. As we show in our experiments, our proposed protocols are also scalable to multiple databases which allows these protocols to be used in a multi-party context [32, 33]. To the best of our knowledge, no such temporal linkage framework has been proposed for PPRL.

Orgarnisation: The rest of this paper is structured as follows. First, in the following section, we provide a review of temporal record linkage approaches proposed in the literature and discuss why existing approaches cannot be used in the privacy-preserving temporal record linkage context. In Section 3 we formally define privacy-preserving temporal record linkage and the decay probabilities that can be used to adjust the similarities between record pairs based on their temporal distances. In Section 4 we describe the novel protocols we proposed for different linkage scenarios in our framework in detail. We theoretically analyse the complexities and privacy of the proposed protocols in Section 5. We then empirically evaluate and compare our approaches on real-world datasets in Section 6. Finally, in Section 7 we summarise our findings and discuss future research directions.

2 Related Work

The concepts of using temporal information for linking records was first introduced by Li et al. [14] 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 differences. The experiments conducted on a bibliographic database showed that using temporal information of records can improve linkage quality.

Christen and Gayler [3] proposed an approach to adaptively train a temporal model using a stream of temporal records. The proposed technique calculates the disagreement decay similar to [14], 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. [1] 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 [14] 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 [1] 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. 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. [10] proposed a linear regression model added into the decay value calculation process that uses a supportive attribute to calculate the decay for a given 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 a gender specific decay model for last name. Experiments conducted on a large 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. [14].

Another recently proposed approach by Christen et al. [6] utilises 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 summarise, 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 organisations 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 [32]. 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.

3 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 time 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 $r_j^i \in \mathbf{D}^i$, $1 \leq j \leq |\mathbf{D}^i|$, in the form of $(r_j^i.A_1, r_j^i.A_2, \cdots, r_j^i.A_m, r_j^i.t)$ where $r_j^i.t$ is the time-stamp associated with record r_j^i, and $r_j^i.A_m$, $m \in [1, M]$, is the value of attribute A_m at time t that represents an entity e in a set of entities $\mathbf{E}$, such that every record r_j^i 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. [14] 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 can also 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 calculate 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 [14]. In this paper we use labelled records that are available in a training dataset to calculate decays, assuming the DOs know the attribute value changes that occur in records that belongs to the same entity in their own databases. With such labelled data we can identify the number of occurrences where an entity changes its attribute value(s) and several entities share the same attribute values over a certain period of time. However, the need of training data is a limitation in both our approaches.

To this end we formally define the disagreement and agreement decays as follows [14]. Consider an attribute A and a time period Δt. Assume each entity $e \in \mathbf{E}$ 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 ($d^{\neq}$) [14] is the probability that an entity e changes its value for attribute A in the Δt time period. We calculate $d^{\neq}$ as follows.

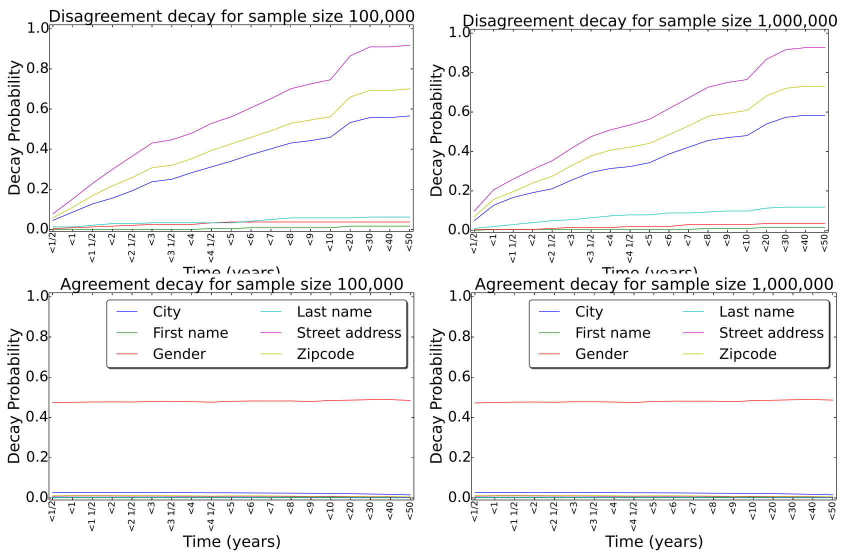


Fig. 2 Decay probabilities for different attributes based on samples of 100,000 and 1,000,000 records from the North Carolina voter registration database (described in Section 6). The top row shows disagreement decay probabilities while the bottom row shows agreement decay probabilities. As can be seen (and as expected [3]), disagreement decays increase monotonically with time while agreement decays do not change over time.

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

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

Definition 3. Agreement decay ($d^{=}$) [14] for Δt is the probability that two entities e_i and e_j, such that $e_i, e_j \in \mathbf{E}$ 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.

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

where $\mathbf{E}_a$ 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, $\mathbf{E}_a = \{(e_i, e_j) : \forall_{e_i, e_j \in \mathbf{E}} \ |r_i.A - r_j.t| \leq \Delta t \wedge r_i.A = r_j.A\} \cup \{(e_i, e_j) : \forall_{e_i, e_j \in \mathbf{E}} \ |r_i.A - r_j.t| \leq \Delta t \wedge r_i.A \neq r_j.A\}$ 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.

Figure 2 shows disagreement and agreement decay probabilities for different attributes from an experimental database described in Section 6. The aim of our framework 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 framework in more detail.

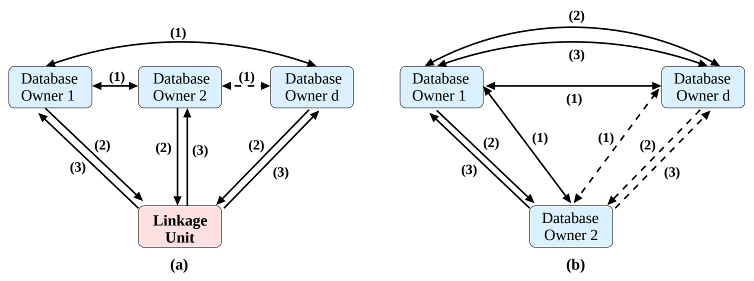


Fig. 3 PPRL linkage models, (a) with a linkage unit and (b) without a linkage unit. In both models the basic communication steps between the participating parties can be given as: (1) exchanging of parameter values, (2) communication of the (somehow) encoded record values of the databases, and (3) exchanging or sending the linkage results.

4 Temporal Privacy-Preserving Record Linkage

In PPRL, depending on the linkage model used, different parties can participate in the linkage process. A *linkage unit* (LU) is an external (third) party that participates in the linkage process that may or may not be external to the database owners(DOs) [21, 31]. In general, a LU does not have any data itself but conducts the linkage of the data sent to it by the DOs. As illustrated in Figure 3, by assuming each database that is to be linked is owned by a different DO, the PPRL protocol can be categorised as follows.

- **Linkage protocols with a LU**
  As shown in Figure 3 (a), DOs are performing PPRL on their databases through a LU [31]. In general the DOs start the process by sharing any required information such as parameters, pre-processing methods, encoding methods, secret keys, and so on. After the records are encoded they will be sent to the LU to perform the matching by calculating the similarities between the encoded records. The matching results are sent back to the DOs that can then either exchange the details of the matched records, or send only a selected set of attribute values of the matched records to a data consumer (such as a researcher).
  While in general such linkage protocols are more efficient due to the centralised processing of records at the LU, a weakness of this type of protocol is that the LU can potentially collude with a set of DOs to infer the attribute values of another DO. Also, sending all encoded records to the LU can potentially increase the risk of privacy attacks because the LU can use the frequency information of the encoded values sent to it by the DOs to try to identify sensitive plain-text values encoded in a database [32].
- **Linkage protocols without a LU**
  The linkage protocols without a LU are proposed as an alternative to overcome the drawback of requiring a LU in the PPRL process. As shown in Figure 3 (b), the DOs communicate directly with each other to perform the matching of their records [11, 31, 34]. This can make a PPRL protocol more secure compared to protocols that require a LU because there is a lower risk of collusion between parties. However, such protocols can become more complex and expensive in

terms of computation and communication due to their requirement of more sophisticated encoding or encryption mechanisms [31].

To this end, we propose a novel framework that consists of temporal linkage [20] protocols that can be applied in both linkage scenarios describe above. Without loss of generality, let us assume two DOs, Alice (DO_A) and Bob (DO_B), with their databases $\mathbf{D}^A$ and $\mathbf{D}^B$, respectively. We assume the DOs are honest and do not collude with any other party, and the LU follows a semi-honest adversary model [16]. We assume each database is not de-duplicated such that each entity can be represented by one or more temporal records in a database. Next, we describe the PPRL protocols that can be used under each linkage scenario.

4.1 Temporal PPRL with a Linkage Unit

Our first temporal protocol [20] allows database owners to link their databases with the assistance from the third party, the LU. Our protocol is designed to allow DOs to compute the decay values globally based on the attribute value changes of entities in their databases. As we described in Section 1, this protocol consists the three main phases of our framework, where Figure 4 shows the steps in each of these main phases.

1. *Decay generation*: 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 sent to the DOs to calculate 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 [25]. To assist the LU in the third phase of our protocol one of the DOs generates a list of masking BFs used to adjust the similarities of pairs of BFs to be 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*: The LU first applies a blocking technique [2] 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 [8]. 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 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 next, both DOs also agree upon a public key pk and a secret key sk to be used in a homomorphic cryptosystem [19] in phase 1 of our protocol. Next we discuss each phase in more detail.

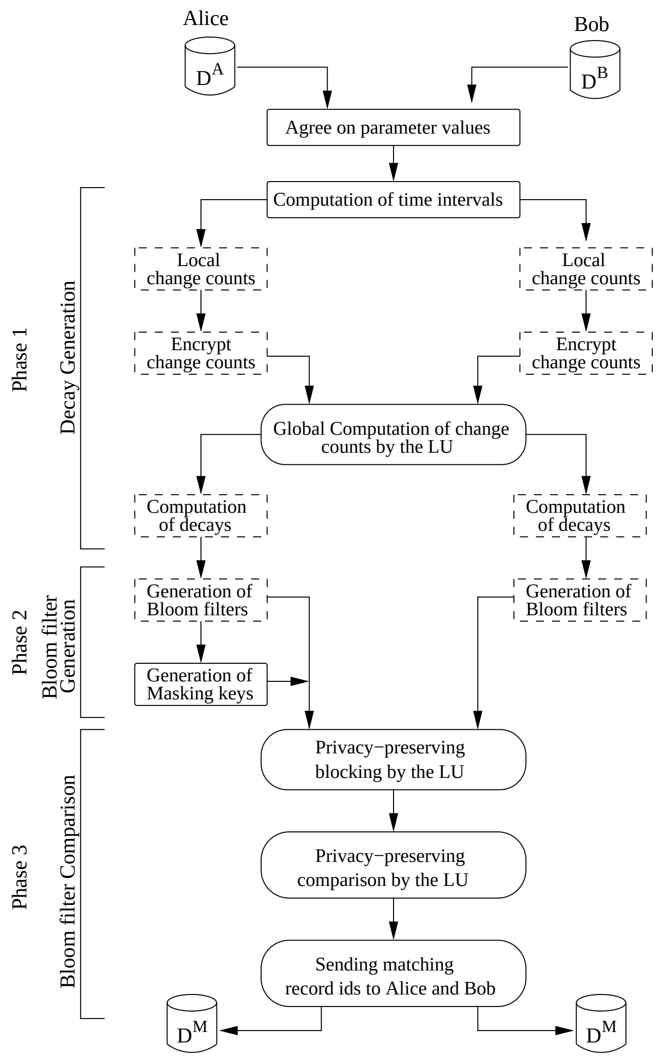

Alice
Bob
D^A
D^B
Agree on parameter values
Computation of time intervals
Local change counts
Local change counts
Encrypt change counts
Encrypt change counts
Global Computation of change counts by the LU
Computation of decays
Computation of decays
Generation of Bloom filters
Generation of Bloom filters
Generation of Masking keys
Privacy−preserving blocking by the LU
Privacy−preserving comparison by the LU
Sending matching record ids to Alice and Bob
D^M
D^M
Phase 1
Decay Generation
Phase 2
Bloom filter Generation
Phase 3
Bloom filter Comparison

Fig. 4 Overview of our temporal PPRL protocol with the LU with three main phases. As detailed in Section 4.1, 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 4.1, 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 ($\mathbf{D}^M$) is sent back to the DOs. Each step shown in boxes with rounded edges is performed by the LU. Rectangular boxes with dashed lines represent the steps that are performed similarly by the DOs.

Decay Generation

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

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: $\mathbf{T}_I = [\,]$                                     // Initialise 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 DOs
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:     $\mathbf{d}^{\neq}, \mathbf{d}^{=} = computeDecay(\mathbf{D}_c, \mathbf{T}_I, \mathbf{A})$       // Calculate decay values
```

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 upon the number of intervals, n_I, that need to be considered. The DOs follow 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 $genInterval()$ in 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 we will explain in Section 5, 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 created for 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 value(s) within I. This could potentially lead to the identification of e in the record comparison phase. As shown in Figure 4, 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* [35] 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 [15]. 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 $\mathbf{D}_c^i$ 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 as well as 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 Figure 2, given a large enough sample size, adding further records to the sample 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$_i$ 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 [15, 19] to add all encrypted $\overline{\mathbf{D}}_c^i$s into $\overline{\mathbf{D}}_c$ (line 10). The LU then sends $\overline{\mathbf{D}}_c$ to all DOs (line 11).

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 3) 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.

Bloom Filter Generation
As shown in Figure 4, 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 [25]. 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 [24].

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 [8]. 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 [8] we then compute the corresponding number of bits for each attribute that need 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 characters (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

Algorithm 2: Bloom filter generation by the DOs

```
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: $L_{bf} = []$   // Initialise 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: $\mathbf{B}_i = \{\}$   // Initialise an empty inverted index
6: foreach $r \in \mathbf{D}^i$ do:   // Loop over all records in database
7:    $b = []$   // Initialise an empty BF
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 $\mathbf{B}_i$
```

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 5, to improve 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 Figure 4. 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 bits 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 Figure 2, since agreement decay ($d^=$) does not change over time we only consider disagreement decay ($d^{\neq}$) as a weight 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

Algorithm 3: Bloom filter comparison by the LU

```
Input:
- $\mathbf{B}$:        List of inverted indexes of BFs, $\mathbf{B} = [\mathbf{B}_i], i \in [1, d]$
- $L_m$:       List of masking BFs
- $sim(,)$:    Similarity function for comparing BFs
- $s_t$:       Similarity threshold value
- $\lambda, \mu$:      Number of iterations and bit positions sampled for HLSH
1: $\mathbf{D^M} = []$                    // Initialise 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
```

0 and if $d^{\neq}(A, I) = 0.0$ 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 4.1, 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 masking of a set of bit positions from each b in the similarity calculation, thereby reducing the similarity component of the different attributes according to their decay values.

Bloom Filter Comparison

As shown in Figure 4, 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) [8] 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 [2] 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 initialising 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*() in line 2. Each block $B \in L_B$ contains one or more BFs from $\mathbf{B}_i$ from different DOs 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 the 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.

4.2 Temporal PPRL without a Linkage Unit

To overcome the need of a third party (the LU) in the linkage process, we now propose a second temporal protocol that requires only DOs to participate in the linkage process. In this protocol, the DOs independently compute the decay probabilities on their databases and use these probabilities to mask BFs before computing the similarities between BFs. Similar to the protocol proposed for DOs with a LU in Section 4.1, this protocol also consists of three main phases, where Figure 5 shows the steps in each of these main phases.

1. *Decay generation*: Similar to the protocol described in Section 4.1, the DOs individually compute a list of time intervals, $\mathbf{T}_I$. These time intervals are utilised to calculate the decay probabilities which are then used to adjust the similarity of pairs of records. Each DO then computes the attribute value changes of the entities in its database independently. In contrast to the temporal PPRL protocol with the LU, each DO uses its own attribute value changes 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 [25]. Using the decay probabilities calculated in the first phase, each DO generates a list of masking BFs used to adjust the similarities of pairs of BFs based on the temporal distance of the corresponding records.
3. *Bloom filter comparison*: In the third phase each DO first applies a blocking technique upon their BF databases and generates a set of blocks, $\mathbf{B}$ [32]. Any PPRL blocking technique can be used to generate the set of blocks for each database, as long as the same technique is used on all databases. Next the DOs participate in a secure multiparty computation protocol to compare the BF pairs in each common block $B \in \mathbf{B}$ (i.e. blocks that are similar enough between DOs to be compared [21, 22]). The DOs use a homomorphic encryption based technique to calculate the similarities between BFs which are then classified using a similarity based classification to identify the matching record pairs [12, 27].

Similar to the protocol described in Section 4.1, prior to these three phases the participating DOs agree upon the list of attributes $\mathbf{A}$ to be used, the parameters required for BF generation, including the BF length l, the q-gram length q (in characters), the number of hash functions k, and the minimum similarity threshold s_t to decide if a pair of BFs is a match or not. As we describe next, to perform homomorphic encryption [19] in the third phase, a DO generates a public and private key pair (pk, sk) and sends the public key pk to all other DOs. Next we discuss each of these phases in more detail.

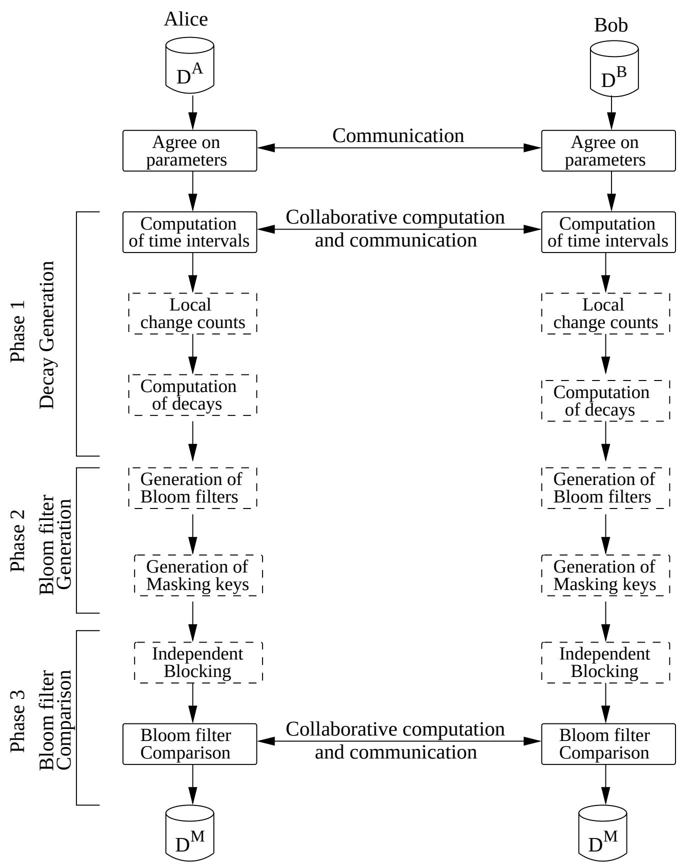

Alice
Bob
D^A
D^B
Agree on parameters
Communication
Agree on parameters
Phase 1 Decay Generation
Computation of time intervals
Collaborative computation and communication
Computation of time intervals
Local change counts
Local change counts
Computation of decays
Computation of decays
Phase 2 Bloom filter Generation
Generation of Bloom filters
Generation of Bloom filters
Generation of Masking keys
Generation of Masking keys
Phase 3 Bloom filter Comparison
Independent Blocking
Independent Blocking
Bloom filter Comparison
Collaborative computation and communication
Bloom filter Comparison
D^M
D^M

Fig. 5 Overview of our temporal PPRL protocol without the LU with three main phases. 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). Each DO calculates a list of masking keys independently based on the decay probabilities calculated in phase 1. In phase 3, each DO applies a blocking technique upon its BF database and generates a set of blocks. The DOs then participate in a secure multiparty computation protocol to compare the BF pairs in each common block and finally share the set of classified matched record identifiers ($\mathbf{D}^M$). Rectangular boxes with dashed lines represent the steps that are performed independently by the DOs, while the boxes with solid lines represent the collaborative computations or communications performed by the DOs.

Decay Generation

As illustrated in Figure 5, in the first phase of this new protocol each DO computes the agreement and disagreement decay probabilities independently from all other DOs. These probabilities are needed in the record comparison phase to adjust similarities between record pairs depending upon their temporal distances. Similar to the decay generation phase in the protocol described in Section 4.1, the DOs follow the same interval generation steps as in Algorithm 1.

As outlined in Algorithm 4, the DOs agree on the number of intervals, n_I, that need to be considered and iteratively compute the list of time intervals $\mathbf{T}_I$ using the function *genInterval*() (line 3). In this function each DO$_i$ ensures there are

Algorithm 4: Decay generation by the DOs

```
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
1: $\mathbf{T}_I = []$                                  // Initialise 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:             // Each DO computes decay independently
6:     $\mathbf{D}^i_c = compChangeCounts(\mathbf{D}^i, n_s, \mathbf{T}_I, \mathbf{A})$        // Get change counts
7:     $\mathbf{d}^{\neq}{}_i, \mathbf{d}^{=}{}_i = computeDecay(\mathbf{D}^i_c, \mathbf{T}_I, \mathbf{A})$     // Compute decay values
```

records of at least $n_k > 1$ entities available with their pairwise time distances that are within I_i to improve privacy.

Once the DOs have calculated the list of intervals $\mathbf{T}_I$, each DO$_i$ computes the list of change counts $\mathbf{D}^i_c$ independently (line 5) for each time interval $I \in \mathbf{T}_I$ using the function *compChangeCounts*() in line 6. Following Definitions 2 and 3, 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 $I \in \mathbf{T}_I$. Each DO first samples a set of records and computes the required change counts to improve the efficiency.

Once change counts are computed each DO$_i$ independently uses the function *computeDecay*() in line 7, to generate the list of disagreement ($\mathbf{d}^{\neq}{}_i$) and agreement ($\mathbf{d}^{=}{}_i$) decay probabilities (as discussed in Section 3) for each attribute $A \in \mathbf{A}$ for each interval $I \in \mathbf{T}_I$. These decays probabilities are used next in the second phase of the protocol to generate a set of masking BFs.

Bloom Filter Generation
As shown in Figure 5, the second phase of the protocol consists of two main steps. Similar to the protocol in Section 4.1, first each DO encodes its database into Bloom filters (BFs). Each DO follows the same steps in Algorithm 2, except in lines 14 and 15 where the generated BF database of each DO is not shared nor sent to any other party.

As outlined in Algorithm 2, an adapted record level BF (RBF) encoding approach is used to encode each record into a BF, where 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. As we discussed before, all DOs need to agree on the length l of the RBF, sub-strings of length q, the k independent hash functions, $\mathcal{H}$, and a random sequence $\mathcal{R}$ used in the BF generation process. Each DO$_i$ adds each of its generated BFs with the corresponding record time-stamp $r.t$ into an inverted index $\mathbf{B}_i$ using its corresponding record identifier $r.id$ as a key.

In contrast to the temporal PPRL protocol with the LU described in Section 4.1, in this protocol each DO generates a set of masking keys for each interval for each attribute $A \in \mathbf{A}$. For generating the set of masking keys for each attribute of a given interval $I \in \mathbf{T}_I$ each DO uses the corresponding disagreement decay probabilities of A it computes in the first phase. As we will explain in Section 5,

Algorithm 5: Bloom filter comparison by the DOs

```
Input:
- $\mathbf{B}$:        List of inverted indexes of BFs, $\mathbf{B} = [\mathbf{B}_i], 1 \leq i \leq d$
- $\mathbf{L_m}$:      List of masking BF lists, $\mathbf{L_m} = [L^i_m], 1 \leq i \leq d$
- $\mathbf{T}_I$:      List of time intervals
- $s_t$:               Similarity threshold value
- $pk, sk$:            Public and private key for encryption and decryption, only known to DO$_i$
1: foreach $\mathbf{B}_i \in \mathbf{B}, i \in [1, d]$ do:
2:   $L^i_B = compBlocking(\mathbf{B}_i)$          // Perform private blocking independently
3: foreach $(B_a, B_b) : B_a \in L^i_B, B_b \in L^j_B, B_a = B_b, i \neq j$ do:     // Loop over all blocks
4:   foreach $(b_x, b_y) \in (B_a, B_b) : b_x \in B_a, b_y \in B_b$ do:     // Loop over all BF pairs
5:     $I = getTempInterval(r_x.t, r_y.t, \mathbf{T}_I)$     // Get the time interval between BFs
6:     $b'_x, b'_y = getMaskedBFs(b_x, b_y, \mathbf{L_m}, I)$     // Conjunct BFs with masking BFs
7:     $s_p = compDotProduct(b'_x, b'_y, pk)$     // Compute secure dot product between BFs
8:     DO$_i$ compute similarity $s = 2 \cdot decrypt(s_p, sk)/(HW(b'_x) + HW(b'_y))$
9:     if $s \geq s_t$ then:                        // Check if similarity above the threshold
10:      DO$_i$ sends similarity $s$ to DO$_j$
```

such an independent masking key generation improves the privacy since each DO does not share its masking keys with the other DOs. Each DO$_i$ adds each masking key b_m into a list of masking keys L^i_m. As we describe next, each BF b_i of DO$_i$ in a pair of BFs (b_i, b_j) is conjuncted (bitwise AND) with its corresponding masking BF b_m based on the time distance t_d of corresponding records of b_i and b_j. This allows to mask a set of bit positions from each BF from the similarity calculation, thereby reducing the similarity component of the different attributes according to their decay values.

Bloom Filter Comparison

As shown in Figure 5, in the third phase the DOs collaboratively conduct the linkage upon their BFs generated in phase 2. The main aim of this phase is to compute the similarities BFs securely such that no DO can learn the attribute value(s) encoded in the BFs of the other DOs. Algorithm 5 outlines the steps involved in this phase.

Each DO starts the linkage process by performing an independent blocking on their BF database. As outline in line 2 of the algorithm, each DO can apply any suitable PPRL blocking technique upon their BFs [28, 32]. In blocking, BFs are grouped according to the blocking key values (BKVs) [2] and only the BFs with the same BKV (i.e. BFs in the same block) from different databases need to be compared and classified. However, the same blocking technique needs to be used by all DOs ensuring the same block structure is generated by all DOs on their BF database allowing the same blocks to be compared in the later steps of the algorithm. Each DO$_i$ adds its generated blocks to a list of blocks L^i_B (line 2).

Once the blocks have been generated by each DO the corresponding BFs in these blocks need to be compared. To identify the common blocks that need to be compared across DOs, each DO can share its list of BKVs with other DOs. Without loss of generality we assume each DO has the same list of BKVs after blocking their BF databases. In line 3 of the algorithm we loop over each pair of blocks (B_a, B_b) in L^i_B and L^j_B of DO$_i$ and DO$_j$, respectively. Next in line 4 we iterate through each BF pair (b_x, b_y) in a given pair of blocks (B_a, B_b) that needs to be compared such that $b_x \in \mathbf{B_i}$ and $b_y \in \mathbf{B_j}$, respectively.

Similar to the protocol described in Section 4.1, each DO needs to apply masking on its BFs before the BF is used in the similarity calculation. Since each DO needs to apply the masking process independently, it needs to identify the corresponding time distance between BFs b_x and b_y securely without revealing the actual time-stamp of its record to the other DOs. Hence the DOs participate in a secure multi-party computation protocol in line 5 of the algorithm.

In the function *getTempInterval()* both DOs first need to identify the BF with the lowest time-stamp. For this computation the DOs uses a homomorphic encryption based protocol proposed by Lin et al. [15] which follows *Yao's Millionaires problem* [35]. Next the DO with the lowest time-stamp computes a list of new time-stamps $\mathbf{T}'_I$ by adding each time interval $I \in \mathbf{T}_I$ identified in phase 1. Next, for each new time-stamp $t \in \mathbf{T}'_I$ the DOs follow the same computation they used to find the lowest time-stamp to identify the appropriate time interval I for a BF pair (line 5).

Next, each DO uses the identified time interval I to get the corresponding masking BF generated in phase 2 in the function *getMaskedBF()* in line 6. The masking BF is conjuncted with each corresponding BF to generate the masked BF. Each DO performs the masking process independently, such that BF pair b_x and b_y are masked into corresponding BFs b'_x and b'_y by DO$_i$ and DO$_j$, respectively.

Next the DOs calculate the Dice similarity [2] s between the conjuncted BFs b'_x and b'_y and classify the BF pair as a match if s is grater than or equal a minimum similarity threshold s_t agreed by both DOs. To calculate the similarity between the BFs securely we first need to calculate the inner dot product between b'_x and b'_y which provides the number of 1 bit positions common in both b'_x and b'_y. To perform this computation we utilised a *somewhat* homomorphic encryption scheme proposed by Naehrig et al. [18] along with a ciphertext packing technique proposed by Yasuda et al. [36].

The somewhat homomorphic scheme bases its security on the *ring learning with errors* problem [17] which ensures a given secret message cannot be distinguished from noisy data. For more details on this homomorphic scheme we refer the reader to [18]. The cryptanalysis packing method introduced by Yasuda et al. [36] improves the efficiency of homomorphic computations by allowing a number of secret messages to be encrypted into a single ciphertext and performing computations on this ciphertext.

To compute the inner product between two BFs, this packing method allows us to encrypt a binary vector as coefficients of a polynomial value and the inner product of two BFs can be performed homomorphically using a single multiplication operation upon these encrypted coefficients. For each BF we can compute two packed ciphertexts named *forward pack* and *backward pack* where the forward pack consists the encrypted coefficients while the backward pack contains the inverse form of the forward pack. To get the inner product of two BFs we need to multiply the forward pack of one BF with the backward pack of the other BF, where once decrypted the constant term in the resulting polynomial gives the inner product of these BFs. For more details on this packing technique and its security proofs we refer the reader to [36].

The function *compDotProduct()* in line 7 computes the inner product of b'_x and b'_y in our protocol. In this function DO$_i$ first generates the forward pack of b'_x and sends this pack to DO$_j$ with DO$_i$'s public key *pk*. Using *pk* DO$_j$ generates the backward pack of b'_y and performs the multiplication using the forward pack of

b'_x and the backward pack of b'_y. DO$_j$ then sends the encypted result s_p to DO$_i$ with the Hamming weight of b'_y (represented as *HW()* in Algorithm 5). DO$_i$ then computes the Dice similarity s between b'_x and b'_y using the decrypted value of s_p and the Hamming weights of b'_x and b'_y (line 8). If this similarity s is at least the minimum similarity threshold s_t (line 9) then DO$_i$ finally sends the similarity s to DO$_j$ indicating the compared BF pair b'_x and b'_y can be classified as a match. Once all the BF pairs are compared the corresponding matching records identifiers can be shared between the DOs for further analysis purposes.

5 Discussion of the Protocols

We now analyse our protocols in terms of complexity and privacy. We assume d database owners (DOs) are participating in each 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.

5.1 Complexity Analysis

We first discuss the computation complexities for each step of the temporal PPRL protocol with a LU described in Section 4.1. We analyse these complexities in terms of a single DO and the LU independently. As illustrated in Figure 4, 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 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 Figure 4, 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 phase 2 of our temporal PPRL protocol with a LU, all DOs encode the records in their databases into BFs and send their BF 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 [8] 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.

We now analyse the computation complexities of the temporal protocol without a LU as proposed in Section 4.2. As illustrated in Figure 5, in the first phase the DOs collaboratively compute the list of time intervals $\mathbf{T}_I$ and then compute the decay probabilities independently. Similar to Algorithm 1, in the function *getInterval*() in Algorithm 4, each DO samples n_s records from its database to compute a time interval $I \in \mathbf{T}_I$. Hence, each DO requires a computational complexity of $O(n_s \cdot n_I)$ to generate n_I time intervals in the first step of phase 1.

Next in lines 6 and 7 each DO samples n_s records from its database to compute change counts and decay probabilities, respectively. Hence, the function *compChangeCounts*() and *computeDecay*() in Algorithm 4 are of $O(n_s^2 \cdot n_I \cdot |\mathbf{A}|)$ and $O(n_I \cdot |\mathbf{A}|)$ complexities, where $|\mathbf{A}|$ represents the number of attributes. As described in Section 4.2, each DO follows the steps in Algorithm 2 to encode each of its records into BFs which requires a complexity of $O(n_r \cdot n_q \cdot n_k \cdot |\mathbf{A}|)$.

In the third phase of the temporal PPRL without a LU protocol, each DO blocks their BF databases independently. The computational complexity of the function *compBlocking*() in Algorithm 5 depends on the blocking technique used to generate the blocks. By assuming each DO generates the same number of blocks n_B and each block $B \in L_B^i$ of DO$_i$ contains $n_b = (n_r/n_B)$ BFs, then $n_b(n_b - 1)/2$ BF pairs need to be compared for each B. By assuming the generation of the forward and backward packs of a BF consumes a constant time, the comparison of lists of blocks L_B^i and L_B^j of DO$_i$ and DO$_j$, respectively, is of $O(n_B \cdot n_b^2)$ complexity.

5.2 Privacy

We analyse the privacy of our temporal PPRL protocol with a LU described in Section 4.1 by assuming each DO is honest and does not collude with any other party that participates in the protocol. We also assume the LU follows the *honest-but-curious* adversary model [16, 24, 32]. This is a common assumption in PPRL protocols [32, 33]. To consider the worst-case scenario, let us assume all DOs and the LU have access to a publicly available database $\mathbf{G}$ where the private databases held by the DOs are subsets of $\mathbf{G}$, $\forall_{i \in [1,d]} \mathbf{D}^i \subseteq \mathbf{G}$.

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 [26] $(k = n_k)$ for the entities in each database because none of the DOs will be able to identify specific information about individual entities in a time interval [32]. 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 analyse 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 a unique 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 $\mathcal{R}$ 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 $\mathcal{R}$ 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 [5] it cannot learn any information about the attribute values that have been encoded in the BFs.

We analyse the privacy of our temporal PPRL protocol without a LU described in Section 4.2 by assuming each DO follows the *honest-but-curious* adversary model [16, 24, 32]. Apart from the step of generating the list of time intervals and BF comparison, each DO performs all other step in the protocol independently without any collaboration with other DOs. As described above, each DO ensures that each time interval computation provides k-anonymous privacy [26] for the entities in its database, and therefore none of the DOs would be able to identify an entity in a database of another DO.

In the third phase of the temporal PPRL without a LU protocol, each DO performs blocking independently. Besides an initial agreement of parameter settings to be used, the blocking technique does not require any further communication between the DOs that would reveal information about their blocks. We also assume each block contains at least $k = n_r/n_B$ records, where n_r is the total number of records in a database $\mathbf{D}$ (assuming all databases are of size $n_r = |\mathbf{D}|$) and n_B is the number of blocks generated for $\mathbf{D}$, to ensure k-anonymous privacy [26]. Therefore, no DO can learn anything about any other DOs sensitive data during the independent blocking step.

In the BF comparison step, two DOs, DO_i and DO_j, compare the BF pairs in their common blocks. In this step let us assume DO_i generates a public and private keys, pk and sk, respectively, and sends pk to DO_j. As we described in Section 4.2, to compare a BF pair (b_x, b_y), where b_x and b_y belong to DO_i and DO_j, respectively, DO_i sends the forward pack of masked b_x to DO_j. Since DO_j does not know the private key sk and the masking key used by DO_i, DO_j cannot learn anything about b_x. DO_j next performs the multiplication of the forward pack of masked b_x with the backward pack of its masked BF b_y and sends the computed inner product result back to DO_i. Since the masking key used to mask bit positions of b_y is only known to DO_j (as explained in Section 4.2, each DO generates its own list of masking keys without sharing them with any other DO), DO_i cannot infer information about the bit positions of b_y that are set to 1. Hence, no DO can learn anything about the encoding of any other DOs sensitive data during the BF comparison step which makes the re-identification of attribute values difficult.

Furthermore, DO_j sends the Hamming weights of its masked BFs to DO_i to compute the Dice similarity of BF pairs that are to be compared. This can allow DO_i to conduct a frequency attack, where the frequency distribution of the Hamming weights is matched with the distribution of known attribute values, such as last names [5]. However, DO_i only receives the Hamming weights of masked BFs of DO_j where the masking depends on the temporal distances of the BF pairs that need to be compared. Hence, a frequency attack by DO_i becomes difficult because

Table 2 The average numbers of entities with value changes due to updated records for different attributes in the datasets (NC-20E and NC-0E) 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

the frequency distribution of the Hamming weights of the original BFs of DO_j is different from the Hamming weights of its corresponding masked BFs.

6 Experimental Evaluation

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

6.1 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 from October 2011 to April 2018 (in total 25 datasets) and built a compound temporal dataset that contains over 8 million records of voter names and addresses. The records (about the same voter) 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 dataset, and (3) *updated*: those records where at least one attribute value has changed over a two month period. We used *first name*, *last name*, *street address*, *city*, and *zipcode* as the set of attributes **A**, because these are commonly used for record linkage [32, 33].

From these NCVR datasets we extracted records to generate datasets for different subsets of DOs, 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 datasets for each DO by extracting records from each dataset assigned to it. In the first variation (named as NC-20E) we extracted 20% of *exact matching* records with all *unique* and *updated* records, while in the second variation (named as NC-0E) we only include *unique* and *updated* records. Each dataset in NC-20E and NC-0E contains the average numbers of 5,185,859 and 3,705,233 records, respectively. Table 2 provides an overview of attribute value changes in the updated records in the datasets we generated.

We evaluated the scalability of our protocol using runtime. We measured the runtime required for each phase of our approach with different database sizes and different numbers of databases. The linkage quality of our protocol is measured 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. We do not present F-measure results given recent work has identified some problematic aspects with this measure when used for record linkage [9].

To evaluate the privacy we used the recently proposed cryptanalysis attack by Christen et al. [5]. 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 scenario of an attacker 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 of another DO the attacker gained accessed to. However, note that such an attack is highly unlikely in practice since the DOs do not send their own databases to any other party. In the temporal PPRL protocol with the LU, we assume the LU acts as an attacker and tries to re-identify the attribute values encoded in the BFs it receives from a DO, while in the temporal PPRL protocol without the LU we assume one DO acts as the attacker.

For comparison with our temporal PPRL protocol with a LU we used a non-temporal state-of-the-art PPRL technique proposed by Schnell et al. [25] because there are no existing PPRL techniques we are aware of that can be used for temporal PPRL. This approach uses the *cryptographic long-term key* (CLK) approach for encoding records into BFs. In CLK, 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 into a BF using k hash functions. Each DO sends its BFs to the LU for comparison, and if the Dice similarity of a pair of BFs is at least a given threshold (s_t) then it is classified as a match [25].

To compare our temporal PPRL protocol without a LU, we used the non-temporal multidatabase PPRL technique proposed by Vatsalan and Christen [29, 30] as a baseline approach. This approach distributively computes the conjunction of a set of BFs from multiple DOs to perform privacy-preserving approximate matching for multidatabase PPRL. Once the conjuncted BF segments are computed by the respective DOs, a secure summation protocol is initiated among the DOs to securely sum the number of common 1-bits in the conjuncted BF segments, as well as the total number of 1-bits in each DOs BF. These two sums are then used to calculate the Dice coefficient similarity of the set of BFs. If the calculated similarity is at least a given threshold (s_t) then it is considered as a match. A filtering approach is employed to reduce the number of comparisons based on segment similarity, such that if a sub-set of BF segments of candidate BFs (as calculated by a respective DO) has a lower similarity compared to a segment similarity threshold (s_m) then the BFs do not have to be compared with any of the BFs from the other DOs. Following the settings used by the authors [29, 30], we set s_m the same as s_t so that each segment contributes the same to the overall BF similarity s_t. However, it is important to note that this approach cannot be used with two parties due to the use of a secure summation protocol [7] which makes the protocol only usable with three or more parties (DOs).

We set the sample size n_s of Algorithms 1, 2, and 4 to 50,000, and computed the list of time intervals of Algorithms 1 and 4 from 1 to 10 years in one-year gaps, and from 10 to 50 years in ten-year gaps by setting $n_k = 10$. We set the public (pk) and private (sk) key lengths to 128 bits [20]. Following earlier work

Table 3 The average runtime in seconds required for each phase of our temporal PPRL protocol with the LU compared with the non-temporal baseline approach [25].

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

in PPRL [25, 32, 33], in phase 2 and the baseline CLK encoding in non-temporal PPRL we set the BF parameters as l = 1,000 bits, k = 30 hash functions, and q = 2. Following [25] and [29] we set the threshold s_t in Algorithms 3 and 5 to 0.8, and used the Dice coefficient [2] as the function $sim()$ in Algorithm 3.

In phase 3 of our temporal PPRL with the LU protocol, for HLSH blocking we used parameter settings in a similar range as used by the authors [8], where the number of iterations is set to $\lambda = 20$ and the number of bits to be sampled from the BFs at each iteration is $\mu = 100$. For ease of comparison we applied the same blocking technique in the non-temporal technique proposed by Schnell et al. [25].

As used in the multidatabase approach proposed by Vatsalan and Christen [29, 30], we applied a Soundex-based phonetic blocking technique [2] for the blocking step in our temporal PPRL protocol without the LU. In this blocking technique the corresponding BFs in the blocks of each DO with the same phonetic codes are compared and classified together [12]. We used the attributes *first name* and *last name* as the blocking keys.

We implemented all the approaches using the Python programming language (version 2.7.3) and used the HElib library (available from: `https://github.com/shaih/HElib`) to implement the somewhat homomorphic scheme used in Algorithm 5. All experiments were run on a server with 64-bit Intel Xeon (2.4 GHz) CPUs, 128 GBytes of main memory, and running Ubuntu 16.04.

6.2 Results and Discussion

We now discuss the experimental results of our approaches. We first discuss the scalability and linkage quality of our temporal PPRL protocol with and without the LU. We also analyse the statistical significance of linkage quality results of our approaches compared with the baselines. Finally, we describe the resilience of our approaches against the cryptanalysis attack described in Section 6.1.

Scalability and linkage quality of temporal PPRL with the LU
Table 3 shows the scalability of our temporal PPRL protocol with the LU 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

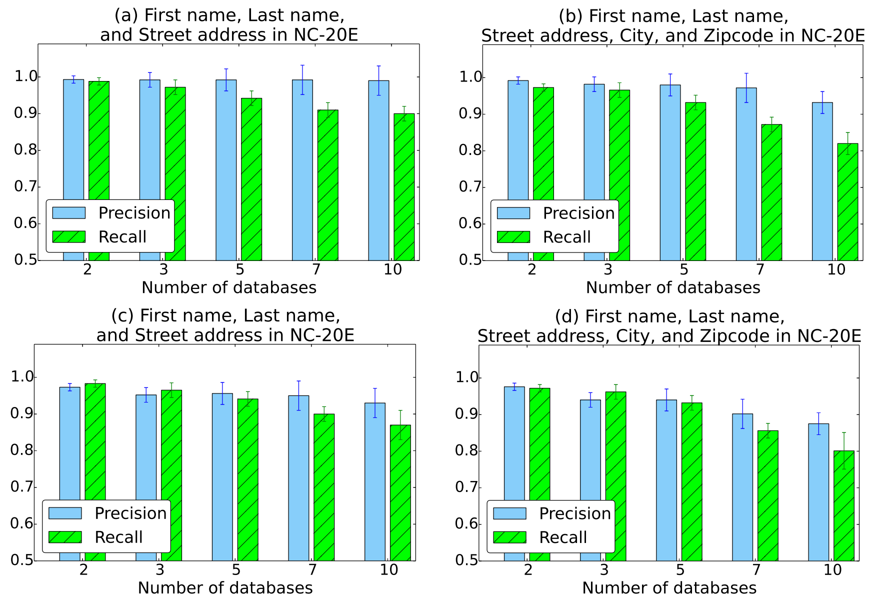


Fig. 6 The linkage quality results with different numbers of databases for the NC-20E datasets. The top row shows the results for our temporal PPRL approach with the LU, as described in Section 4.1, and the plots in the bottom row show results for the non-temporal baseline PPRL protocol proposed by Schnell et al. [25].

the number of records and the number of databases, respectively. However, we noted that our approach consumes more runtime compared to the non-temporal baseline approach [25] because each DO has to calculate the decay probabilities in phase 1 which requires the comparison of $n_s(n_s-1)/2$ record pairs.

Figures 6 and 7 shows the linkage quality of our temporal PPRL approach with the LU compared with the non-temporal PPRL technique with the LU. Figures 6 (a) and (b) and 7 (a) and (b) show that our protocol achieves better precision compared to the non-temporal approach proposed by Schnell et al. (Figures 6 (c) and (d) and 7 (c) and (d)) for the NC-20E and NC-0E databases, respectively.

As illustrated in Figures 7 (b) and (d) the proposed temporal PPRL approach achieves a precision of 0.913 while the non-temporal baseline approach achieves a precision of 0.802 for the linkage of 10 databases with five attributes in **A**, respectively. 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 noted that our temporal PPRL protocol with the LU achieves higher precision even with an increasing number of attributes used in the linkage process. This is because the similarities of all BF pairs are adjusted separately for each attribute which reduces the number of false positives in the BF comparison phase.

However, as shown in Figures 7 (a) to (d) the recall drops in the linkage of the NC-0E databases compared to the NC-20E databases shown in Figures 6 (a) to (d). 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 based blocking step

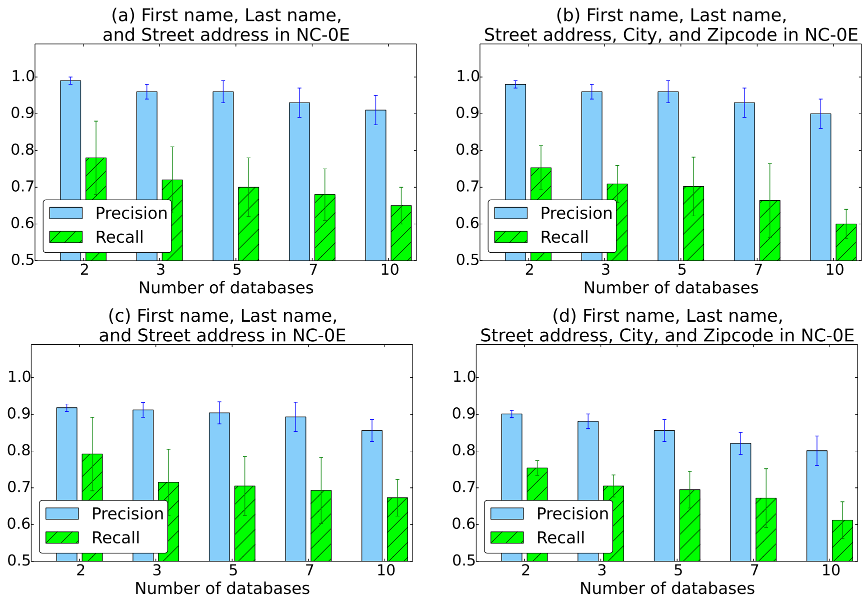


Fig. 7 The linkage quality results with different numbers of databases for the NC-0E datasets. The top row shows the results for our temporal PPRL approach with the LU, as described in Section 4.1, and the plots in the bottom row show results for the non-temporal baseline PPRL protocol proposed by Schnell et al. [25].

Table 4 The t-test results that compare the precision and recall values between our temporal PPRL approach with the LU and the non-temporal baseline approach [25] for different numbers of DOs for the NC-20E and NC-0E datasets.

	Precision					Recall				
Number of DOs	2	3	5	7	10	2	3	5	7	10
NC-20E	0.495	0.113	0.127	0.028	0.021	0.89	0.98	0.85	0.48	0.37
NC-0E	0.006	0.008	0.003	0.002	0.003	0.98	0.91	0.86	0.92	0.76

in Algorithm 3. Hence, further investigation is required to incorporate temporal information into the blocking of BFs.

As shown in Table 4, we conducted a t-test to evaluate the statistical significance between precision and recall values between our proposed temporal PPRL approach with the LU and the non-temporal baseline approach for the NC-20E and NC-0E datasets with different number of DOs. As can be seen, for precision the p-values ranged between 0.021 and 0.003 for 10 DOs with the NC-20E and NC-0E datasets, respectively. Such high statistical significance confirms that the use of temporal information can improve the overall linkage quality (precision) in our approach. Further, based on the obtained p-values it can be seen that the reduction of recall in our approach compared to the non-temporal baseline approach is not statistically significant. As shown in Table 4, we also noted the p-values for precision for 2, 3, and 5 databases are not highly significant with the NC-20E datasets. This is because the skewness of exact matching records in the NC-20E datasets which leads to high precision for both temporal and non-temporal approaches.

Table 5 The average runtime in seconds required for each phase of our temporal PPRL protocol without the LU compared with the non-temporal PPRL technique proposed by Vatsalan and Christen [29, 30].

Number of records	Our temporal PPRL approach without the LU			Non-temporal PPRL without the LU [29, 30]
	Decay Generation	Bloom filter Generation	Bloom filter Comparison	
10,000	3,782.5	19.0	183.8	3.7
50,000	3,855.1	96.0	786.1	32.3
100,000	3,981.9	201.4	1,635.2	189.1
500,000	4,001.7	998.7	5,879.7	828.9
1,000,000	4,075.4	2,156.2	9,718.3	4,824.1
5,000,000	4,212.5	9,876.8	46,634.1	–

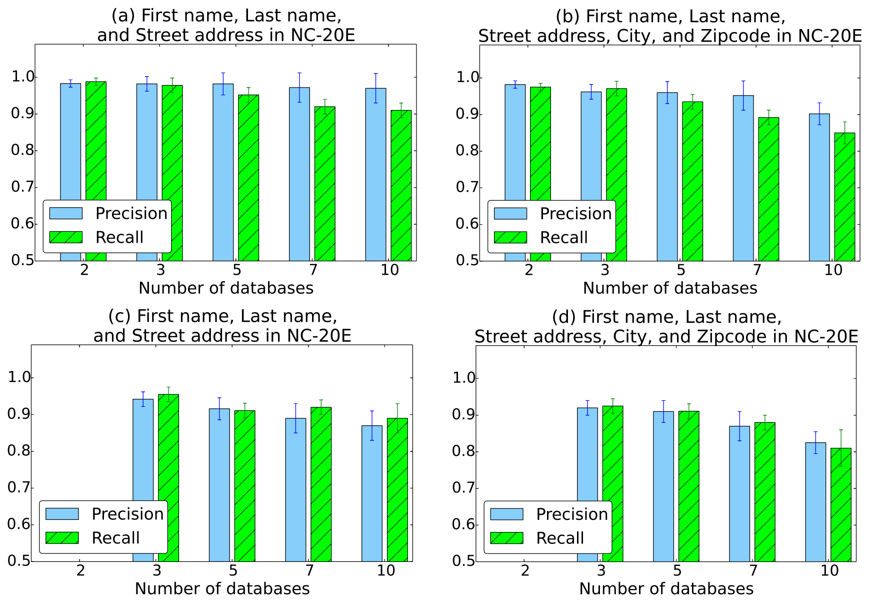


Fig. 8 The linkage quality results with different numbers of databases for the NC-20E datasets. The top row shows the results for temporal PPRL without the LU, as described in Section 4.2, and the plots in the bottom row show results for the non-temporal baseline PPRL protocol proposed by Vatsalan and Christen [29, 30] for 1,000,000 records.

Scalability and linkage quality of temporal PPRL without the LU

We next discuss the experimental results of our temporal PPRL approach without the LU compared with the non-temporal approach proposed by Vatsalan and Christen [29, 30]. Table 5 shows the average runtime required by a DO for each phase. As can be seen, in the decay generation phase our temporal PPRL protocol without the LU consumes less runtime compared to the temporal PPRL protocol with the LU (as shown in Table 3). This is because each DO computes the decay probabilities independently without any communication with other DOs.

Similar to our temporal PPRL with the LU protocol, the runtime required by the temporal PPRL protocol scales linearly in the BF generation and the BF comparison phases with the number of records and the number of databases, respectively. However, we noted that our temporal protocol consumes more runtime

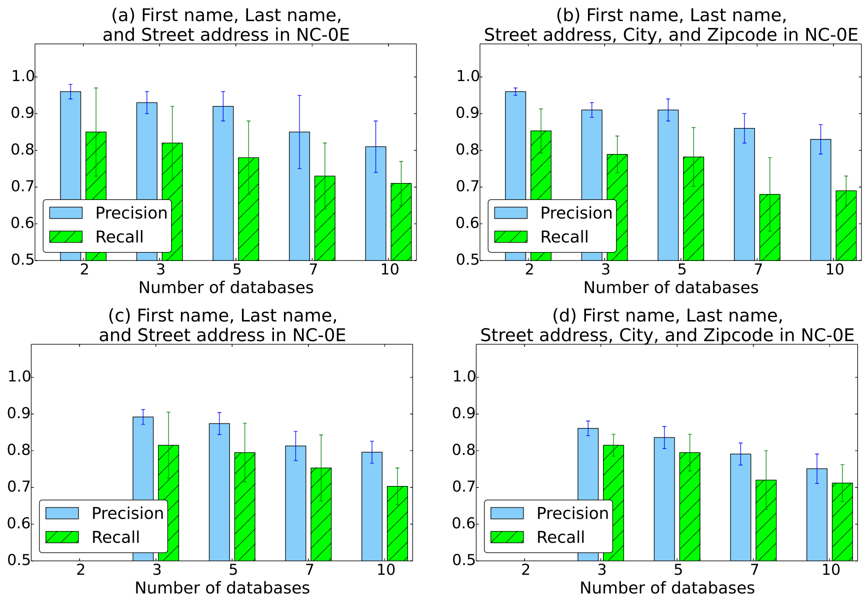


Fig. 9 The linkage quality results with different numbers of databases for the NC-0E datasets. The top row shows the results for temporal PPRL without the LU, as described in Section 4.2, and the plots in the bottom row show results for the non-temporal baseline PPRL protocol proposed by Vatsalan and Christen [29, 30] for 1,000,000 records.

Table 6 The t-test results that compare the precision and recall values between our temporal PPRL approach without the LU and the non-temporal baseline approach [29, 30] for different numbers of DOs for the NC-20E and NC-0E datasets.

	Precision					Recall				
Number of DOs	2	3	5	7	10	2	3	5	7	10
NC-20E	-	0.062	0.074	0.017	0.021	-	0.41	0.31	0.59	0.13
NC-0E	-	0.049	0.011	0.012	0.008	-	0.28	0.72	0.13	0.19

compared to the non-temporal approach by Vatsalan and Christen [29, 30]. This is due to the homomorphic encryption based computation required in the similarity calculation for each BF pair in Algorithm 5.

We also measured the memory required for each DO where on average our temporal PPRL protocol only uses between 30% to 40% of the memory required by the non-temporal PPRL approach by Vatsalan and Christen [29]. As shown in Table 5, we were unable to conduct experiments for the non-temporal PPRL approach with database sizes larger than 1 million records due to their memory requirements. Such memory consumption occurs due to the exponential number of BF comparisons and logical conjunctions the protocol calculates which limits the non-temporal protocol to a small number of databases.

We measured the linkage quality of our temporal PPRL protocol without the LU in terms of precision and recall for NC-20E and NC-0E dataset, as illustrated in Figures 8 and 9, respectively. Similar to the temporal PPRL protocol with the LU, our temporal PPRL protocol without the LU achieves higher precision even with an increasing number of attributes used in the linkage process compared to

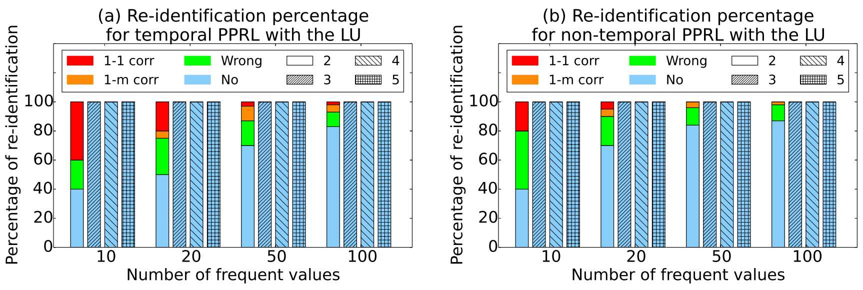


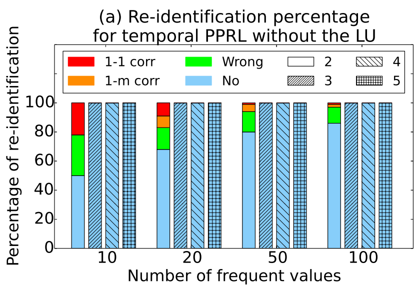
Fig. 10 Results of the cryptanalysis attack performed on the (a) temporal and (b) non-temporal PPRL approaches with the LU with different number of attributes.

the non-temporal PPRL approach. However, with our temporal PPRL protocol without the LU we noted an average of 5 to 10% reduction in precision in contrast to the temporal PPRL protocol with the LU (Figures 7 and 9). This is because the independent masking process by each DO can result in a matching BF pair to have a low similarity in the BF comparison phase.

Similar to our temporal PPRL with the LU approach, we conducted a t-test to compare precision and recall results of our temporal PPRL without the LU approach with the non-temporal baseline approach. As shown in Table 6, a high statistical significance can be seen for precision which suggests that temporal information can successfully be used in the linkage process to improve the linkage quality. Also, the p-values for recall show that the reduction in recall of our approach is not statistically significant compared to the non-temporal baseline approach. However, as shown in Figures 8 and 9, we also noted that recall is improved due to the phonetic blocking techniques we used in Algorithm 5 compared to the HLSH approach used in Algorithm 3. Overall it can be seen that the use of temporal information in the linkage process can improve the linkage quality compared to non-temporal approaches.

Resilience to a cryptanalysis attack
We performed the cryptanalysis attack [5] in the third phase of our temporal PPRL approach with the LU assuming the LU is conducting the attack on the masked BFs it received from DOs, while we assume a DO is conducting the attack on the masked BFs it received by another DO in our temporal PPRL approach without the LU. We evaluate the re-identification accuracy [5] 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 Figures 10 and 11, respectively. We conducted the attack for different numbers of attributes (two to five) and attribute combinations.

Figure 10 shows the re-identification results from the cryptanalysis attack applied in the third phase of our temporal PPRL protocol with the LU approach. As can be seen in this figure, in both temporal (Figure 10 (a)) and non-temporal (Figure 10 (b)) PPRL approaches 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.



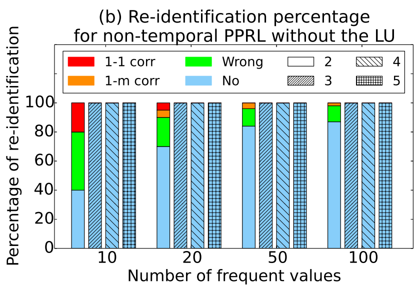


Fig. 11 Results of the cryptanalysis attack performed on the (a) temporal and (b) non-temporal PPRL approaches without the LU with different number of attributes.

Table 7 The number of 1-to-1 (1-1) and 1-to-many (1-m) correct re-identifications by the cryptanalysis attack with two attribute combinations for different numbers of frequent values with our temporal approaches and the non-temporal baseline approaches.

	With the LU				Without the LU			
	Temporal		Non-temporal		Temporal		Non-temporal	
Number of frequent values	1-1	1-m	1-1	1-m	1-1	1-m	1-1	1-m
10	4	0	2	0	3	0	2	0
20	4	1	1	1	2	1	1	1
50	2	5	0	2	1	3	0	2
100	2	5	0	2	1	2	0	2

We also note that for two attribute combinations more values are re-identified in our approach compared to the non-temporal approach. This is because q-grams from different attributes are mapped to independent bit positions in a RBF, and therefore the frequencies of these q-grams can potentially be analysed. However, as we increase the number of attributes an attacker could not re-identify any 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.

As shown in Figure 11 (a), the re-identification accuracy of the attack decreases in our temporal PPRL approach without the LU compared to the temporal PPRL protocol with the LU (as shown in Figure 10 (a)). This is because each DO masks its own database independently and the masking keys of a DO are not shared with any other party. We also noted that, in both temporal (Figure 11 (a)) and non-temporal (Figure 11 (b)) PPRL approaches, the attack can exactly or partially re-identify a similar percentage of encoded plain-text values when two attributes are used in the BF encoding process. This is because the masking process removes a certain amount of 1-bits from the encoded BFs in our temporal approach which results in certain q-grams to be incorrectly identified by the attack. Hence, this leads to more wrong re-identifications.

Table 7 shows the number of 1-to-1 and 1-to-many correct re-identifications by the cryptanalysis attach performed on our temporal approaches compared with non-temporal baseline approaches. As can be seen, similar to the temporal PPRL approach with the LU, for two attribute combinations the attack can still identify attribute values encoded in BFs. However, as we encode more attributes the success of the attack reduces as not enough frequency information is available to identify q-grams that are encoded in BFs. Furthermore it can be noted that the independent masking of BFs by a DO improves the privacy of entities compared to the masking by a third party LU. Hence, our temporal PPRL protocol without a LU is more applicable to linkage scenarios where the participating parties are semi-honest [16].

7 Conclusion

We have proposed a novel scalable privacy-preserving framework for temporal record linkage. Our framework consists of two protocols that can be used in different linkage scenarios (with and without a third party) that are applicable in real-world applications. Each protocol has three main phases, where in the first phase all database owners (DOs) securely compute decay probabilities (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 filter (BF) encoding. In the third phase we use a set of masking BFs that are generated based on the decay probabilities calculated in the first phase to adjust the attribute similarities between pairs of BFs to identify the matching record pairs. Our experimental evaluation with real databases showed that the proposed temporal approaches can achieve better linkage quality when incorporating temporal information into the linkage process compared to non-temporal PPRL approaches 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 [4] for learning decay probabilities to overcome the need of training data in the first phase of our approach. We plan to incorporate other secure BF encoding mechanisms [33] in our approach to improve privacy. Extending our protocol to dynamic databases [3], where records are added and updated dynamically, is another research direction that needs further investigation.

Acknowledgements This work was funded by the Australian Research Council under Discovery Project DP160101934.

References

1. Chiang YH, Doan A, Naughton JF (2014) Modeling entity evolution for temporal record matching. In: ACM SIGMOD, pp 1175–1186
2. Christen P (2012) Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer
3. Christen P, Gayler RW (2013) Adaptive temporal entity resolution on dynamic databases. In: PAKDD, Springer, pp 558–569
4. Christen P, Vatsalan D, Wang Q (2015) Efficient entity resolution with adaptive and interactive training data selection. In: IEEE ICDM
5. Christen P, Schnell R, Vatsalan D, Ranbaduge T (2017) Efficient cryptanalysis of Bloom filters for privacy-preserving record linkage. In: PAKDD
6. Christen V, Groß A, Fisher J, Wang Q, Christen P, Rahm E (2017) Temporal group linkage and evolution analysis for census data. In: EDBT, pp 620–631

7. Clifton C, Kantarcioglu M, Vaidya J, Lin X, Zhu M (2002) Tools for privacy preserving distributed data mining. SIGKDD Explorations 4(2):28–34
8. Durham EA, Toth C, Kuzu M, Kantarcioglu M, Xue Y, Malin B (2013) Composite Bloom filters for secure record linkage. TKDE
9. Hand D, Christen P (2018) A note on using the F-measure for evaluating record linkage algorithms. Statistics and Computing 28(3):539–547
10. Hu Y, Wang Q, Vatsalan D, Christen P (2017) Improving temporal record linkage using regression classification. In: PAKDD
11. Inan A, Kantarcioglu M, Ghinita G, Bertino E (2010) Private record matching using differential privacy. In: International Conference on Extending Database Technology, ACM, pp 123–134
12. Karakasidis A, Verykios V (2011) Secure blocking+ secure matching= secure record linkage. Journal of Computing Science and Engineering 5(3):223–235
13. Li F, Lee ML, Hsu W, Tan WC (2015) Linking temporal records for profiling entities. In: ACM SIGMOD, pp 593–605
14. Li P, Dong XL, Maurino A, Srivastava D (2011) Linking temporal records. VLDB Endowment 4(11):956–967
15. Lin HY, Tzeng WG (2005) An efficient solution to the Millionaires problem based on homomorphic encryption. In: Applied Cryptography and Network Security, Springer, pp 456–466
16. Lindell Y, Pinkas B (2009) Secure multiparty computation for privacy-preserving data mining. JPC 1(1):5
17. Lyubashevsky V, Peikert C, Regev O (2012) On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, `https://eprint.iacr.org/2012/230`
18. Naehrig M, Lauter K, Vaikuntanathan V (2011) Can homomorphic encryption be practical? In: 3rd ACM Workshop on Cloud Computing Security Workshop, ACM
19. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT, Springer-Verlag, pp 223–238
20. Ranbaduge T, Christen P (2018) Privacy-preserving temporal record linkage. In: IEEE ICDM, pp 1161–1171
21. Ranbaduge T, Vatsalan D, Christen P (2014) Tree based scalable indexing for multi-party privacy-preserving record linkage. In: AusDM, CRPIT 158, Brisbane
22. Ranbaduge T, Vatsalan D, Christen P (2015) Clustering-based scalable indexing for multi-party privacy-preserving record linkage. In: PAKDD'09, Springer LNAI, Vietnam
23. Randall S, Ferrante A, Boyd J, Semmens J (2013) The effect of data cleaning on record linkage quality. BMC Med Inform Decis Mak
24. Randall SM, Ferrante AM, Boyd JH, Bauer JK, Semmens JB (2014) Privacy-preserving record linkage on large real world datasets. JBI 50(0)
25. Schnell R, Bachteler T, Reiher J (2009) Privacy-preserving record linkage using Bloom filters. BMC Med Inform Decis Mak 9
26. Sweeney L (2002) K-anonymity: A model for protecting privacy. International Journal of Uncertainty Fuzziness and Knowledge Based Systems 10(5):557–570
27. Vatsalan D, Christen P (2012) An iterative two-party protocol for scalable privacy-preserving record linkage. In: AusDM, CRPIT 134, Sydney, Australia

28. Vatsalan D, Christen P (2013) Sorted nearest neighborhood clustering for efficient private blocking. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, pp 341–352
29. Vatsalan D, Christen P (2014) Scalable privacy-preserving record linkage for multiple databases. In: ACM CIKM, pp 1795–1798
30. Vatsalan D, Christen P (2016) Multi-party privacy-preserving record linkage using Bloom filters. arXiv preprint arXiv:161208835
31. Vatsalan D, Christen P, Verykios V (2013) Efficient two-party private blocking based on sorted nearest neighborhood clustering. In: ACM CIKM, San Francisco, pp 1949–1958
32. Vatsalan D, Christen P, Verykios VS (2013) A taxonomy of privacy-preserving record linkage techniques. JIS 38(6)
33. Vatsalan D, Sehili Z, Christen P, Rahm E (2017) Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges, Springer International Publishing, pp 851–895
34. Yakout M, Atallah M, Elmagarmid A (2009) Efficient private record linkage. In: IEEE International Conference on Data Engineering, pp 1283–1286
35. Yao AC (1982) Protocols for secure computations. In: IEEE SFCS
36. Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T (2015) New packing method in somewhat homomorphic encryption and its applications. Security and Communication Networks 8(13):2194–2213