Rinku Dewri*, Toan Ong, and Ramakrishna Thurimella

Linking Health Records for Federated Query Processing

Abstract: A federated query portal in an electronic health record infrastructure enables large epidemiology studies by combining data from geographically dispersed medical institutions. However, an individual's health record has been found to be distributed across multiple carrier databases in local settings. Privacy regulations may prohibit a data source from revealing clear text identifiers, thereby making it non-trivial for a query aggregator to determine which records correspond to the same underlying individual. In this paper, we explore this problem of privately detecting and tracking the health records of an individual in a distributed infrastructure. We begin with a secure set intersection protocol based on commutative encryption, and show how to make it practical on comparison spaces as large as 10^{10} pairs. Using bigram matching, precomputed tables, and data parallelism, we successfully reduced the execution time to a matter of minutes, while retaining a high degree of accuracy even in records with data entry errors. We also propose techniques to prevent the inference of identifier information when knowledge of underlying data distributions is known to an adversary. Finally, we discuss how records can be tracked utilizing the detection results during query processing.

Keywords: private record linkage, commutative encryption

DOI 10.1515/popets-2016-0013
Received 2015-11-30; revised 2016-03-01; accepted 2016-03-02.

1 Introduction

Many large epidemiology studies combine data from geographically dispersed sources. The HMO Research Network consisting of 19 large health care delivery organizations across the United States is an example of a distributed network of medical data providers for such studies. In a setting of widely dispersed providers, concerns about individuals seen in multiple organizations are little or none. Hence, no attempt is made to detect overlapping patients. However, as large scale electronic health record networks are formed, regional nodes must be able to detect and track data on an individual from different local sources, ranging from disease registries, clinical and claims databases, electronic medical records, cohort studies, and case control studies. Failure to correctly identify the same patient seen at different locations causes estimates of the incidence of diseases to be misjudged, and negative outcomes to be underestimated [4]. Federated query tools can return more precise estimates of the number of patients matching a query once an initial linkage of overlapping patients is complete [41]. Overlaps ranging from 5% to 35% have been reported in multiple studies [16, 28, 41]. Linked records also provide a richer data source by enabling queries which can only be answered by collating a patient's data split across multiple institutions.

1.1 Federated query processing

Consider a typical distributed architecture for federated processing of health data queries (Figure 1). Participants of such an architecture establish regional "grid nodes" containing databases of standardized electronic health data. Authorized users can request data from partner grid nodes via a web-based query portal. The query portal transmits queries submitted by the user to a federated query portal (FQP). The FQP contacts each grid node selected by the user, and submits the user's query to those grid nodes. Query results are then compiled on the FQP and presented to the user. The FQP and each grid node maintain their own list of authorized groups and users, and interact via an intermediate authentication, authorization, and accounting (AAA) component. The SAFTINet architecture [37] is an example of such an architecture, and has been approved by multiple institutional review boards.

A regional grid node has a structure similar to the larger network. A regional FQP (rFQP) receives queries

***Corresponding Author: Rinku Dewri:** University of Denver, E-mail: rdewri@cs.du.edu
Toan Ong: University of Colorado, Denver, E-mail: Toan.Ong@ucdenver.edu
Ramakrishna Thurimella: University of Denver, E-mail: ramki@cs.du.edu

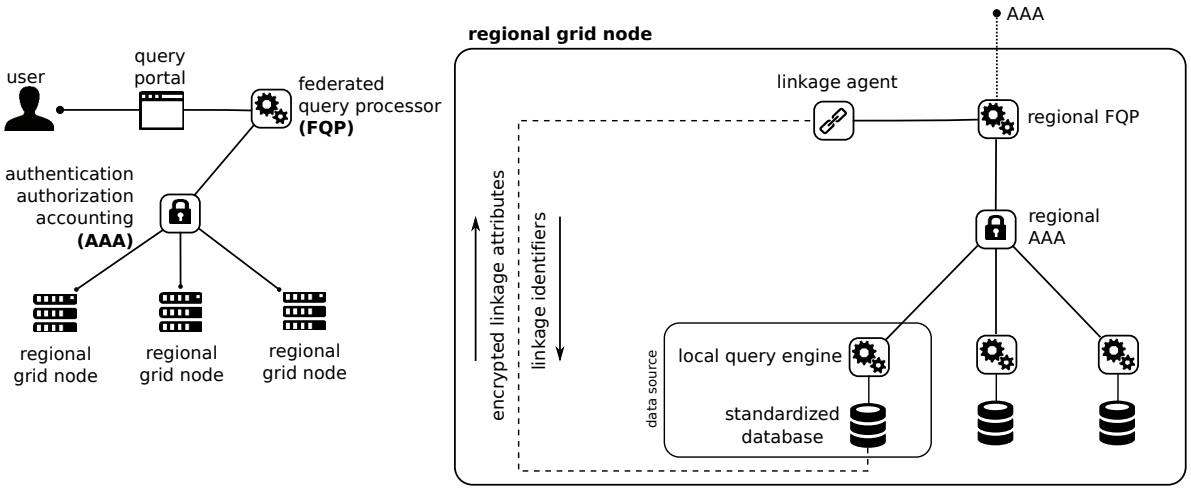


Fig. 1. Schematic for a distributed architecture for federated query processing.

from the FQP and compiles/forwards the results from multiple data providers in the region. Similarly, a regional AAA (rAAA) component manages connections between the rFQP and data providers. All query specific data in a result set are encrypted for consumption only by the querying user; no other component in the data flow can view the raw medical data (even when de-identified). Prior to becoming a data source for the rFQP, a data provider has to establish credentials with the rAAA, collate and create a unified view of an individual's record, convert electronic health data to research limited data sets, convert local codes to standardized codes, and replace direct identifiers with random identifiers. Altogether, each provider undergoes a common ETL process.

1.2 Linkage agent

An essential part of establishing a regional grid node is to resolve data overlap issues. The rFQP is required to deduplicate and join query results obtained from the data providers in order to ensure correctness. Therefore, the rFQP is supported by a linkage agent that detects the presence of distributed data records, and generates the information necessary to track such records over time. Federal entities such as the CDC's National Center for Health Statistics in the USA, or other agencies/institutions under their supervision, can play this role. In the simplest setting, records can be tracked by using a unique linkage identifier assigned to all records pertaining to the same individual. The problem that we address in this work is to *detect distributed records in databases containing records in the order of millions, often with inconsistencies, and with the constraint that the linkage agent is not authorized to view the content of data records.* We assume that the infrastructure is collusion free, strictly regulated, and is kept under systematic review.

1.3 Organization

We describe a basic detection algorithm in Section 2, followed by prior work done in this area in Section 3. Since we will be presenting our experimental results in conjunction with our proposals, we discuss our experimental setup early on in Section 4. Section 5 details our approach to perform linking using commutative ciphers, and our techniques to make it practical on large data sets. Later, we propose improvements to this approach in Section 6, with an assessment of the privacy and speed trade-off. We present the linkage performance of a greedy algorithm in Section 7. Discussion on how linkage results can be utilized by a rFQP is in Section 8. We conclude the paper in Section 9.

2 Detecting Distributed Records

The detection of distributed records is also known as *record linkage* or *record matching*. Record linkage is the process of identifying records about the same entity from multiple databases. The task is trivially accomplished by assigning a unique primary key to each entity, which then acts as an identifier for the entity across multiple databases. When the entity is a person, person-

ally identifying information (PII) such as social security numbers, name, address, sex, etc. may be used as identifiers, often in combination.

2.1 Linkage issues

Clear text record linkage is infeasible under a privacy preserving setting. The reluctance to perform record linkage using PII may emanate from multiple reasons—(i) the databases under consideration are distributed across multiple parties that are competing for the same consumer base, (ii) sharing of PII outside the organizational boundary is prohibited by regulations, or (iii) potential privacy risks exist if a third party gains knowledge of an entity's inclusion in a particular database. Therefore, private record linkage methods have been pursued. Such methods attempt to facilitate record linkage in such a way that no party is required to reveal clear text PII of its consumer base, and no party can learn the PII of a consumer that is exclusive to another party.

In a perfect world, a person's PII at two different databases will be exactly the same. In practice, data sites can have inconsistent representations of the same information, arising from data entry errors and differences in data coding systems. Non-exact (or *approximate*) matching is more appealing in this setting. For example, instead of comparing full strings for equality, a linkage method can break the strings into sets of bigrams (all 2 character substrings), and determine equality based on the extent of overlap of the bigram sets. In the next few subsections, we will first discuss a basic linkage algorithm based on bigram matching, and then pursue a practical and privacy-preserving version of the algorithm.

2.2 Similarity scores

Let $\mathcal{B}$ denote an ordered set of all possible bigrams, constructed from characters in A B...Z 0 1...9 ~ ‘ ! @ # $ % ^ & * ( ) − _ + = { } [ ] | \ : ; ” ′ < > , . ? / and the blank space. A total of 4761 bigrams is possible in $\mathcal{B}$. Let D_A and D_B denote two databases sharing z attributes, and owned by two sites S_A and S_B respectively. The objective is to perform a privacy-preserving join of the two databases based on the z attributes, given that data attributes can have data entry errors.

Record similarity. Let r_A and r_B be two records from D_A and D_B respectively. Every record has a record identifier assigned by the owning site. The similarity between the two records is measured as the weighted similarity across the z shared attributes, given as

$$S(r_A, r_B) = \sum_{i=1}^{z} w_i S_{Attr}(r_A[i], r_B[i]),$$

where w_i is the weight of attribute i, $r_A[i]$ ($r_B[i]$) denotes the value in attribute i of the record, and S_{Attr} is a similarity measure between two strings. Weight assignments are typically performed based on the discriminatory power of an attribute. For example, the sex attribute has less discriminatory power (only 2 possible values) compared to the day of birth (up to 31 possible values) attribute.

Attribute similarity. We will measure the similarity between r_A and r_B in a given attribute i based on the number of bigrams common in that attribute in the two records. Let α and β be the sets of bigrams obtained from $r_A[i]$ and $r_B[i]$ respectively. Then, the similarity is computed with the Dice's coefficient, given as

$$S_{attr}(r_A[i], r_B[i]) = \frac{2 \mid \alpha \cap \beta \mid}{\mid \alpha \mid + \mid \beta \mid}.$$

2.3 Greedy matching

A crucial step towards successful record matching is data standardization. This involves first converting attribute values to the same case (e.g. all uppercase) and format (e.g. date format and no punctuations). Second, attributes may be converted into sub-attributes, and linkage can be performed using the sub-attributes, with weights assigned such that matches on different parts carry different significance. For example, a name attribute can be converted into a first, middle and last name component, and then the last name is given more weightage than the rest. Finally, attributes and/or sub-attributes on which matching will be performed have to be chosen. Independence of attributes (e.g. ZIP code is not independent of city) and data quality in the attributes are considered in such decisions.

A matching algorithm performs a pairwise comparison of the records at two sites, and assigns a similarity score to each pair. The record pairs are then divided into three groups—*matches*, *non-matches* and *undecidables*. Matches are those pairs whose scores exceed a given upper threshold, non-matches have scores below a lower threshold, and remaining pairs are undecidables. We focus below on a procedure to find matching pairs

of records, although the method is not very different for other types. The algorithm we consider is greedy in linking; a linked record is not considered for linking in remaining matching attempts.

1. Choose a record r_A from D_A.
2. For all r_B in D_B, compute $S(r_A, r_B)$.
3. Let $r_B^* = \text{argmax}_{r_B} S(r_A, r_B)$. Ties are resolved by picking the first record found with the maximum score.
4. Given an upper threshold $\mathcal{T}_{upper}$, if $S(r_A, r_B^*) \geq \mathcal{T}_{upper}$, then output (r_A, r_B^*) as a match and remove r_B^* from D_B.
5. Remove r_A from D_A and repeat from step 1 until D_A is empty.

The core of this algorithm is in the computation of the matching score, which in turn requires the computation of the set intersection size of two bigram sets. The focus of this work is to make this computation efficient and privacy-preserving. As such, optimizing matching performance is not a priority in this work.

2.4 Using hashing

The use of cryptographic hash functions to protect identifying attributes have been proposed in the medical informatics community. Unfortunately, even minute differences in the attributes produce significant changes in the hash output for such functions. Hashing bigrams instead of full attribute values are also not useful in preventing inferences. To use a hash function for bigram set matching, each site will have to use the same hash function, and share any keys necessary for the function. Since the space of all bigrams is rather small, a site can enumerate over all possible bigrams and precompute the hash values for all bigrams. If hash values are leaked, a dictionary attack is simple to execute. Otherwise, the frequency of hash values observed at different sites can help distinguish between the bigrams.

3 Related Work

Most of the statistical underpinnings of record linkage are described in the seminal work by Fellegi and Sunter [15], who approached it as a parameter estimation problem to balance linkage error and human intervention. Extensive surveys, such as the one performed by Winkler [42], cover this traditional record linkage literature, which we shall omit in this review and focus on the emerging private record linkage domain.

Early proposals on performing private record linkage by applying one-way hash functions to identifiers can be found in the works by Dusserre et al. [13] and Grannis et al. [19]. However, Agrawal et al. invalidated the security guarantees of such a method on grounds of the possibility of dictionary or frequency-based attacks [3]. With access to a dictionary of possible inputs, any party can compute the hash value of each possible input, and identify the inputs corresponding to the outputs revealed by other parties.

Churches and Christen address the problem of input errors by breaking strings into a set of n-grams [8, 9]. Two attributes will be regarded the same if their n-gram sets have a "significant" overlap. The authors resort to a trusted third party to do the matching of the n-gram sets. Other methods have argued for the use of phonetic codes of words [24]. The n-gram approach has been shown to be superior to the phonetic method, and performs reasonably better than exact matching on real world patient identifier data [10]. These positive results have attracted the attention of developers/researchers implementing electronic health records sharing infrastructures for regional research projects. However, this approach provides no privacy guarantees since exact strings can be easily reconstructed from their n-grams. It is also susceptible to dictionary and frequency attacks if shared hash functions are used to encode the n-grams.

In order to counter the risk of reconstruction, Schnell et al. proposed a technique using Bloom filters for private linkage [39]. A Bloom filter is a binary array (values either 0 or 1) of a fixed size. Bits in the Bloom filter are set based on the output of hash functions applied to the n-grams. Bloom filters are then compared for similarity, instead of the n-grams. Kuzu et al. recently demonstrated an attack that can utilize a global dataset with demographic data (often available publicly) to determine what input strings were used to create the Bloom filter encodings [26]. In a follow up study, the authors demonstrated that the attack is feasible in practice, but the speed and precision of the attack may be worse than theoretical predictions [27]. Methods such as encoding n-grams from multiple identifiers into a single Bloom filter may provide resistance to such attacks, but can adversely impact linkage performance. Recently, Neidermeyer et al. proposed an easier attack and demonstrated that Bloom filter encodings can be broken without the need for high computational resources [31]. Unfortunately, the use of basic bloom fil-

ters is still being proposed in the medical informatics community as a privacy preserving method [34, 38]. In order to address frequency attacks on basic bloom filters, Durham et al. propose combining multiple Bloom filters by using a statistically informed method of sampling [11]. The method makes frequency attacks difficult, but requires the tuning of a security parameter that can affect linkage results. Note that Bloom filters can generate false positives (even though with a very small probability), owing to which the correctness of intersecting two n-gram sets may be affected. As such, we are motivated to compare n-gram sets without introducing a data structure that approximates the sets.

The proposal to use cryptographic primitives such as commutative encryption and homomorphic encryption for record linkage is not new. Agrawal et al. first proposed using commutative encryption to privately compute intersections of sets [3]. Malin and Airoldi use the protocol to find common values in the attributes of two records, but claimed that its use is not scalable for approximate matching using n-grams [30]. Adam et al. assume that unique patient identifiers are already present in the distributed records, and propose performing query joins on commutatively encrypted versions of those identifiers [1]. Lazrig et al. use homomorphic encryption on a similar problem [29]. Note that the private detection of distributed health records must be made in order to establish unique and uniform record identifiers across multiple data sources. Inan et al. claim that cryptographic operations are often slow for use in practical record linkage of large data sets. Towards this end, they propose separating data records into small subsets and then perform linkage only within a subset using homomorphic encryption [22, 23]. While encryption based techniques are known to provide stronger security and privacy guarantees, earlier works have disregarded its practical significance (or applied it in a limited manner) on grounds of heavy computation and communication complexity. To the best of our knowledge, this work provides the first evidence that, with careful implementation, record linkage using commutative encryption is very much achievable on large data sets, and that too using commodity hardware.

We note that a number of other proposals to perform private set intersection (PSI) exists in the "secure multiparty computation" community. However, we see a fundamental difference in the nature of the problem in this work and that treated in the area. Much of the research in PSI protocols is aimed at reducing the communication and computation cost of performing the intersection of two sets with "large" number of elements; our problem requires efficient protocols to perform many private intersections of small sets, with elements from a small domain. Pinkas et al. recently reported on the practical applicability of PSI protocols [32]. Based on the timing and bandwidth values reported in their work, PSI protocols are not yet suitable for performing the multiple set intersections (in the order of 10^{10}) necessary during the linking of two realistic data sets. It has been observed that secure protocols designed specifically for an application domain (under domain specific assumptions that can be made for the secure computation task) can be more amenable for practical usage than using a generic algorithm. For instance, Wang et al. recently demonstrated the use of garbled circuits to securely compute the edit distance between two genome sequences [40]. Their method involves the use of a set difference size computation, which can help determine a close approximate of the edit distance in the case of human genomes.

4 Experimental Setup

The data sets referred to in the subsequent sections are derived from a North Carolina Voters Registration database obtained in 2012 (ncsbe.gov/ncsbe/data-statistics). This database contains 7,088,370 individual records with demographic data. We use the name and street address attributes in this study. The name attribute is a concatenation of the first name, the middle name, and the last name. Similarly, the street address is composed of the house number, street direction, street name, and the street type, concatenated in that order. We standardized all records to use upper case characters, and removed any blank space appearing in a name or a street address. This database also reflects input errors typically seen in the real world.

A linking experiment requires two data sets, one corresponding to each site. We create such pairs of data sets based on two parameters–$\%overlap$: percentage of overlapping records in the two data sets, and $\%error$: the percentage of records that undergo synthetic error insertion. First, we sample (uniformly at random) the database for enough records to create two data sets with 100,000 records in each, ensuring that the required number of overlapping records exists in the two sets. Next, synthetic errors are inserted in a randomly selected set of records (based on $\%error$) in one of the data sets. In each record chosen for error insertion, up to three changes are made, decided randomly based on a proba-

bility of 0.6, 0.3 and 0.1 for one, two and three changes respectively. A change is randomly chosen to be one of many possibly operations: insert a character in an attribute, delete a character, substitute a character, transpose two characters, swap two sub-attributes (e.g. first name and last name), and remove a sub-attribute (e.g. street direction). These error types, although simple in nature, reflect the most common forms of typing errors [5]. Since it is not known which records in the original data set have actual data entry errors, these manually inserted errors will be our only source to evaluate how the linking method performs on records with errors. Using this process, we created ten data set pairs with the following $(\%overlap, \%error)$ values: (1,10), (5,10), (10,10), (15,10), (20,10), (25,5), (25,10), (25,20), (25,30), and (25,50).

We ran the linking experiments in two different machine platforms: (I) a 2014 assembled system with a 3.5GHz Intel Xeon E3-1246v3 CPU, 2×4GB 1600MHz DDR3 RAM, and running XUbuntu 14.04.3 LTS, and (II) a 2010 Mac Pro with a 2.8 GHz Intel Xeon W3530 CPU, 2×4GB 1066MHz DDR3 RAM, and running Mac OS X 10.10.3. Default processes are running in the background during timing experiments. Timing results in these two platforms will help us observe the impact of a few years of hardware advancement. Both processors support up to eight hardware threads using the Intel hyper-threading technology. As such, our implementation makes use of data parallelism and divides work equally between eight hardware threads. All programs are written in C, use the POSIX threading API and the GNU multi-precision (GMP) arithmetic library, and are compiled using gcc version 4.8.4 (Apple LLVM version 6.1.0 in the Mac Pro) with the -Ofast and -march=native options.

Since a linking algorithm needs to compare all record pairs across two sites for a possible match, a process known as "blocking" is often employed to reduce the comparison space to a subset of records that already agree on a simple attribute or a derived attribute (e.g. name initials) [7]. Attributes used for blocking are required to be fairly complete (not missing in many records), relatively cleaner, and preferably have a small domain of values that can be automatically corrected without ambiguity (e.g. city name). Figure 2 shows that performing name initials based blocking on the North Carolina population creates subsets of less than 100,000 records, while using the ZIP code for blocking can produce much smaller subsets of less than 50,000 records. Therefore, although we do not perform any blocking in this study, our data set size of 100,000 records is justified. Only a few empirical evaluations in record linkage have been reported to employ such large data sets; however, the methodology in such settings are based on hash functions and bloom filters (see Appendix B).

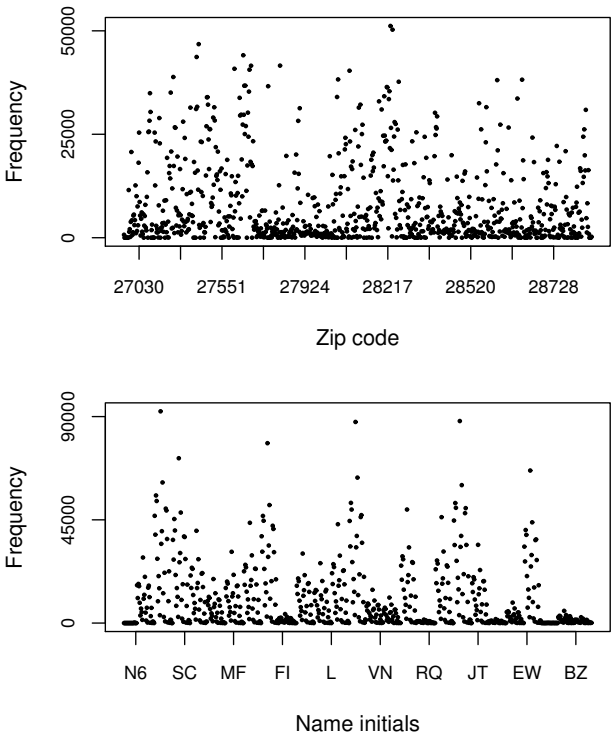


Fig. 2. Impact of blocking on the North Carolina voter's registration database (7,088,370 records).

5 Linking Using A Fixed Key Set

As noted in Section 2.4, the use of a shared key to hash bigrams serves no purpose. Therefore, we need a hashing procedure that can support individual secrets (keys) at each site. A commutative encryption scheme such as the Pohlig-Hellman exponentiation cipher is relevant here. The Pohlig-Hellman algorithm [33] is primarily an iterative algorithm to compute a discrete logarithm: given g and h from a finite field $GF(p)$, find integer $1 \leq x \leq p-1$ such that $h = g^x \mod p$. When g is a primitive root of p, the solution x is unique. The algorithm is effective only when p is the product of small prime numbers, i.e. p is smooth. Using the Pohlig-Hellman algorithm, as well as other known algorithms to compute discrete logarithms, it is computationally infeasible to find x when p is a "large" safe prime, i.e. $p = 2q + 1$, where q is also a prime number.

5.1 Pohlig-Hellman exponentiation cipher

The Pohlig-Hellman exponentiation cipher $E_K(m)$ of a message $1 \leq m \leq p-1$ when using the parameter $1 \leq K \leq p-2$ is $E_K(m) = m^K \mod p$, where p is a large safe prime and K is such that $\gcd(K, p-1) = 1$. K is also known as the key, and is never released to another site.

In our context, each bigram $b_i \in \mathcal{B}$ is mapped to a unique value $1 \leq g_i \leq p-1$ such that g_i is a primitive root of p. We denote this mapping by H. For a given key K, the cipher for b_i is then $E_K(b_i) = (H(b_i))^K \mod p$. Using primitive roots to represent bigrams ensures that the same cipher is not obtained for the same bigram from two different keys.

5.2 Determining common bigrams

Let r_A and r_B denote two records at sites S_A and S_B respectively, and $\alpha = \{b_{\alpha_1}, ..., b_{\alpha_m}\}$ and $\beta = \{b_{\beta_1}, ..., b_{\beta_n}\}$ be the sets (no duplicates) of bigrams corresponding to the values of an attribute in r_A and r_B respectively. S_A and S_B have decided on the mapping H and the prime p. A trivial protocol to privately compute $\mid \alpha \cap \beta \mid$ is as follows.

1. For every $(b_i, b_j) \in \alpha \times \beta$,
   (a) S_A and S_B pick their secret keys K_A and K_B respectively.
   (b) S_A computes $l_A = E_{K_A}(b_i)$, S_B computes $l_B = E_{K_B}(b_j)$, and exchange their computed values.
   (c) S_A computes $l_A^{'} = E_{K_A}(l_B)$, S_B computes $l_B^{'} = E_{K_B}(l_A)$, and exchange their computed values.
   (d) S_A and S_B conclude b_i matches b_j if $l_A^{'} = l_B^{'}$.
2. S_A and S_B determine $\mid \alpha \cap \beta \mid$ as the number of matches found in step 1.

Correctness. The above protocol is correct because modular exponentiation is commutative. For $(b_i, b_j) \in \alpha \times \beta$, $b_i = b_j$ if and only if $l_A^{'} = l_B^{'}$. If $b_i = b_j$,

$$\begin{aligned} l_A^{'} &= E_{K_A}\left(E_{K_B}(b_j)\right) \\ &= \left(H(b_j)^{K_B} \mod p\right)^{K_A} \mod p \\ &= (H(b_i))^{K_B K_A} \mod p \\ &= \left(H(b_i)^{K_A} \mod p\right)^{K_B} \mod p \\ &= E_{K_B}\left(E_{K_A}(b_i)\right) = l_B^{'}. \end{aligned}$$

Further, since H maps bigrams to different primitive roots, and different primitive roots will generate different values after the modular exponentiation with the same key, two different bigrams will not be encoded to the same value.

Privacy. We consider a "honest but curious" attacker model where H, p and the bigram distribution in the data set is public knowledge, and every site honestly follows protocol steps, but is curious to infer additional information, if possible. The semi-honest assumption is not unrealistic since parties participating in a health data exchange infrastructure are likely to be tied by multiple regulations, some of which may legally force them to maintain a code of conduct. Without loss of generality, site S_A can attempt to learn the bigrams in site S_B's set by analyzing the cipher values. Since all bigrams are represented by primitive roots, there will always be an exponent corresponding to every bigram that will generate a given cipher value. S_A can verify if an exponent satisfies the properties of a key ($\gcd(K, p-1) = 1$), but this would require solving a hard discrete logarithm problem to find the exponent in the first place. Knowledge of the bigram distribution is also not useful since S_B picks a random secret key K_B for each comparison.

We note that the computation of the set intersection in the protocol relies on the ability to compare two cipher text values (their equivalence implies plain text, or bigram, equivalence). As such, a semantically secure method of generating cipher texts will not be applicable in this protocol. In general, although Pohlig-Hellman resists known-plaintext attacks, guarantees are not possible for chosen-plaintext or chosen-ciphertext attacks. Therefore, we have to require that the linkage agent does not have access to either the encryption or the decryption agent of sites. We will later visit the case of known-ciphertext attacks and provide techniques to deter them. Private set intersection protocols with semantic security guarantees are available in the secure computation literature, e.g. [17], but suffer from the practical issues discussed in Section 3.

Considerations. Although the above process works in theory, there are limitations to consider when working with large data sets. For example, consider one of our data sets: 100,000 records in each site, with an average of 16 bigrams per record in the name attribute. As per the aforementioned protocol, there will potentially be 2.56×10^{12} bigram comparisons to be done! Assuming that p is a 2048-bit (256 bytes) prime, this amounts to the exchange of approximately 2.56×10^{12} comparisons$\times$4 transfers per comparison$\times$256 bytes per

transfer = 2.3 petabytes of data. Also, with an average of 2.7 milliseconds per modular exponentiation (average time for 10000 operations of the mpz_powm_sec function in the GMP library, executed in our XUbuntu platform), the modular exponentiations alone will require more than 54 years when run using eight hardware threads. While the computation time can be improved by using cryptographic transformations based on elliptic curves, the communication cost will be impractically high even with the corresponding smaller key size (224 bits).

5.3 Precomputing for a fixed key set

In order to make the above protocol practical on a large data set, we restrict the domain of keys at each site to a smaller set, and then precompute lookup tables for the modular exponentiations. Later, during the record comparison phase, a site picks keys from its set, and refers to pointers in the precomputed tables instead of performing the modular exponentiations. We next elaborate on the setup and execution of this protocol.

Key generation. Each site generates a set of w random keys, denoted by KS_A and KS_B for S_A and S_B respectively. We will use K_A^j to denote the j^{th} key with site S_A. Similarly, K_B^j for site S_B. Each key K follows the requirements stated earlier: $1 \leq K \leq p-2$ and $\gcd(K, p-1) = 1$. The set size w need not be same at each site as assumed here. It can be appropriately chosen based on the distribution of the bigrams appearing in an attribute. We shall discuss this in a later subsection.

Level-1 precomputation. The first level of precomputation is performed independently at each site. S_A and S_B decide on w (secret) permutations of the ordered bigram set $\mathcal{B}$. A permutation π is a mapping of the sequence $1, ..., |\mathcal{B}|$ to a shuffling of the same sequence. Therefore, each permutation produces a different shuffling of the elements in $\mathcal{B}$. The element at index i in $\mathcal{B}$ is at index $\pi(i)$ in the shuffling resulting from the permutation π. Let $\pi_A^1, \pi_A^2, ..., \pi_A^w$, and $\pi_B^1, \pi_B^2, ..., \pi_B^w$ denote the permutations decided by S_A and S_B respectively. Then, site S_A computes ordered set

$$L1_A = \{E_{K_A^j}(b_{\pi(i)}) \mid i \in \{1, ..., |\mathcal{B}|\}, K_A^j \in KS_A, \pi = \pi_A^j\}$$

and S_B computes ordered set

$$L1_B = \{E_{K_B^j}(b_{\pi(i)}) \mid i \in \{1, ..., |\mathcal{B}|\}, K_B^j \in KS_B, \pi = \pi_B^j\}.$$

We can index into an element of these sets by providing the permuted index of the bigram and a key index. For example, $L1_A(p, q)$ is the level-1 cipher for a bigram $b_i \in \mathcal{B}$ for which $\pi_A^q(i) = p$. S_A and S_B then exchange $L1_A$ and $L1_B$ respectively. With knowledge of the permutation functions, a site can still refer to a specific bigram of choice; however, without their knowledge, no site can infer which underlying bigram is referred to by the permuted index.

Level-2 precomputation. The second level of precomputation is specific to a pair of sites. Therefore, a site has to perform this precomputation for every other site with which it seeks to link records. Level-2 precomputation is the generation of ciphers using each key in a site's key set, and for values in the level-1 precomputation obtained from another site. Therefore, S_A computes ordered set

$$L2_A = \{E_{K_A^j}(l_B) \mid l_B \in L1_B, K_A^j \in KS_A\}$$

and S_B computes ordered set

$$L2_B = \{E_{K_B^j}(l_A) \mid l_A \in L1_A, K_B^j \in KS_B\}.$$

Indexing into elements of these sets require a permuted bigram index, a key index from S_A and a key index from S_B. For example, $L2_A(p, q, r)$ is a level-2 cipher created using the r^{th} key at site S_A, and for a bigram b_i for which $\pi_B^q(i) = p$.

Observe that, by nature of the commutativity of modular exponentiation, $L2_A(\pi_B^j(i), j, k) = L2_B(\pi_A^k(i), k, j)$, for all $i \in \{1, ..., |\mathcal{B}|\}$. If S_A obtains $L2_B$, it can cross-reference values in it with those in $L2_A$, and reverse the permutations used in $L1_B$. Therefore, S_A and S_B do not exchange their level-2 precomputations, but instead send them to the linkage agent.

Using precomputed tables. With $L2_A$ and $L2_B$ available with the linkage agent, it can determine the number of bigrams common in an attribute of two records from sites S_A and S_B. Unlike the protocol in Section 5.2, S_A and S_B now provide lookup indices for their bigram sets, and the linkage agent uses those indices to lookup the precomputed ciphers for the bigrams. It then computes the intersection size of those sets for use in the matching algorithm. The steps of the process are summarized below.

1. S_A has $\alpha = \{b_{\alpha_1}, ..., b_{\alpha_m}\}$ and S_B has $\beta = \{b_{\beta_1}, ..., b_{\beta_n}\}$.
2. For each bigram $b_i \in \alpha$, S_A chooses a $j \in \{1, ..., |KS_A|\}$, and sends $\langle \pi_A^j(i), j \rangle$ to the linkage agent. Similarly, for each bigram $b_i \in \beta$, S_B chooses a $j \in \{1, ..., |KS_B|\}$, and sends $\langle \pi_B^j(i), j \rangle$ to the linkage agent. Let V_A and V_B denote the set of tuples sent by S_A and S_B respectively.
3. For each $(\langle p_A, q_A \rangle, \langle p_B, q_B \rangle) \in V_A \times V_B$, the linkage agent concludes that there is a match if $L2_A(p_B, q_B, q_A) = L2_B(p_A, q_A, q_B)$.
4. $|\alpha \cap \beta|$ is the number of matches found in step 3.

The correctness of the above protocol is based on the observation that $\forall b_i \in \mathcal{B}, L2_A(\pi_B^j(i), j, k) = L2_B(\pi_A^k(i), k, j)$. We revisit the privacy aspect in Section 5.4.

Implementation. We implemented the precomputation with randomly generated permutation functions and a 2048-bit safe prime. Primality is tested using the mpz_probab_prime_p function in the GMP library, which is based on the Miller-Rabin primality test. Primitive roots for the H mapping are obtained by first randomly sampling a number, and then testing if it is a primitive root. For $p = 2q + 1$, an integer $1 \leq g \leq p - 1$ is a primitive root if $g^2 \mod p \neq 1$ and $g^q \mod p \neq 1$. Keys are also generated by first randomly sampling a number, and then testing for co-primality with $p - 1$. With 4761 possible bigrams in $\mathcal{B}$ and $w = 50$, each of $L1_A$ and $L1_B$ will be $4761 \times 50 \times 2048$ bits $= 58.1MB$. Each of $L2_A$ and $L2_B$ will then be $50 \times 58.1MB = 2.8GB$. With data parallelism over eight hardware threads, the level-1 precomputations for each site take approximately 162 seconds, and level-2 precomputations take 2.2 hours in the XUbuntu platform. This timing includes the time required to write the computed values to a file.

While two level-2 precomputation tables can easily fit in memory during record matching, we will not be able to get much advantage from the shared L3 cache of modern multi-core CPUs. It would therefore be advantageous if the precomputed tables can be reduced in size. We employ a bit stripping method where a site determines the minimum number of low-order bits that needs to be retained in its level-2 precomputed values in order to keep them unique, and removes all remaining bits. We observed that retaining the low-order 48 bits (6 bytes) was sufficient to guarantee uniqueness in our precomputed tables, which reduced the size of $L2_A$ and $L2_B$ to $68.1MB$ each. Once again, elliptic curve cryptography may be opted for to improve on the timing of the precomputation and generate smaller sized encryptions. However, neither of the two is a practical concern in our implementation (Pohlig-Hellman cipher with bit stripping).

We convert each of our data sets into a format as would be seen by the linkage agent. In each record, the name and address strings are first converted to sets of bigrams; bigrams are then replaced by their index in $\mathcal{B}$. A key index is chosen for each bigram, and accordingly the corresponding permutation function is applied to the bigram indices. The permuted bigram index and the key index is the final representation of a bigram in the data set. We use 24 bits for this representation–a 11-bit key index and a 13-bit permuted bigram index. A site is assumed to perform a bulk transfer of the transformed representation of its data set in advance to the linkage agent. Hence, the transfers referred to in step 2 of the above protocol are in fact done before the matching process begins. This would be an average transfer of $8.5MB$ for each of our data sets. Appendix A illustrates the various steps involved in the entire workflow.

5.4 Frequency smoothing

Using a fixed key set allows us to precompute the data required for record matching, and significantly decrease the data bandwidth usage. However, it also makes available the distribution characteristics of the bigrams. Figures 3(a) and 3(d) show the frequency of bigrams appearing in the name and street address attributes of one of the (25,10) data sets. Only bigrams with a frequency of more than 1000 are shown. This frequency information is public knowledge in our attacker model. The linkage agent can learn the frequency of different bigrams (from the permuted bigram index) on a per key basis. Permuted bigram indices with two different key indices cannot be correlated since different permutation functions are used across keys. When a site picks a key uniformly at random from its fixed set, the occurrence of bigrams also gets uniformly distributed. As a result, for each key, the linkage agent observes a frequency distribution very similar to the underlying bigram distribution, thereby making the protocol susceptible to a frequency attack.

In order to control this privacy risk, we propose using a frequency smoothing technique. Frequency smoothing will ensure that bigrams reported with a given key index will be indistinguishable from each other based on the frequency of occurrence of permuted bi-

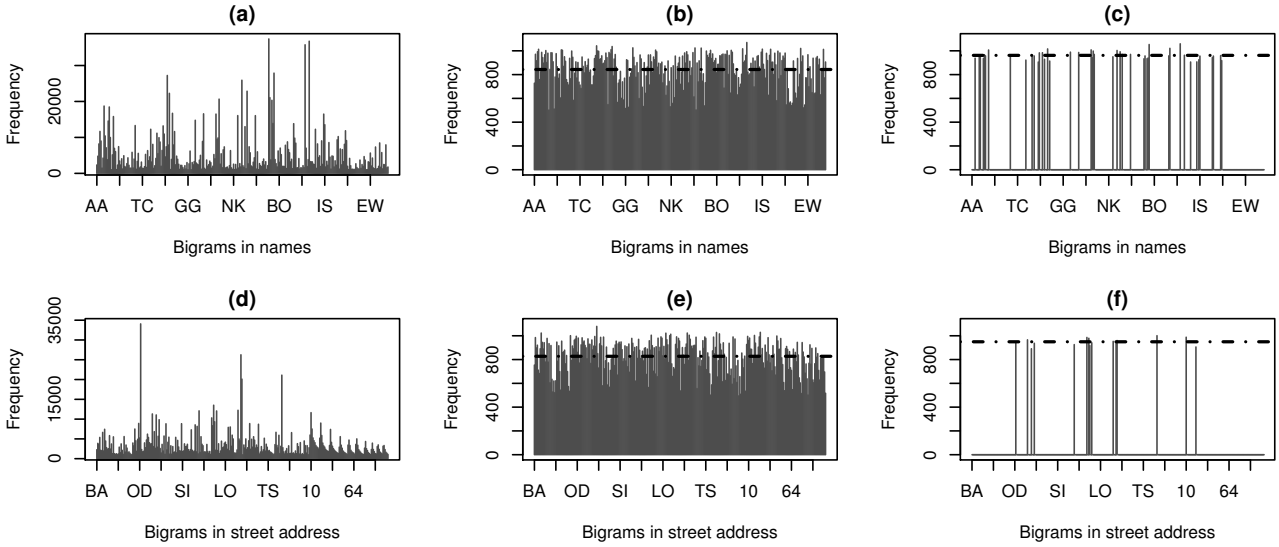


Fig. 3. (a), (d) The raw distribution of bigrams appearing more than 1000 times in the name and street address attributes of one of the (25,10) data sets. (b), (e) The distribution of the same bigrams in key #1. (c), (f) The distribution in key #10. Dashed horizontal line reflects average of non-zero values.

gram indices. To do so, we distribute high frequency bigrams over a large number of keys, and low frequency bigrams over a smaller number. Let f_i be the frequency of bigram $b_i \in \mathcal{B}$ in a given attribute of a data set. Let f_{target} be the frequency we seek to achieve for bigrams reported with a given key index. Note that this is only achievable for bigrams whose frequencies are at least f_{target}. The choice of a key for bigram b_i is then restricted to key numbers 1 through $\lceil \frac{f_i}{f_{target}} \rceil$. As such, a lower numbered key is always more frequently used than a higher numbered one. The minimum number of keys necessary to perform frequency smoothing is then $\lceil \frac{max(f_i)}{f_{target}} \rceil$, which is how the size w of a key set can be decided. For our data sets, we chose $f_{target} = 1000$, which is the minimum frequency of a bigram that appears in at least 1% of the total number of records (=100,000). This produced a key set size of $w = 38$.

Figures 3(b) and 3(e) show the frequency distribution as seen in key 1 when frequency smoothing is used. Bigrams now demonstrate a relatively uniform distribution, with an average frequency of 842.9 and 827.5 in the name and street address attributes respectively. Similar results are observed in other keys as well (key 10 depicted in Figures 3(c) and 3(f)).

5.5 Exposure risk from key indices

Observe that the number of bigrams appearing in key 1 (Figures 3(b) and 3(e)) is much higher than that appearing in key 10 (Figures 3(c) and 3(f)). Depending on the bigram frequency distribution in an attribute, certain keys may get used for a much smaller set of bigrams than others. Therefore, the set of bigrams appearing (or not appearing) with a given key index can be inferred from the key index. If a certain key is used with only one bigram, then use of the key index immediately reveals the presence of the bigram without uncertainty. We therefore compute the expected exposure of each bigram in an attempt to evaluate the privacy risk of the protocol. The expected exposure for a bigram b_i in an attribute is

$$Exp(b_i) = \sum_{j=1}^{w} \Pr(b_i|K^j_{site})\eta(b_i, K^j_{site}),$$

where $\eta(b_i, K^j_{site})$ is the number of times bigram b_i is used with key K^j_{site} at a site, and $\Pr(b_i|K^j_{site})$ is the probability that bigram b_i is in use when key K^j_{site} is in use. The probability is computed using Bayes' rule as

$$\begin{aligned}
\Pr(b_i|K^j_{site}) &= \frac{\Pr(K^j_{site}|b_i)\Pr(b_i)}{\sum_t \Pr(K^j_{site}|b_t)\Pr(b_t)}, \text{where} \\
\Pr(K^j_{site}|b_i) &= \begin{cases} 1/\lceil \frac{f_i}{f_{target}} \rceil & , j \le \lceil \frac{f_i}{f_{target}} \rceil \\ 0 & , otherwise \end{cases}, \text{and} \\
\Pr(b_i) &= f_i / \textstyle\sum_k f_k.
\end{aligned}$$

For percentage expected exposure, we divide $Exp(b_i)$ with the total number of occurrences of b_i ($= f_i$), and then multiply by 100. Figure 4 depicts the

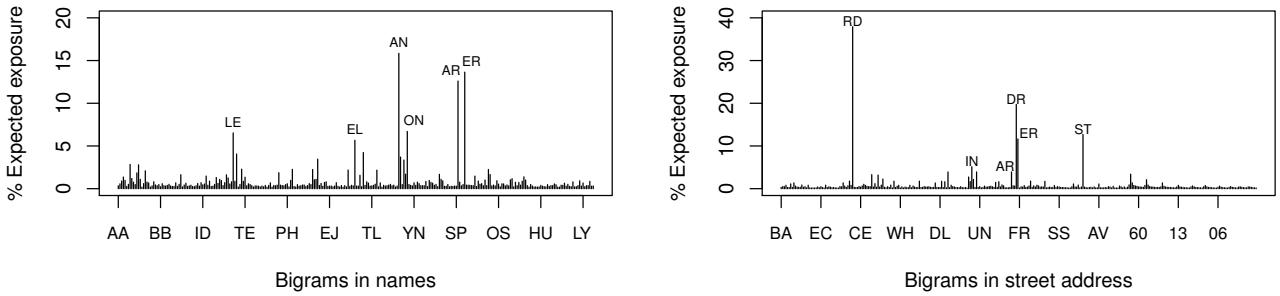


Fig. 4. Expected exposure of bigrams appearing more than 1000 times in one of the (25,10) datasets.

expected exposure of bigrams in the name and street address attributes in one of the data sets. Certain "very" high frequency bigrams have a comparatively higher exposure risk than most other bigrams. For example, the bigram 'RD' has a 37.9% probability of being correctly inferred based simply on the key indices used along with it. This happens because, as a result of the frequency smoothing, key numbers higher than 27 are only used with this bigram. Note that exposure of a few bigrams is not sufficient to reconstruct the underlying strings, unless the strings are composed mostly of high frequency bigrams. In the data sets we use, a small 0.0087 fraction of the strings have at least half (and at most 80%) of their bigrams in the set of ten most frequent bigrams of the data set. Expected exposure is observed to be less than 5% if a bigram is not in the ten most frequent set. Therefore, we consider frequency smoothing to be a viable technique that leaves an acceptable residual risk.

Our approach to estimating the risk from frequency analysis involves finding the exposure probability of each individual bigram using inference techniques based solely on the knowledge of the algorithm and the bigram frequency information. The higher the probability of exposure, the higher is the chance that a bigram will be correctly inferred by an adversary. As such, other bigram based approaches can also estimate the exposure probability within their own context for the purpose of comparative studies.

5.6 Linkage timing

Table 1 lists the wall clock execution times of the record matching algorithm (Section 2.3) when applied to our ten data set pairs. Default processes are running in the background during timing experiments. Recall that the algorithm has quadratic complexity in the number of records (pairwise comparisons). In addition, with the key set approach, every bigram in an attribute of a record has to be compared with every bigram in the attribute of a record on the other site. This makes the algorithm quadratic in the number of bigrams in an attribute's value. Nonetheless, with the precomputations in place, and the reduced size of the precomputed tables, we could compare 100,000 records from one site with 100,000 records from another site in less than two hours. This is irrespective of the extent of overlap or the extent of error present in the data sets. The execution time is higher by a factor of two or more in the older Mac OSX platform.

6 Using A Single Key Per Site

While results so far indicate that record matching on large data sets can be efficiently and privately done using a key set, the benefits of having a single key demands a closer scrutiny. A single key per site approach has two significant advantages.

1. Since there is only one key per site, the precomputed level-2 tables are much smaller in size. With 4761 bigrams and 6 bytes per cipher, each level-2 precomputed table is approximately $28KB$ in size. Therefore, the tables can fit entirely in the L1-cache of a CPU core, resulting in much faster lookups.
2. Since all bigrams from a site are now encoded using the same key, the set intersection size ($|\alpha \cap \beta|$) can be obtained in time linear in the size of the two sets by using a hash table.

Table 1. Time (in hours) to match records using the finite key set approach. $w = 38$ keys are used with frequency smoothing.

	%Overlap, %Error (dataset of 100,000 records at each site)									
	1, 10	*5, 10*	*10, 10*	*15, 10*	*20, 10*	*25, 5*	*25, 10*	*25, 20*	*25, 30*	*25, 50*
Platform-I	**1.51 h**	**1.48 h**	**1.44 h**	**1.40 h**	**1.36 h**	**1.33 h**	**1.33 h**	**1.33 h**	**1.33 h**	**1.32 h**
Platform-II	**3.16 h**	**3.11 h**	**3.02 h**	**2.97 h**	**2.89 h**	**2.91 h**	**2.88 h**	**2.77 h**	**2.76 h**	**2.74 h**

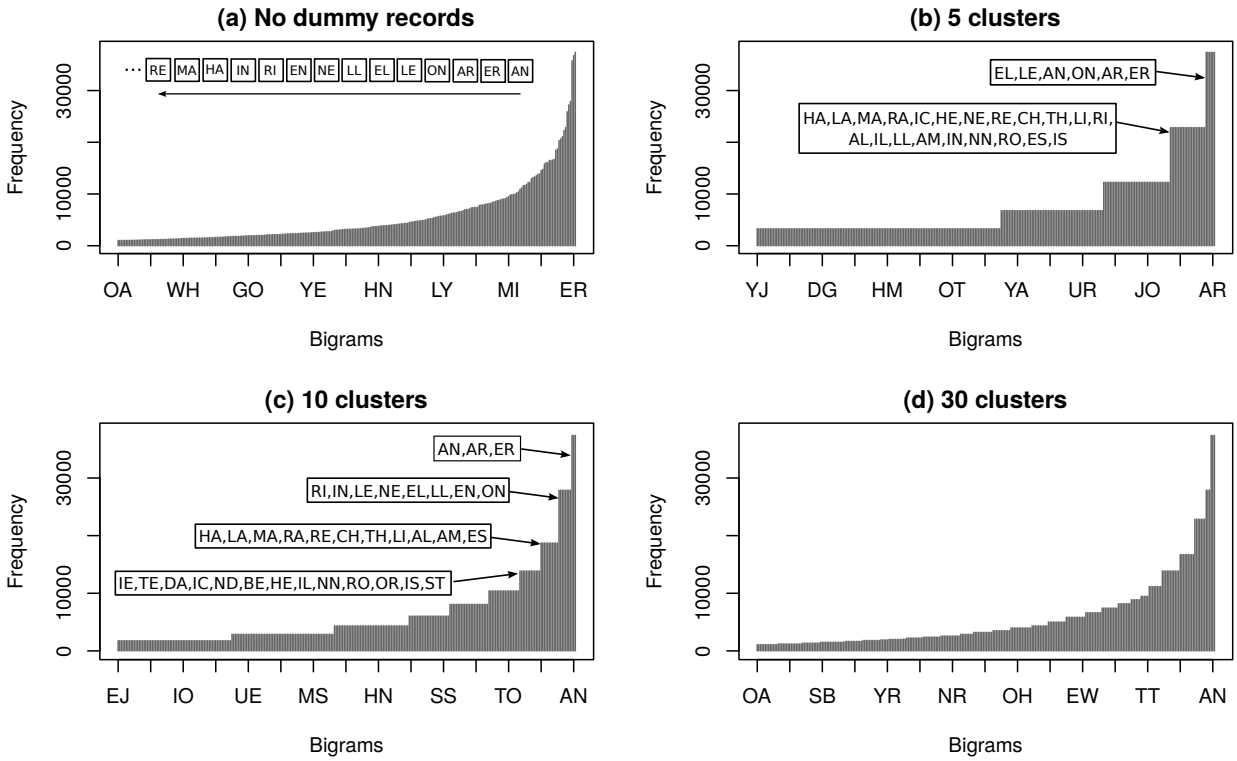


Fig. 5. Frequency distribution of bigrams (that appear at least 1000 times) with and without cluster smoothing and dummy records. Distributions shown are for the name attribute in a (25,10) data set.

However, the downside of the approach is that the bigram distribution in an attribute is exactly revealed.

6.1 Dummy records

One method to smooth out the differences in the frequency is to introduce extra bigrams. However, introducing bigrams in a real record can affect the matching score, thereby affecting the linkage performance. Therefore, extra bigrams have to be introduced in records of their own, which we call *dummy records*. The number of extra bigrams inserted into a dummy record should be close to the average bigram set size of an attribute. For example, on an average, there are 16 bigrams in a name and 13 bigrams in a street address. In an ideal situation, the total number of dummy occurrences to include for a particular bigram b_i should be equal to the difference of its frequency with that of the most frequently occurring bigram, i.e. $\max_j(f_j) - f_i$. By doing so, the ciphers of all bigrams will be seen $\max_j(f_j)$ times during the matching phase, thus producing a flat frequency distribution. Unfortunately, doing so will produce an excessively large number of dummy records because of the heavy-tailed nature of the distribution (Figure 5(a)).

6.2 Cluster smoothing

An alternative method is to introduce dummy bigrams such that a given bigram becomes indistinguishable from a few other bigrams (instead of every other bigram). We start this process by first applying a clustering algorithm (k-means) to divide the bigrams into k clusters based on the proximity of their frequency values. Let C_t denote the set of bigrams in cluster number t. Then, for a bigram $b_i \in C_t$, the number of additional occurrences to be inserted is $f_i^{'} = \max_{b_j \in C_t}(f_j) - f_i$.

Table 2. Number of dummy records generated for a given cluster size in one of the (25,10) data sets.

Number of clusters	Number of dummy sets	
	Name	Street Address
5	37252	29510
10	14825	12630
20	8419	8507
30	5219	6185
50	3528	4422
75	2021	2773
100	1277	1837

This process results in groups of bigram ciphers, with all bigrams in a group having the same frequency of occurrence. Figures 5(b)-(d) show the frequency distribution of the bigrams in the name attribute when dummy records are created after cluster smoothing with 5, 10 and 30 clusters respectively. The plots also show how bigrams get grouped for different cluster sizes.

Our process of creating dummy records is as follows. Once the $f_i^{'}$ values for each bigram are computed for a certain attribute, we create dummy bigram sets for the attribute by sampling bigrams from $\mathcal{B}$ with probabilities proportional to the number of times a bigram remains to be inserted. A new set is started once a predetermined number of dummy bigrams (randomly picked between $\mu - 2$ and $\mu + 2$, where μ is the average number of bigrams in the attribute of a given data set) have been inserted into a set. Ideally, one dummy record is composed of one dummy set for each attribute. However, there is no guarantee that the number of dummy sets will be equal in all attributes of the data set. In that case, we merge the dummy sets of an attribute such that all attributes have the same number of dummy sets. Elements from the largest set are inserted into the remaining sets in increasing order of size, until the number of dummy sets left is equal to the smallest number of dummy sets observed in an attribute. The dummy records are then interspersed into the data set. A site can track dummy records by their record identifiers. Table 2 lists the number of dummy sets that needs to be created to smoothen the frequencies in a cluster. Higher number of clusters allow for smaller groups. Smaller groups result in smaller adjustments in frequency, and therefore produce fewer dummy records.

6.3 Expected exposure

Smaller groups also imply a higher probability of correctly guessing the underlying bigram. For example, in Figure 5(b), the chance that a cipher value corresponds to the bigram 'AN' is $\frac{1}{6}$, whereas that in Figure 5(c) it is $\frac{1}{3}$. The probability of correctly guessing a bigram b_i from the cipher value is therefore $1/|C|$, where b_i is in cluster C. Figure 6 shows the percentage expected exposure of the 200 most frequently occurring bigrams in a name of a (25,10) data set. Clearly, when there are no dummy records, the exposure risk of a bigram depends on how many other bigrams have the same frequency value. As the number of clusters is decreased, the risk in most bigrams starts to decrease as well. Certain high frequency bigrams have comparatively higher exposure risk since k-means grouping was based on simple absolute difference in frequency values. On an average, the top 200 bigrams were found to have an exposure risk that is 4.95 times higher than that in the key set approach. As mentioned earlier, the exposure of a few bigrams is not sufficient to reconstruct the underlying strings, unless the strings are composed mostly of high frequency bigrams. The 90^{th} percentile of the percentage exposure values depicted in the plot is 7.69% for the case of 10 clusters. About 1% of the strings have at least half of their bigrams in the set of 20 most frequent bigrams in this setting.

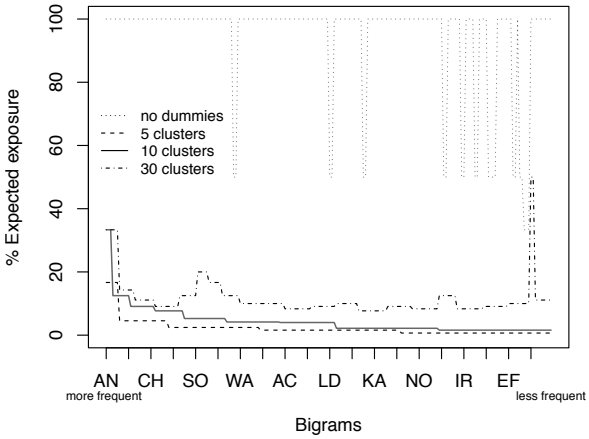


Fig. 6. Expected exposure in top 200 bigrams for names in a (25,10) data set.

It should be noted that including dummy records only during linkage is not sufficient to guarantee privacy. Dummy records ought to appear some times as a result to some query. The encrypted sensitive data in the corresponding result record could include metadata to signify that the record should be ignored, or the query issuer can be provided statistics to help correct inference results.

Given that linkage is performed at an external site, the method of hashing using a shared key can also ben-

Table 3. Time (in minutes) to match records using the single key approach. Number of clusters used = 10.

	%Overlap, %Error (dataset of 100,000 records at each site + ∼ 12,500 dummy records)									
	1, 10	*5, 10*	*10, 10*	*15, 10*	*20, 10*	*25, 5*	*25, 10*	*25, 20*	*25, 30*	*25, 50*
Platform-I	**11.53 m**	**11.32 m**	**11.39 m**	**9.41 m**	**9.64 m**	**10.61 m**	**9.78 m**	**10.56 m**	**8.60 m**	**9.49 m**
Platform-II	**26.69 m**	**26.49 m**	**25.86 m**	**25.09 m**	**24.07 m**	**23.77 m**	**23.72 m**	**23.72 m**	**23.17 m**	**23.51 m**

Table 4. Estimated matching time. *s*: seconds, *m*: minutes, *h*: hours.

No. of records at each site	Number of hardware threads (example processor)		
	12 (E5-2643v3)	*16 (E5-2667v3)*	*20 (E5-2687Wv3)*
1,000	**0.039 s**	**0.029 s**	**0.023 s**
10,000	**3.9 s**	**2.9 s**	**2.3 s**
100,000	**6.5 m**	**4.8 m**	**3.9 m**
500,000	**2.7 h**	**2.0 h**	**1.6 h**
1,000,000	**10.8 h**	**8.1 h**	**6.5 h**

efit from techniques such as permutation of the bigram set and frequency smoothing using dummy records. However, under such a method, different encoded versions of a data set have to be maintained corresponding to the different keys that a party shares with other parties. Our approach is asymmetric in this regard—every site can privately choose its own key(s), and encode its data independent of the party with whom linkage is performed.

6.4 Linkage timing

As hypothesized, using a single key provides significant improvements in the linkage time. Overall, using 10 clusters, the linkage process took less than 12 minutes in Platform-I for every data set pair (Table 3). This is an improvement by a factor of eight over the multi-key approach. These timings also include the unproductive time spent trying to find matches with/for dummy records. Platform-II runs are at least a factor of two slower. The price paid for this significant speed-up is in the (small but positive) increase of the expected exposure risk. Moving forward, we estimated the execution time for other data set sizes, and when a larger number of hardware threads are available (Table 4). The estimation is based on the average duration to match one record pair in Platform-I for the (25,10) data sets. With an average of 16 bigrams in a name and 13 bigrams in a street address, these timings reflect an average of 58 lookups per record pair. The numbers imply that we can perform matching on a trillion ($10^6 \times 10^6$) record pairs in less than 12 hours using an affordable server. Further, we can link two databases with 10 million records in each, and blocked on an average of 100,000 records, in less than 11 hours. This is a conservative extrapolation since we only consider the ratio of the number of available cores in these estimations. Other features such as the per-core cache amount and the I/O bus speed are typically better in servers configured with the example processors; therefore, larger improvements in the linkage timing can be expected.

7 Linkage Performance

We ran the greedy matching algorithm (Section 2.3) to our ten data set pairs. The matching is performed based on the name and street address attributes, and scores are computed using equal weights on the two attributes. A threshold value of $\mathcal{T}_{upper} = 0.7$ is used. For a data set pair, the algorithm reports the identifiers of the record pairs for which a potential match is detected. We determine four sets from this output: (i) true positive (TP): detected matches that are true matches, (ii) true negative (TN): undetected matches that are not true matches, (iii) false positive (FP): detected matches that are not true matches, and (iv) false negative (FN): undetected matches that are true matches. Based on these, we compute two metrics for performance evaluation.

$$Precision = \frac{|TP|}{|TP| + |FP|}$$

$$Recall = \frac{|TP|}{|TP| + |FN|}$$

Precision (or positive predictive value) is the fraction of detected matches that are correct. Recall (or sensitivity) is the fraction of correct matches that the method is able to detect. Table 5 lists the precision and recall values obtained by the algorithm on different data set pairs. We have used the fixed key set approach with $w = 38$ keys to generate the bigram sets. Precision values demonstrate an upward trend as the amount of overlap increases between two data sets. On the other hand, the number of records having errors ($\%error$) do not

Table 5. Precision and recall of greedy matching algorithm on different data set pairs.

	%Overlap, %Error									
	1, 10	*5, 10*	*10, 10*	*15, 10*	*20, 10*	*25, 5*	*25, 10*	*25, 20*	*25, 30*	*25, 50*
Dataset of 100,000 actual records at each site										
Precision	0.592	0.882	0.940	0.964	0.974	0.981	0.981	0.982	0.982	0.984
Recall	0.994	0.993	0.995	0.995	0.994	0.995	0.994	0.992	0.988	0.985
Within the records with input errors										
Precision	0.727	0.901	0.957	0.980	0.986	0.990	0.989	0.989	0.988	0.989
Recall	0.962	0.966	0.980	0.979	0.973	0.972	0.972	0.970	0.969	0.972

seem to have an influence on the precision of the algorithm. In other words, the presence of more records with errors do not lead to spurious matches. Recall values indicate that the algorithm detected almost all matching record pairs. We also looked at the precision and recall values with respect to the subset of records in which an error was inserted. Once again, precision increases with the extent of overlap, but is not influenced by the percentage error. Also, most matching record pairs with errors in them are detected. This would imply that the bigram matching process works quite well for records with input errors. The exact same precision and recall values are obtained when using the single key approach with 10 clusters. In other words, in the single key approach, no matches were found for dummy records. For the small fraction of matching record pairs missed by the algorithm, we observed that most of them either had an entire attribute removed as part of the error insertion process, or had significant changes been made to an attribute's value as a result of multiple error insertions. We emphasize that validation of linkage results is a crucial step for the matches to be admissible [12]. Although difficult to perform across sites due to privacy concerns, special regulatory permissions are sought to perform a small scale validation.

8 Tracking Linked Records

The task of a linkage agent is to identify data records distributed across multiple sites but belong to the same individual. It needs to be invoked when new sites enter the data exchange system, or a new patient record is entered into an already linked site. Linkage information is also required to join distributed records during query processing. We briefly discuss the role of the linkage agent in each of these situations. We assume that sites have a pre-decided set of attributes on which linkage is to be performed.

8.1 Site entry

Prior to using data from a site, its records need to be linked with records from other participating sites. This linkage is performed with every site for which there is a possibility of record overlap, which are typically sites in the same region. The new site and the sites with which linkage is necessary have to perform the necessary precomputations in preparation for the linkage. The new site provides the encoded (permuted bigram indices) attribute values to the linkage agent. Once the precomputation results are made available to the linkage agent, the record matching can happen. The output produced by the matching process can be used in one of two ways.

Cross-site linkage identifier. Matched record pairs can be assigned the same linkage identifier across sites. If a record on one site has already been assigned a linkage identifier, it is reused as an identifier for the matched record on the new site. Records at the new site that are not matched with any record on any other site are assigned new linkage identifiers. As sites undergo pairwise linking, the linkage identifiers propagate and become the primary key for the distributed records. Therefore, if a site gains knowledge of the presence of an identifier at another site, it can infer the identity of the patient at the other site. This membership information may be sensitive information, not authorized for disclosure. The linkage agent can also learn that a patient visits two sites, but the identity of the patient is hidden from the agent. Another issue with cross-site identifiers is that every site has to be updated in the event that an identifier has to be refreshed.

Per-site linkage identifier. In this scheme, the linkage agent assigns a unique linkage identifier to each record, but retains the association between them as equivalence sets. Identifier refresh only involves an update with the relevant site. Per-site identifiers can also help prevent the inference of site membership from

leaked identifiers. However, the association data available at the linkage agent becomes critical to track the distributed records during query processing.

8.2 Record entry

Linkage invocations due to new site entries will be rare after initial system setup. Comparatively, new records will be created more frequently. All entries corresponding to one individual are maintained as one record. When a new record is created, its encoded attributes are sent to the linkage agent as an update operation to the data it communicated during the first linkage. The linkage agent attempts to match the new record with other sites. If a match is found, then an identifier is assigned (for the cross-site scheme, the matched record's identifier is used), and the process stops; otherwise a new identifier is assigned. Gruenheid at al. propose using incremental graph clustering to handle update operations on linked databases [21].

8.3 Joining records

A crucial task of the regional federated query processor is to join results from multiple data sources, either to remove duplicate counts, or answer cross-site queries. After the query processor collects the query results from individual sites, it accesses the linkage results to perform this join. This is trivially done if cross-site linkage identifiers are in use. Otherwise, the query processor communicates with the linkage agent to determine which linkage identifiers correspond to the same record. Records resulting after the join can be assigned different identifiers for use by the querying user. Such identifiers will also allow the user to refer back to an individual's data during a longitudinal study.

9 Conclusion

The detection of distributed health records is important to resolve data duplication and facilitate complex join queries. However, the presence of strict regulations on the sharing of patient identifiers, as well as inconsistencies in the data, make this a non-trivial task. Methods utilizing hash functions and basic Bloom filters have gained popularity for their ease of use, albeit their privacy guarantees are weakly founded. On the other hand, cryptographic methods are known for their strong guarantees, but have been claimed unsuitable for linking large health databases because of high computation and communication costs. Through this work, we have invalidated such claims and shown that by precomputing repetitive cryptographic operations, parallelizing computations over a small number of hardware threads, and truncating large encryption outputs, it is in fact possible to execute a quadratic record matching algorithm on data sets as large as 100,000 records each, and obtain results in less than 10 minutes with insignificant communication cost. Our methods do not require sharing of secret keys, and make frequency attacks futile by smoothing frequencies or introducing dummy records. We hope that these feasibility results will prompt the adoption of cryptographic primitives in the design of fast and private record linkage components.

In terms of future work, extension of our techniques to Internet-scale linkage will be an interesting direction. We have provided evidence that large data sets (even up to a million records per site) can be handled within a reasonable time frame. Nonetheless, we assume that linking is done pairwise between sites. At the Internet-scale, where the number of records or the number of sites can be much larger, it will be worthwhile to explore if the proposed techniques can be extended for multiparty linking with sub-quadratic complexity. We are also yet to apply our technique to link actual medical data sets, where decisions on the quantity and quality of attributes necessary for productive linking has to be made. As an immediate extension, we plan to asses the suitability of Elliptic curves in our techniques with the objective of further improving the linkage time.

Acknowledgement

We thank Dr. Michael Kahn at the Anschutz Medical Campus (University of Colorado-Denver) for providing insights into the SAFTINet infrastructure, the data duplication problem, and the privacy hurdles faced by the medical informatics community in record linkage.

References

[1] N. Adam, T. White, B. Shafiq, J. Vaidya, and X. He. Privacy preserving integration of health care data. In *AMIA Annual Symposium Proceedings*, pages 1–5, 2007.

[2] N. Adly. Efficient record linkage using a double embedding scheme. In *International Conference on Data Mining*, pages

274–81, 2009.

[3] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD International Conference on Management of Data*, pages 86–97, 2003.

[4] H. Brenner. Application of capture-recapture methods for disease monitoring: Potential effects of imperfect record linkage. *Methods of Information in Medicine*, 33(5):502–506, 1994.

[5] P. Christen. Probabilistic data generation for deduplication and data linkage. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 109–116, 2005.

[6] P. Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 1065–68, 2008.

[7] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[8] T. Churches and P. Christen. Blind data linkage using n-grams similarity comparisons. In *Advances in Knowledge Discovery and Data Mining*, pages 121–126, 2004.

[9] T. Churches and P. Christen. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making*, 4:9, 2004.

[10] E. Durham, Y. Xue, M. Kantarcioglu, and B. Malin. Private medical record linkage with approximate matching. In *AMIA Annual Symposium Proceedings*, pages 182–186, 2010.

[11] E. A. Durham et al. Composite bloom filters for secure record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2956–2968, 2013.

[12] S. B. Dusetzina et al. Linking data for health services research: A framework and instructional guide. Technical Report 14-EHC033-EF, Agency for Healthcare Research and Quality (US), 2014.

[13] L. Dusserre, C. Quantin, and H. Bouzelat. A one way public key cryptosystem for the linkage of nominal files in epidemiological studies. *MedInfo*, 8 (Pt 1):644–647, 1995.

[14] S. Duvall, R. Kerber, and A. Thomas. Extending the Fellegi-Sunter probabilistic record linkage method for approximate field comparators. *Journal of Biomedical Informatics*, 43(1):24–30, 2010.

[15] I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[16] J. T. Finnell. In support of emergency department health information technology. In *AMIA Annual Proceedings Symposium*, pages 246–250, 2005.

[17] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–19, 2004.

[18] S. J. Grannis, J. M. Overhage, S. Hui, and C. J. McDonald. Analysis of a probabilistic record linkage technique without human review. In *AMIA Annual Symposium Proceedings*, pages 259–63, 2003.

[19] S. J. Grannis, J. M. Overhage, and C. McDonald. Analysis of identifier performance using a deterministic linkage algorithm. In *AMIA Annual Symposium Proceedings*, pages 305–309, 2002.

[20] S. J. Grannis, J. M. Overhage, and C. McDonald. Real world performance of approximate string comparators for use in patient matching. *Studies in Health Technology and Informatics*, 107(Pt 1):43–7, 2004.

[21] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708, 2014.

[22] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco. A hybird approach to private record linkage. In *International Conference in Data Engineering*, pages 496–505, 2008.

[23] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *International Conference on Extending Database Technology*, pages 123–134, 2010.

[24] A. Karakasidis and V. S. Verykios. Privacy preserving record linkage using phonetic codes. In *Balkan Conference in Informatics*, pages 101–106, 2009.

[25] A. Karakasidis and V. S. Verykios. Secure blocking + Secure matching = Secure record linkage. *Journal of Computing Science and Engineering*, 5(3):223–235, 2011.

[26] M. Kuzu, M. Kantarcioglu, E. Durham, and B. Malin. A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In *International Conference on Privacy Enhancing Technologies*, pages 226–245, 2011.

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

[28] D. V. LaBorde, J. A. Griffin, H. K. Smalley, P. Keskinocak, and G. Matthew. A framework for assessing patient crossover and health information exchange value. *Journal of American Medical Informatics Association*, 18(5):698–703, 2011.

[29] I. Lazrig et al. Privacy preserving record matching using automated semi-trusted broker. In *Annual Working Conference in Data and Applications Security and Privacy*, pages 103–118, 2015.

[30] B. Malin and E. Airoldi. Confidentiality preserving audits of electronic medical record access. *Studies in Health Technology and Informatics*, 129(1):320–324, 2007.

[31] F. Niedermeyer, S. Steinmetzer, M. Kroll, and R. Schnell. Cryptanalysis of basic bloom filters used for privacy preserving record linkage. *Journal of Privacy and Confidentiality*, 6(2):59–79, 2014.

[32] B. Pinkas, T. Schneider, and M. Zoner. Faster private set intersection based on ot extension. In *23rd USENIX Conference on Security Symposium*, pages 797–812, 2014.

[33] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[34] S. M. Randall, A. M. Ferrante, J. H. Boyd, J. K. Bauer, and J. B. Semmens. Privacy-preserving record linkage on large real world datasets. *Journal of Biomedical Informatics*, 50:205–212, 2014.

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

[36] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In *ACM SIGMOD International Conference on Management of Data*, pages 653–64, 2007.

[37] L. M. Schilling et al. Scalable Architecture for Federated Translational Inquiries Network (SAFTINet) technology infrastructure for a distributed data network. *eGEMs (Generating Evidence & Methods to improve patient outcomes)*, 1(1):1027, 2013.

[38] K. Schmidlin, K. M. Clough-Gorr, and A. Spoerri. Privacy Preserving Probabilistic Record Linkage (P3RL): A novel method for linking existing health-related data and maintaining participant confidentiality. *BMC Medical Research Methodology*, 15(46):open access, 2015.

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

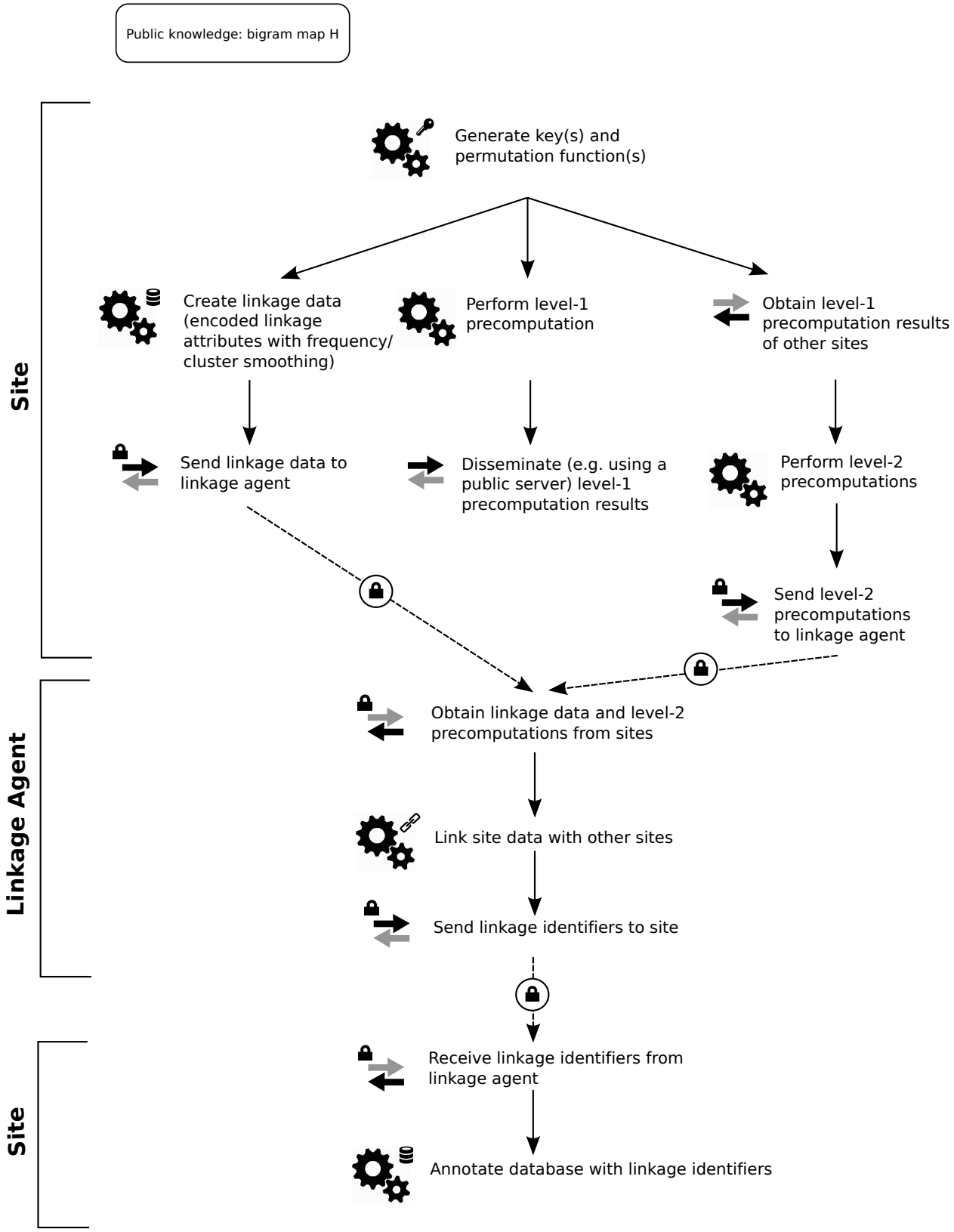
[40] X. S. Wang et al. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 492–503, 2015.

[41] G. M. Weber. Federated queries of clinical data repositories: The sum of the parts does not equal the whole. *Journal of American Medical Informatics Association*, 20:e155–e161, 2013.

[42] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau of the Census, 1999.

[43] M. Yakout, M. J. Atallah, and A. K. Elmagarmid. Efficient private record linkage. In *International Conference in Data Engineering*, pages 1283–1286, 2009.

Appendix A: Private linkage workflow



Appendix B: Data sets used in recent studies related to private record linkage

Grannis et al. [18–20]	Patient registry records from two hospital registries in Indiana, and linked to a subset of the Death Master File; 6,000 record pairs
Agrawal et al. [3]	No experimental evaluation; computation and communication cost is estimated
Churches and Christen [8, 9]	No experimental evaluation; states that the method incurs very high data transmission overheads, which the authors believe can be handled by modern high-bandwidth research networks
Ravikumar et al. [35]	Cora (http://www.cs.umd.edu/~sen/lbc-proj/data/cora.tgz) dataset consisting of 2,708 scientific publications and 5,429 links
Scannapieco et al. [36]	British Columbia voter's list (1,000 records); personal and business data maintained by an Italian public administration agency, with two tables of 7,846 and 7,550 records in the former, and 20,000 records in the latter; duplicates are inserted artificially
Inan et al. [22, 23]	Census-Income Adult data set from UCI machine learning repository; experiment performed on two subsets of the data, with 20,108 records in each; subsets are generated such that overlap is known
Adly [2], Yakout et al. [43]	British Columbia voter's list; Adly used datasets with 4,000, 10,000 and 20,000 records, generated by sampling from the list; manually controlled and identified the percentage of similar records between each set pair
Schnell et al. [39]	Two German private administration databases, each with about 15,000 records
Durham et al. [10]	Created 100 datasets with 1,000 records in each from the identifiers and demographics within the patient records in the electronic medical record system of the Vanderbilt University Medical Center; data sets to link to are generated from these 100 sets using a "data corrupter"
DuVall et al. [14]	Used the enterprise data warehouse of the University of Utah Health Sciences Center; 118,404 known duplicate record pairs, identified using the Utah Population Database
Karakasidis et al. [25]	Used the FEBRL synthetic data generator [6] for performance and accuracy experiments
Kuzu et al. [26]	A sample of 20,000 records from the North Carolina voter's registration list; to evaluate the effect of typographical and semantic name errors, the sample was synthetically corrupted
Durham et al. [11]	Ten independent samples of 100,000 records from the North Carolina voter's registration list; each sample was independently corrupted to generate samples at the second party
Dusetzina et al. [12]	Individuals in the North Carolina Central Cancer Registry (NCCCR) diagnosed with colon cancer linked to enrollment and claims data for beneficiaries in privately insured health plans in North Carolina; 104,360 record pairs
Gruenheid et al. [21]	Cora dataset; Biz dataset consisting of multiple versions of a business records dataset, each with 4,892 records
Randall et al. [34]	approximately 3.5×10^9 record pairs from ten years of the West Australian Hospital Admissions data; approximately 16×10^9 record pairs from ten years of the New South Wales admitted patient data
Schmidlin et al. [38]	No experimental evaluation; timing estimated for a linkage attempt with 100,000 records in one data set and 50,000 records in another